Monotonicity in Markov Models

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Jip Spel, M. Sc.
aus Blaricum

Berichter: Prof. Dr. Ir. Dr. h.c. Joost-Pieter Katoen
Jun.-Prof. Dr. Tatjana Petrov

Tag der mündlichen Prüfung: 07-07-2023

Monotonicity in Markov Models
Jip Spel
Submitted: February 2023
Defended: July 2023
Published: September 2023

Advisors:
Prof. Dr. Ir. Dr. h.c. Joost-Pieter Katoen
Examiners:
Jun.-Prof. Dr. Tatjana Petrov

Abstract

Many systems exhibit probabilistic behavior, such as randomized protocols, communication protocols, or biological systems. Probabilistic model checking is a common way to analyze these type of systems. We often model these systems as Markov models and check them against a specification typically given in a probabilistic extension of LTL or CTL. One of the main goals is to analyze the probability to reach or the expected total reward upon reaching a set of target states. When checking for these properties, we assume probabilities to be fixed constants. In practice, however, these probabilities are often not fixed but bounded to a given interval. Moreover, these probabilities can be dependent on other probabilities. Therefore, we consider parametric Markov models, in which the probabilities are given symbolically by rational functions over parameters.

We focus on parametric Markov chains (pMCs) and parametric Markov decision processes (pMDPs). Both models have a finite set of states, of which one is the initial state, and one or more are the target states. The transitions are given by a parametric transition function. PMCs have a deterministic transition function, whereas for pMDPs the transition function might be non-deterministic.

One problem addressed when analyzing parametric Markov models is the almost-optimal synthesis problem, i.e., finding a parameter instantiation such that the expected total reward is ε-close to the real *unknown* optimal expected total reward. This parameter instantiation must lay within a given region, which is a subset of the parameter space. A common technique to tackle this almost-optimal synthesis problem is by checking rational functions obtained by state elimination. These functions are known to be exponential in the number of parameters and become infeasible for more than two parameters. Parameter lifting is an approach which uses an abstraction technique to overcome this problem. It replaces parameter values with a non-deterministic choice over the extremal values. This gives lower and upper bounds on the probabilities or the expected total reward of a region in the parameter space. We now pick a parameter instantiation and check the value of the solution function at this instantiation. If the found solution is ε-close to such a bound, we have found a solution to the almost-optimal synthesis problem, otherwise, the region has to be split. The drawback of parameter lifting is an exponential blow-up in the regions we need to check, due to this region splitting, making it only feasible for a handful of parameters.

In this dissertation, we introduce and define the monotonicity of parametric Markov models. We observe that many systems are monotonic in one or more parameters, e.g., if we increase the probability of a parameter, the expected total reward also increases. We use this property to deal with the almost-optimal synthesis problem and improve parameter lifting with this. In order to do so, we, first of all, define monotonicity on pMCs, and show how it can be computed by using a partial order on the states. We then lift this approach to pMDPs. We implement the monotonicity checking for

pMCs and pMDPs in the model checker Storm and experimentally show that our approach can automatically detect monotonicity. Furthermore, we explain our novel algorithmic approach in which we combine monotonicity checking and parameter lifting for pMCs, and experimentally show that we significantly improve the time to determine almost-optimal synthesis.

Another problem we deal with is the feasibility problem for parametric Markov models, i.e., finding parameter instantiations that reach the target with at least a given probability (or expected total reward). Techniques tackling the feasibility problem include particle swarm optimization (PSO), quadratically-constrained quadratic program (QCQP), and sequential convex programming (SCP). We consider the well-known gradient descent method to deal with the feasibility problem. This approach is suitable for pMCs as, first of all, analyzing the gradient of the rational function of a pMC can be done efficiently, without calculating the entire function. Secondly, in pMCs, many parameters may occur and gradient-based search methods scale to models with a large number of parameters. We deal with several classical and adaptive gradient-based methods and compare them to the state-of-the-art methods PSO, QCQP, and SCP. The resulting method, as implemented in Storm, scales to significantly larger pMCs than before and empirically outperforms PSO and QCQP, often by at least one order of magnitude. Furthermore, it performs akin to SCP.

Finally, we focus on parametric probabilistic timed automata (pPTA). Probabilistic timed automata (PTA) combine the features of timed automata with those of MDPs. They are often used to analyze communication protocols and networked systems. As for other Markov models, they assume probabilities to be known. We extend PTAs to pPTAs, such that the probabilities can be unknown. Furthermore, we show that some techniques to reduce PTAs to MDPs also work for pPTAs. An implementation of this reduction for pPTAs is available in the Modest Toolset. We check for monotonicity in the obtained pMDPs.

Zusammenfassung

In vielen Systemen treten Formen von Warscheinlichkeiten auf, unter anderem in Kommunikationsprotokollen, Zufallsprotokollen und biologischen Systemen vor. Solche Systeme können wir als Markow-Modelle modellieren, und anhand einer Spezifikation in (einer probabilistischen Erweiterung der) lineare temporalen Logik (LTL) oder baumbasierten temporalen Logik (CTL) verifizieren. Das Ziel ist die Berechnung der Wahrscheinlichkeit oder der erwarteten Gesamtkosten für das Erreichen eines der Teilzuständen. Die Wahrscheinlichkeiten in einem Markow-Modell werden oft als ein exakter Wert angegeben. In der Praxis sind diese Wahrscheinlichkeiten jedoch nicht genau bekannt, sondern liegen in einem Intervall. Außerdem können diese Wahrscheinlichkeiten von anderen Wahrscheinlichkeiten abhängen. Deshalb konzentrieren wir uns auf parametrische Markow-Modelle, in denen die Wahrscheinlichkeiten symbolisch durch Parameter in rationalen Funktionen dargestellt werden.

In dieser Arbeit konzentrieren wir uns auf parametrische Markow-Ketten (pMCs) und parametrische Markow-Entscheidungsprozesse (pMDPs). Beide Arten von Markow-Modellen haben eine endliche Menge von Zuständen, von denen einer den Ausgangszustand darstellt, und einer oder mehrere Zielzustände sind. Die Übergänge zwischen den Zuständen werden mit einer parametrischen Wahrscheinlichkeitsfunktion gekennzeichnet. Bei pMCs ist diese Funktion für alle Zustände deterministisch; bei pMDPs kann diese Funktion auch für einige Zustände nicht-deterministisch sein.

Eines der Probleme, das wir in dieser Arbeit analysieren, ist das sogenannte ε-optimale Syntheseproblem. Hier besteht das Ziel darin, einen Wert für die Parameter zu finden, sodass die erwartete Gesambelohnung für diese Werte so nah wie möglich am *unbekannten* Optimum ist. Eine Möglichkeit, dieses Problem zu lösen, besteht darin, die rationale Funktion zu berechnen, indem man die Zustände im Modell eliminiert. Diese Funktion ist jedoch exponentiell zur Anzahl der Variablen, wodurch dieser Ansatz für Modelle mit mehr als zwei Parametern nicht praktikabel ist. Ein anderer Ansatz ist das sogenannte Parameter-Lifting. Dabei handelt es sich um eine Abstraktionstechnik, bei der jeder Parameter durch eine nicht-deterministische Wahl zwischen den Extremwerten für den Parameter ersetzt wird. Auf dieser Weise erhalten wir eine untere und obere Grenze für die erwartete Gesamtkosten des Modells für einen gegebenen Parameterbereich. Wenn wir nun solche Parameterwerte finden, sodass die erwartete Gesamtkosten höchstens ε von einem solchen Grenzwert entfernt ist, haben wir eine Lösung für das ε-optimale Syntheseproblem gefunden. Andernfalls müssen wir die Parameterregion aufteilen und die Grenzen für jede Teilregion berechnen. Ein Nachteil dieses Ansatzes ist, dass wir eine exponentielle Anzahl von Teilregionen erhalten können.

In dieser Arbeit führen wir die Monotonie von parametrischen Markow-Modellen ein und definieren sie. In vielen Systemen sehen wir, dass, wenn wir die Wahrscheinlichkeit eines Parameters erhöhen, auch die erwartete Gesamtkosten steigt. Wir nutzen

diese Eigenschaft, um das Problem der ε-optimalen Synthese anzugehen und so die Parametersynthese zu verbessern. Zunächst definieren wir die Monotonie für pMCs und zeigen, wie man sie mit Hilfe einer partiellen Ordnung auf den Zuständen berechnen kann. Als nächstes erweitern wir diesen Ansatz auf pMDPs. Anhand einer Implementierung im Model Checker Storm zeigen wir, dass unser Ansatz automatisch Monotonie für pMCs und pMDPs berechnen kann, ohne dass zuvor die rationale Funktion berechnet werden muss. Außerdem kombinieren wir Parameter-Lifting und Monotonie für pMCs und zeigen anhand von Experimenten, dass wir das ε-optimale Syntheseproblem mit Hilfe dieses kombinierten Ansatzes schneller lösen können.

Ein weiteres Problem, mit dem wir uns in dieser Arbeit befassen, ist das Erfüllungsproblem (feasiblity) für parametrische Markow-Modelle. Dabei suchen wir Parameterwerte mit denen wir das Ziel mit mindestens einer bestimmten Wahrscheinlichkeit (oder erwarteten Gesamtkosten) erreichen. Wir verwenden verschiedene Gradientenmethode aus der Literatur und zeigen, wie man das Erfüllungsproblem mit diesen Methoden lösen kann. Wir analysieren mehrere klassische und adaptive Methoden und vergleichen sie mit den existierenden Methoden Partikelschwarmoptimierung (PSO), quadratisch eingeschränkten quadratischen Programmen (QCQP) und sequentieller konvexer Programmierung (SCP). Die verschiedene Gradientenmethode sind in Storm implementier. Die beste Gradientenmethode skaliert auf wesentlich größere pMCs als zuvor und übertrifft empirisch PSO und QCQP, oft um mindestens eine Größenordnung. Im Vergleich zu SCP schneidet die resultierende Methode ähnlich gut ab.

Zum Schluss konzentrieren wir uns auf eine parametrische Erweiterung von probabilistischen Zeitautomaten (PTA). Diese PTA kombinieren die zeitlichen Aspekte von zeitgesteuerten Automaten (TA) mit MDPs. PTAs werden z.B. zur Analyse von Kommunikationsprotokollen und Netzwerksystemen verwendet. Wie bei anderen Markow-Modellen auch, wird vorausgesetzts, dass die Wahrscheinlichkeiten im PTA vorgegeben sind. Wir erweitern PTAs zu parametrischen PTAs, sodass die Wahrscheinlichkeiten in einem bestimmten Intervall liegen und voneinander abhängig sein können. Die Semantik eines pPTA ist ein unendlicher pMDP. Wir analysieren einige bestehende Techniken für PTA und verwenden sie, um einen endlichen pMDP aus einem pPTA zu extrahieren. Diese Funktion ist im Modest Toolset tool implementiert. Um zum Beispiel die Monotonie in einem pPTA zu bestimmen, können wir nun das zugrundeliegende pMDP mit den in dieser Arbeit vorgestellten Techniken analysieren.

Samenvatting

Onder andere in communicatieprotocollen, gerandomiseerde protocollen en biologische systemen komt probabilistisch gedrag voor. Dit soort systemen kunnen we modelleren als Markov modellen en toetsen tegen een specificatie in (een probabilistische uitbreiding van) lineaire temporele logica (LTL) of boomgebasseerde temporele logica (CTL). Het doel is om de waarschijnlijkheid van of de verwachtte totale beloning voor het bereiken van een reeks deeltoestanden te berekenen. De kansen in een Markov model zijn vaak gegeven als een specifieke waarde. In de praktijk zijn deze kansen echter niet een specifiek waarde, maar bevinden ze zich in een gegeven interval. Daarnaast kunnen deze kansen afhankelijk zijn van andere kansen, daarom richten we op parametrische Markov modellen, waarin de kansen symbolisch worden weergegeven door rationele functies over parameters.

In dit proefschrift richten we ons op parametrische Markov ketens (pMCs) en parametrische Markov beslisprocessen (pMDPs). Beide type Markov modellen hebben een eindige reeks toestanden, waarvan er één de begintoestand weergeeft, en er een of meer doeltoestanden zijn. De transities tussen de toestanden zijn gelabeld met een parametrische kansfunctie. Bij pMCs is deze functie voor alle toestanden deterministisch, bij pMDPs, kan deze functie voor een toestand ook niet-deterministisch zijn.

Een van de problemen die we in dit proefschrift analyseren is het ε-optimale synthese probleem, waarbij het doel is om een waarde voor de parameters te vinden, zodat de verwachtte totale beloning voor deze waarde maximaal ε van de *onbekende* optimale verwachtte totale beloning af ligt. Een manier om dit probleem aan te pakken, is om de rationele functie te berekenen, door de toestanden in het model te elimineren. Deze functie is echter exponentieel in het aantal variabelen, waardoor deze aanpak onuitvoerbaar is voor modellen met meer dan twee parameters. Een andere aanpak is parameter lifting, dit is een abstractie techniek waarbij elke parameter vervangen wordt door een niet-deterministische keuze tussen de extreme waarde voor de parameter. Op deze manier krijgen we een onder- en bovengrens voor de verwachtte totale beloning van het model voor een parametergebied. Door nu willekeurig een waarde in het parametergebied te kiezen, kunnen we kijken of de verwachtte totale beloning maximaal ε van de berekende grens afligt. Als dit het geval is hebben we een oplossing voor het ε-optimale synthese probleem gevonden. Anders moeten we het parametergebied splitsen, en voor elk subdomein de grenzen berekenen. Een nadeel van deze aanpak is dat we een exponentieel aantal subdomeinen kunnen krijgen.

In dit proefschrift introduceren en definiëren we de monotoniciteit van parametrische Markov modellen. In veel systemen zien we dat als we de kans van een parameter verhogen, dat ook de verwachtte totale beloning toeneemt. Deze eigenschap gebruiken we om het ε-optimale syntheseprobleem aan te pakken en daarmee parameter lifting te verbeteren. Allereerst definiëren we monotoniciteit voor pMCs en laten zien hoe we deze kunnen berekenen met behulp van een preorde van de toestanden. Vervolgens,

breiden we deze aanpak uit naar pMDPs. Met behulp van een implementatie in de modelchecker Storm, laten we via experimenten zien dat onze aanpak automatisch monotoniciteit voor pMCs en pMDPs kan berekenen zonder eerst de rationele functie te berekenen. Verder combineren we parameter lifting en monotoniciteit voor pMCs, en laten we met behulp van experimenten zien dat we met deze gecombineerde aanpak sneller het ε-optimale synthese probleem kunnen oplossen.

Een ander probleem dat we behandelen in dit proefschrift is het vervulbaarheidsprobleem voor parametrische Markov modellen, namelijk het vinden van parameter waardes, zodat we het doel bereiken met ten minste een gegeven waarschijnlijkheid (of verwachtte totale beloning). We gebruiken een gradiëntmethodes uit de literatuur en laten zien hoe we met behulp van deze methodes het vervulbaarheidsprobleem kunnen oplossen. We analyseren diverse klassieke en adaptieve methoden, en vergelijken deze met de bekende methodes particle swarm optimization (PSO), quadratically-constrained quadratic program (QCQP) en sequential convex programming (SCP). De resulterende methode, zoals geïmplementeerd in Storm, schaalt naar aanzienlijk grotere pMC's dan voorheen en presteert empirisch beter dan PSO en QCQP, vaak met minstens één orde van grootte. In vergelijking met SCP presteert de resulterende methode hetzelfde.

Ten slotte richten we ons op een parametrische uitbreiding van probabilistische getimede automaten (PTA). Deze PTA combineren de tijdsaspecten van getimede automaten met MDPs. Ze worden gebruikt om communicatieprotocollen en netwerksystemen te analyseren. Net als bij andere Markov modellen, zijn de kansen in de PTA gegeven, daarom breiden we ze uit naar parametrische PTA, zodat de kansen zich in een gegeven interval kunnen bevinden en afhankelijk van elkaar kunnen zijn. De semantiek van een pPTA is een oneindige pMDP, wij analyseren een aantal bestaande technieken voor PTA en gebruiken deze om een eindige pMDP uit een pPTA te extraheren. De implementatie hiervan is beschikbaar in de Modest Toolset. Om bijvoorbeeld monotoniciteit in een pPTA vast te stellen, kunnen we nu de onderliggende pMDP analyseren met de techniek die in dit proefschrift geïntroduceerd wordt.

Gearfetting

Ûnder oare yn kommunikaasjeprotokollen, randomisearre protokollen en biologyske systemen komt probabilistysk gedrach foar. Dit soarte fan systemen kinne we modellearje as Markov modellen en toetse tsjin een spesifikaasje yn (een probabilistyske utwreiding fan) lineare temporele logika (LTL) of beam-basearre temporele logika (CTL). It doel is om de wierskynlikheid fan de ferwachtte totale beleaning foar it berikken fan in reeks dieltastannen te berekkenjen. De kânsen yn in Markov model binne faaks jûn as in spesifieke wearde. Yn de praktyk binne dizze kânsen lykwols net in spesifieke wearde, mar befine se sich yn in bepaald ynterval. Derneist kinne dizze kânsen ôfhinklik wêze fan oare kânsen, dêrom rjochtsje we ús op parametryske Markov modellen, wêryn de kânsen simboalysk werjûn wurde troch parameters yn rasjonele funksjes.

Yn dit proefskrift rjochtsje we ús op parametryske Markov ketens (pMCs) en parametryske Markov beslisprosessen (pMDPs). Beide types Markov modellen hawwe in einige rigel tastannen, wêrfan der ien de begjintastân werjout, en der ien of mear dieltastannen binne. De transysjes tusken de tastannen binne labele mei in parametryske kânsfunksje. By pMCs is dizze funksje foar alle tastannen deterministysk, by pMDPs, kin dizze funksje foar in tastân ek net-deterministysk wêze.

Ien fan de swierricheden dy't we yn dit proefskrift analysearje is it benei-optimale syntheseprobleem, wêrby it doel is om in wearde foar de parameters te finen, sadat de ferwachtte totale beleaning foar dizze weardes sa ticht mogelik by de ûnbekende ferwachtte totale beleaning is. In wize om dit probleem oan te pakken, is om de rasjonele funksje te berekkenjen, troch de tastannen yn it model te eliminearjen. Dizze funksje is lykwols eksponinsjeel yn it oantal fariabelen, wêrtroch dizze oanpak ûnútfierber is foar modellen mei mear dan twa parameters. In oare oanpak is parameter lifting, dit is in abstraksje technyk werby eltse parameter ferfongen wurdt troch in net-deterministyske kar tusken de ekstreme wearde foar de parameter. Op dizze wize krije we in ûnder- en boppegrins foar de ferwachtte totale beleaning fan it model foar in parametergebiet. As we no parameter weardes fine sadat de ferwachtte totale beleaning maximaal ε fan sa'n grins ôfleit, hawwe we in oplossing foar it benei-optimale syntheseprobleem fûn. Oars matte we it parameter gebiet splitse, en foar elk subdomein de grinzen berekkenje. In neidiel fan dizze oanpak is dat we in eksponensjeel oantal subdomeinen krije kinne.

Yn dit proefskrift yntrodusearje en definiearje we de monotoniciteit fan parametryske Markov modellen. Yn in protte systemen sjogge we dat as we de kâns fan in parameter ferheegje, dat dan ek de ferwachtte totale beleaning tanimt. Dizze eigenskip brûke we om it benei-optimale syntheseprobleem oan te pakken en dermei parameter lifting te ferbetterjen. Alderearst definiearje we monotoniciteit foar pMCs en litte we sjen hoe we dizze berekkenje kinne mei help fan in preoarder fan de tastannen. Dernei wreidzje we dizze oanpak út nei pMDPs. Mei help fan in implementaasje yn de modelchecker

Storm, litte we troch eksperiminten sjen dat ús oanpak automatysk monotoniciteit foar pMCs en pMDPs berekkenje kin sûnder earst de rasjonele funksje te berekkenjen. Fjirder kombinearje we parameter lifting en monotoniciteit foar pMCs, en litte we mei help fan eksperiminten sjen dat we mei dizze kombinearre oanpak sneller it benei-optimale syntheseprobleem oplosse kinne.

In oar probleem dat we behannelje yn dit proefskrift is it ferfolberhydsprobleem foar parametryske Markov modellen, nammentlik it finen fan parameter weardes, sadat we it doel berikke mei ten minste in beskate wierskynlikens (of ferwachtte totale beleaning). We brûke in gradiïntmetoade út de literatuer en litte sjen hoe we mei help fan dizze metoades it ferfolberhydsprobleem oplosse kinne. We analysearje ferskate klassike en adaptive metoaden, en ferlykje dizze mei de bekinde metoades particle swarm optimization (PSO), quadratically-constrained quadratic program (QCQP) en sequential convex programming (SCP). De resultearjende metoade, sa as implementearre yn Storm, skaalt nei oansjenlik gruttere pMC's as foarhinne en prestearret empirysk better dan PSO en QCQP, faaks mei minstens ien oarder fan grutte. Yn fergeliking mei SCP prestearret de resultearjende metoade itselde.

Oant slot rjochtsje we ús op in parametryske útwreiding fan probabilistyske timede automaten (PTA). Dizze PTA kombinearje de tiidsaspekten fan timede automaten mei MDPs. Se wurde brûkt om kommunikaasjeprotokollen en netwurksystemen te analysearjen. Krekt as by oare Markov modellen, binne de kânsen yn de PTA jûn, dêrom wreidzje we se út nei parametryske PTA, sadat de kânsen sich yn in beskaat interfal befine kinne en ôfhinklik fan elkoar wêze kinne. De semantyk fan in pPTA is in ûneinige pMDP, wy analysearje in oantal besteande technyken foar PTA en brûke dizze om in einige pMDP út in pPTA te ekstrahearjen. De implementaasje hjirfan is beskikber yn de Modest Toolset. Om bygelyks monotoniciteit yn in pPTA fêst te stellen, kinne we no de ûnderlizzende pMDP analysearje mei de technyk dy't yn dit proefskrift yntrodusearre wurdt.

Acknowledgment

During the principles of model checking course at the University of Twente, Joost-Pieter told us about the possibility to write a Master's thesis at his group in Aachen. As I liked the principles of model checking course and wanted to go abroad (although not too far away), I decided to write Joost-Pieter for the possibility of writing my Master's thesis in Aachen, not knowing I would stay there for another five years. After some discussions with Sebastian and Joost-Pieter, I decided to focus on monotonicity in Markov chains. Once I finished writing my Master's thesis, we felt there was still a lot of work to do on this topic, and Joost-Pieter suggested applying for a Ph.D. Position at the research training group UnRAVeL. I am deeply thankful to Joost-Pieter for offering me the possibility to join both his group and UnRAVeL. I'm grateful for allowing me to find my way in academia, guiding me when necessary but also giving me the freedom to make my own mistakes. Defending your dissertation is of course not possible with the help of others. Therefore, I would like to express my heartfelt gratitude to Tatjana Petrov for graciously serving as my external examiner, and to Sandra Geisler and Jan Bender for readily stepping into the roles of committee members despite the short notice.

Of course, conducting research is not something you do on your own. First of all, I would like to thank Sebastian for guiding me through my Master's thesis into Ph.D. life and helping me to develop my own ideas on Markov models. Next, I would like to thank my co-authors Arnd, Bram, Joost-Pieter, Joshua, Linus, Sebastian, and Svenja for the research we did together. Also, I would like to thank Matthias for his feedback on my dissertation, and all the help with the Zusammenfassung and the LaTeX issues. Furthermore, I would like to thank everyone at UnRAVeL for the nice discussions on topics that weren't directly related to Markov models or model checking. These discussions really helped me get a broader view of computer science. Finally, I would thank everyone in the group for the fruitful discussions, for being supportive, and for dealing with my hyperactivity. Even when I was jumping around the hallway, doing silly stuff, no one ever complained about it. Special thanks to Jana, for being the dream team with table soccer and allowing me to always be in the Dutch team. Also, I would like to thank Tim, Sara, Hanna, and Mia for inviting me to fun activities like visiting the Tierpark or playing with Lego.

Part of academia is also guiding students when write their theses. I would like to thank Linus, Bram, Joel, and Anike for writing their thesis with me. Some of these even resulted in publications. Also, student assistants are really important. One of the first students I supervised for a course was Svenja, for pro-seminar. We immediately made a good match, and Svenja became my student assistant. First refactoring my code, and later working out ideas I had. Also, Linus became my student assistant, after finishing his Bachelor's thesis. I would like to thank him for the rewarding discussion we had on how to tackle problems within pMCs.

Everyone who knows me knows that sport plays an important role in my life. When moving to Aachen, I immediately joined the Limburgse Schaatsvrienden, they welcomed me to their association and allowed me to, next to training, also develop myself as a trainer for both children and adults. Teaching children how to inline skate and ice skate really provides me with a lot of joy, but also training adults gives me a lot of energy. I would like to express my heartfelt gratitude to Bart and Yvonne, as well as Denise, Amina, and Youssef, for their generous hospitality in providing me a place to stay whenever I sought company, desired engaging activities, or simply found myself reluctant to commute home following my training sessions. Also, I would like to thank the entire Skeuvel and Shorttrackteam GO community for their warm reception whenever I returned to Enschede. Special thanks to Esther and Marijn for the delightful pancake-filled moments we shared, and to the Diepvriesburgers for the wonderful training camps.

Finally, I wish to extend my gratitude to my family for their unwavering support throughout this expedition. Their patience, motivation, and faith in my abilities helped me to complete my dissertation. I am particularly thankful to my partner, Sybe, for his steadfast encouragement during this journey. His constant understanding of late nights, countless revisions, and moments of academic frustration truly show the strength of our bond. I express my heartfelt appreciation for your constant presence and unwavering support. Thank you for always standing by me.

Contents

1 Introduction

1.1 Motivation

Markov models | Probabilistic behavior occurs in many systems. E.g., in self-stabilizing protocols for distributed systems [IJ90] and exponential backoff mechanisms in wireless networks (e.g., [BMM12]), this behavior occurs. Also, for the robustness in autonomous systems with noisy sensors [TBF05], obfuscation, and (fuzz) test coverage [FS18], randomization plays an important role. We can model these systems as Markov models and check them against a specification given in (a probabilistic extension of) LTL or CTL. We refer to this as probabilistic model checking [Kat16; BAFK18]. The main focus of model checking is computing the reachability probability or expected total reward upon reaching a set of target states. E.g., with a high probability an autonomous system should not crash, or for self-stabilizing protocol, the expected number of steps to reach a stable configuration should be within a given range. Efficient probabilistic model checkers include PRISM [KNP11] and Storm [HJKQ+22].

We discuss two types of Markov models, namely, Markov chains and Markov decision processes. Markov chains (MCs) are the common model to describe close-loop systems with probabilistic behavior. The transition probabilities in an MC only depend on the current state. Markov decision processes (MDPs) extend MCs with non-determinism in the transitions. The non-determinism can be resolved using schedulers. E.g., if we want to maximize the probability of a given property, we try to find a maximizing scheduler that resolves the non-determinism. Such a scheduler can be memoryless (i.e., fixed for a given state) or be based on the visited states.

Partial observability | In AI and planning [RN20], many systems exhibit partial observability, e.g., in robots that have a partial perception of their environment. Partially observable MDPs (POMDPs) [KLC98; RN20] are the key model for such systems. Instead of knowing the actual state of a system, one only knows the state's observation. Multiple states may have the same observation; thus, the current state can, in general, not be uniquely identified. For POMDPs, the controllers (aka. schedulers or policies) try to resolve the non-determinism based on the (a part of) the observation history (cf. schedulers for MDPs). Optimal decision-making in POMDPs is even more complex than for MDPs, i.e., finding a controller to maximize reachability probabilitiesis

undecidable for POMDPs [MHC03], whereas this is a polynomial-time problem for MDPs.

Parametric Markov models | A major practical obstacle in the Markov models introduced is that the transition probabilities need to be given precisely. Therefore, uncertainty for these probabilities cannot be modeled in these basic models. One way to deal with this is to specify transition probabilities by intervals, as in interval Markov chains [JL91; CSH08]. A disadvantage of this is that parameter dependencies cannot be modeled. However, parameter dependencies occur naturally. Consider, e.g., a communication channel over which messages are sent. If we send two messages over the same channel, we want the probability of successful transmission for both messages to be the same. In interval Markov chains, this dependency is lost. Therefore, we consider parametric Markov models (pMMs, which include pMCs and pMDPs), where arithmetic expressions over real-valued parameters give the transition probabilities [Daw04].

A pMC (pMDP) represents an uncountably large family of MCs (MDPs): each parameter instantiation from the parameter space induces an MC (MDP). Both reachability and expected total reward properties can be lifted to pMMs. They are satisfied for a subset of the family of the Markov model or, equivalently, for a subset of the parameter instantiations. Key problems for pMMs are:

- **Feasibility:** does there exist a parameter instantiation such that a given property holds?
- **Verification:** do all parameter instantiations satisfy a given property?
- **Synthesis:** which parameter instantiations satisfy a given property?
- **Optimal synthesis:** for which parameter instantiations is the probability of satisfying a given property maximal?

In addition to optimal synthesis, there also exists ε-optimal synthesis, where the probability for the parameter instantiation may be at most ε away from the real optimum.

Applications of pMCs include model repair [BGKR+11; CHHK+13; PÁJT+15; CK18; GST19], strategy synthesis in AI models such as POMDPs [JJWQ+18], and optimizing randomized distributed algorithms [AVBK+17].

Synthesizing parameter instantiations | The main challenge in analyzing pMMs is the complexity. E.g., the feasibility problem of a pMC for a reachability probability exceeding 1/2 is ETR-complete[1] (thus NP-hard) [JKPW21]. Also, symbolic computations, as opposed to numeric computations using floating-point numbers, significantly

[1] ETR = Existential Theory of the Reals. ETR lies between PSPACE and NP. ETR-complete decision problems are as hard as finding the roots of a multivariate polynomials.

impact the analysis time [QDJJ+16]. However, in recent years, efficient techniques have been developed to analyze pMMs. E.g., for the synthesis problem, it suffices to guess a correct parameter instantiation. Verifying this guessed instantiation against the property can be done by using off-the-shelf model-checking algorithm for the given Markov model. Most recent progress is based on advanced techniques that make informed guesses: This ranges from using sampling techniques [HHZ11a], guided sampling such as particle swarm optimization [CHHK+13], greedy search [PÁJT+15], or solving different variants of a linear/convex optimization problem around a sample [CJJK+17; CJJK+18; CJJK+22]. Sampling has been accelerated by reusing previous model checking results [QDJJ+16] and just-in-time compilation of the parameter function [GHS18]. Another technique that may help finding a solution to the synthesis problem is gradient descent [SSY18; DDB20; 4]. The method is rooted in two key observations:

- gradient-based search methods, i.e., variants of gradient search, scale to high-dimensional search spaces, and
- in pMCs, the gradient at a parameter evaluation can be efficiently evaluated.

Therefore, we introduce gradient descent for pMCs.

Optimal synthesis and monotonicity | The methods introduced above are not suitable for the optimal synthesis problem. One way to find optimal parameter instantiation is by using rational functions obtained through some form of state elimination [Daw04; HHZ11b; GHS18]. These functions are, however, exponential in the number of parameters [HBK17] and become practically infeasible for more than two parameters. Another technique for optimal synthesis is parameter lifting [QDJJ+16; CDPK+17; CJJK19], combined with region splitting. Parameter lifting is an abstraction technique that prevents an exponential blow-up in the number of parameters as it avoids computing these rational functions. However, the region splitting might still yield an exponential blow-up. Therefore, parameter lifting is only feasible for a handful of parameters.

Monotonicity can help simplify the analysis of pMMs; e.g., if we want to check whether all points within a rectangular set of parameter values satisfy a property, it is reduced to checking a line fragment if one of the parameters is monotonic. Thus, any problem for an n+k-dimensional hyper-rectangle reduces to checking an n-dimensional hyper-rectangle when the considered pMM is monotonic in k parameters.

Probabilistic timed automata | Probabilistic timed automata (PTA) [GJ95; KNSS02] combine the non-deterministic probabilistic behavior of MDPs with the continuous real-time behavior of timed automata (TA) [AD94]. They are used to study randomized algorithms in an environment where non-deterministic actions finish after (bounded) delays. E.g., they have been used to verify the performance and reliability of communication protocols and network systems, e.g., [DKN04; Fru06; DJKG16; KK20]. Over

the past two decades, several approaches have been developed to verify PTA. Most of these approaches take an abstraction of the continuous-time behavior by turning the PTA into an equivalent MDP, on which the property can be verified using standard MDP model checking [BAFK18; BHK19].

As for the Markov models discussed before, we require knowledge of the exact probabilities of all random events when building a PTA model. Requiring this knowledge is unproblematic for a randomized algorithm (e.g., the binary exponential backoff procedure of the CSMA/CA wireless networks [KNSW07]). However, uncertainty about the operating environment often means that we do not know, e.g., the precise probability of message loss once we decide to send. Therefore, we introduce parametric PTA (pPTA), where the transition probabilities can be parametric as for pMMs.

1.2 Contribution

Below we elaborate on the main contributions of the core chapters of this dissertation. Chapter 3 defines both global and local monotonicity in a parametric Markov chain (pMC). It shows that obtaining monotonicity in pMCs is as hard as pMC verification. Finally, it provides a sufficient condition for monotonicity and an algorithmic approach to check this condition automatically. Chapter 4 extends the work of Chapter 3 to parametric Markov decision processes (pMDPs). Chapter 5 shows how parameter lifting can benefit from monotonicity checking. Chapter 6 shows how we can use the optimization method gradient descent to solve the feasibility problem for pMCs. Chapter 7 defines parametric probabilistic timed automata and shows how we can define the semantics as finite-state pMDPs. Properties for a parametric probabilistic timed automaton, such as monotonicity, can now be checked on the pMDP.

1.3 Personal Contributions

Chapters 3 and 5 through 7 partly represent earlier work co-authored by me. Chapter 4 builds upon Chapter 3 and represents new (unpublished) work. Following the regulations of the RWTH Aachen University, I shortly state my contributions to each of the relevant papers on which this dissertation is based upon. I want to preface this list by stating my personal opinion that research is very much a team effort that yields the best results through the exchange and discussion of ideas. Ideas are constantly shaped, discussed, adapted, implemented, evaluated, improved— and sometimes also discarded. Therefore, it is hard to quantify the contributions of each author with respect to research results. I summarize only my own recollection of what I (think that I) contributed. The presentation below is necessarily subjective, but my recollections have been backed by at least multiple co-authors per paper.

- [1] *'Are Parametric Markov Chains Monotonic?'* Spel, Junges and Katoen in ATVA 2019. This paper is a different approach to the monotonicity problem as also discussed in [Spe18]. The authors together came up with the idea to use a pre-order on the states to obtain monotonicity. I implemented the ideas and refined them. The writing process was a group effort.

- [2] *'Tweaking the Odds in Probabilistic Timed Automata'* Hartmanns, Katoen, Kohlen and Spel in QEST 2021. This paper is the result of the masters' thesis by Kohlen. I supervised Kohlen during his master thesis at RWTH Aachen University. After Kohlen finished his thesis and moved to the University of Twente to work as a PhD student with Hartmanns, we wrote this paper together.

- [3] *'Finding Provably Optimal Markov Chains'* Spel, Junges and Katoen in TACAS 2021. This paper builds on the results from [1]. As was already suggested there, we use monotonicity to improve parameter lifting [QDJJ+16]. I implemented this combined approach. The writing process was a group effort.

- [4] *'Gradient-Descent for Randomized Controllers Under Partial Observability'* Heck, Spel, Junges, Moerman and Katoen in VMCAI 2022. This paper is an extension to the work done by Heck as a bachelors thesis at the RWTH Aachen University. The idea of using gradient descent was brought up by Junges, and the basics were extended by Moerman. I supervised Heck during his thesis, and brainstormed with him about the implementation. The writing process was a group effort.

- [5] *'POMDP Controllers with Optimal Budget'* Spel, Stein and Katoen in QEST 2022. This paper considers monotonicity for the expected total reward. Stein made the first implementation for this, by creating the structure of the code. I refined and improved the code and supervised the student assistant Stein throughout her work. The paper was written by me, with valuable feedback from my co-authors.

1.4 Synopsis and Reading Guide

Each chapter in this dissertation starts with a *summary*, the *origin*, and the *background* of the chapter. The summary outlines the structure of the chapter. The origin paragraph describes which (co-authored) papers relate to the chapter. The background refers to parts of previous chapters which are background knowledge for the chapter. After this, a section on the related work follows, providing an overview of the literature relevant to the chapter. Finally, for all chapters with an empirical evaluation (Chapters 3 through 6), we formulate questions that we answer during this empirical evaluation. Below, we outline the structure of the dissertation.

Chapter 2: Preliminaries (pages 9-28) | This chapter contains background material that many readers are already familiar with. The main purpose of this chapter is to introduce the common model types and problem statements used in the other chapters.

Chapter 3: Monotonicity for Markov Chains (pages 29-67) | This chapter introduces global and local monotonicity of the solution function of parametric Markov chains. As obtaining monotonicity is as hard as pMC verification, we define a *sufficient condition* for monotonicity. This condition is based on a pre-order on the states. We explain how we can algorithmically build such an order and show how we obtain monotonicity from this. Finally, we support the work by empirically evaluating the approach.

Chapter 4: Monotonicity for Markov Decision Processes (pages 69-97) | This chapter focuses on monotonicity for parametric Markov decision processes. In parametric Markov decision processes, we focus on the optimal solution functions. Therefore, we extend the approach of Chapter 3 by determining an optimal scheduler. We show how we can algorithmically compute such a scheduler on-the-fly. We conclude this chapter by empirically evaluating the approach.

Chapter 5: Tackling the ε-Optimal Synthesis Problem (pages 99-124) | This chapter considers the ε-optimal synthesis problem, where the goal is to find an (almost) optimal instantiation for the solution function of the parametric Markov chain. First, we introduce parameter lifting [QDJJ+16], which is a technique to tackle the ε-optimal synthesis problem. Then we show how parameter lifting can benefit from monotonicity checking and study the use of this integrated approach by an empirical evaluation.

Chapter 6: Gradient Descent (pages 125-157) | This chapter describes how we can use the optimization method gradient descent to solve the feasibility problem, where the goal is to find an instantiation for the parametric Markov chain such that a given property is satisfied. We study the use of several gradient descent methods and compare them to each other and to the baseline methods particle swarm optimization [CHHK+13], quadratically-constrained quadratic programming [CJJK+18], and sequential convex programming [CJJK+22]..

Chapter 7: Parametric Probabilistic Timed Automata (pages 159-176) | This chapter defines parametric probabilistic timed automata and shows how we can define its semantics as a finite-state parametric Markov decision process. We can then check properties for the parametric probabilistic timed automata by analyzing the underlying parametric Markov decision process. E.g., to check for monotonicity, we could use the theory of Chapter 4.

Chapter 8: Conclusion and Future Work (pages 177-179) | This chapter summarizes the results of all previous chapters and discusses open questions.

Appendix A: Algorithms (pages 181-188) | This chapter contains the full algorithms to automatically infer monotonicity on pMCs (Algorithm 10) and pMDPs (Algorithm 11).

Appendix B: Experiments (pages 189-198) | This chapter provides the set-up for the experiments of Chapters 3 through 5. It also describes the benchmarks used for the experiments in more detail.

Appendix C: Obtaining pMCs from POMDPs (pages 199-202) | This chapter contains background material on transforming partially observable Markov decision processes into parametric Markov chains. It is solely based on [Jun20]. We use the theory of this chapter to obtain some of the benchmarks as described in Appendix B.2.1.

Back matter (pages 203-235) | Finally, the back matter consists of the Declaration of authorship, a bibliography (which consists of the references on pages 205-222 and my own publications on page 223), a list of symbols (pages 225-229), a list of acronyms (pages 225-229), and an index (pages 233-235).

2 Preliminaries

Summary | First, we introduce in Section 2.1 basic notation on multivariate polynomials and probability distributions. Secondly, we introduce parametric Markov models (Markov chains and Markov decision processes) in Section 2.2. Then, Section 2.3 introduces reachability and expected total reward properties. Finally, we introduce in Section 2.4 several problem statements for parametric Markov Models.

Origin | This chapter does not contain any novel material. We reused parts of the preliminaries sections of [1; 3; 4; 5]. Some helpful textbooks are, e.g., [Fel01; BK08].

Background | A list of acronyms and a list of symbols are given on pages 231-232 and pages 225-229, respectively.

2.1 Basic Notation

Multivariate polynomials | Let $\vec{a} \in \mathbb{R}^n$ denote the vector $(a_1, \ldots, a_n)$, and $\vec{e}_i$ denote the vector with $e_j = 1$ if $i = 0$ and $e_j = 0$ otherwise. The set of *multivariate polynomials* over ordered *variables* $\vec{x} = (x_1, \ldots, x_n)$ is denoted $\mathbb{Q}[\vec{x}]$.

polynomial
variable

Definition 2.1 An *instantiation* for a finite set V of real-valued variables is a function $u : V \to \mathbb{R}$.

instantiation

We typically denote u as a vector $\vec{u} \in \mathbb{R}^n$ with $u_i := \vec{u}(x_i)$. A polynomial f can be interpreted as a *function* $f : \mathbb{R}^n \to \mathbb{R}$, where $f(\vec{u})$ is obtained by substitution i.e., $f[\vec{x} \leftarrow \vec{u}]$, where each occurrence of x_i in f is replaced by $\vec{u}(x_i)$.

Example 2.1 Let $V = \{p, q\}$ denote a set variables, function $f := 3p^2 - p \cdot q + q$ is a polynomial with 2 variables. For instantiation $\vec{u}$ with $\vec{u}(p) = 1$ and $\vec{u}(q) = 2$, we have $f(\vec{u}) = 3 \cdot 1^2 - 1 \cdot 2 + 2 = 3$.

·monotonic increasing

Definition 2.2 A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is *monotonic increasing in x_i on set $R \subset \mathbb{R}^n$*, denoted $f(\vec{u})\uparrow^R_{x_i}$, if for all $\vec{a} \in R$ and $b \in \mathbb{R}_{\geq 0}$:

$$f(\vec{a}) \leq f(\vec{a} + b \cdot \vec{e}_i)\ .$$

·monotonic decreasing

A function $f(\vec{u})$ is *monotonic decreasing in x_i on R*, denoted $f(\vec{u})\downarrow^R_{x_i}$, if $(-f(\vec{u}))\uparrow^R_{x_i}$. A function $f(\vec{u})$ is *monotonic increasing on R*, denoted $f(\vec{u})\uparrow^R$, if $f(\vec{u})\uparrow^R_{x_i}$ for all $x_i \in \vec{x}$. $f(\vec{u})\downarrow^R$ is defined analogously.

Given a continuously differentiable function $f(\vec{u})$ on open set $R \subset \mathbb{R}^n$. If $\frac{\partial}{\partial x_i} f(\vec{u}) \geq 0$ for all $\vec{u} \in R$, then $f(\vec{u})\uparrow^R_{x_i}$. Moreover, any $f(\vec{u}) \in \mathbb{Q}[\vec{x}]$ is continuously differentiable on $\mathbb{R}^n$.

Example 2.2 Consider $f(p,q) = 3p^2 - p \cdot q + q$ from the previous example. We have $\frac{\partial}{\partial p} f(p,q) = 6p - q$ and $\frac{\partial}{\partial q} f(p,q) = -p + 1$. First of all, let $R_1(p) = [0,1]$ and $R_1(q) = 0$. For R_1, we have $\frac{\partial}{\partial p} f(p,q) \geq 0$ and $\frac{\partial}{\partial q} f(p,q) \geq 0$ for any $(p,q) \in R_1$. We thus obtain $f(p,q)\uparrow^{R_1}_p$ and $f(p,q)\uparrow^{R_1}_q$. From this, we obtain $f(p,q)\uparrow^{R_1}$. Next, let $R_2(p) = [1,2]$ and $R_2(q) = [0,1]$. We now have $\frac{\partial}{\partial p} f(p,q) \geq 0$ and $\frac{\partial}{\partial q} f(p,q) \leq 0$ for any $(p,q) \in R_2$. So we obtain $f(p,q)\uparrow^{R_2}_p$ and $f(p,q)\downarrow^{R_2}_q$. As $f(p,q)$ is monotonic increasing for p and monotonic decreasing for q, we cannot obtain general monotonicity for $f(p,q)$ on region R_2. Finally, let $R_3 = [0,1]^2$. For R_3, we have $\frac{\partial}{\partial p} f(0,1) = -1$ and $\frac{\partial}{\partial p} f(1,0) = 6$, thus we cannot obtain monotonicity of $f(p,q)$ in p. For variable q, we can obtain monotonicity, as $\frac{\partial}{\partial p} f(p,q) \geq 0$ for all $(p,q) \in R_3$, we have $f(p,q)\uparrow^{R_3}_q$.

Remark 2.1 In the remainder of this dissertation, we often refer to variables as *parameters*, and we refer to R as the *parameter space* or *region* (see also page 14).

probability distribution

·parametric

Probability Distribution | A *probability distribution* over a finite or countably infinite set X is a function $\mu\colon X \to [0,1] \subseteq \mathbb{R}$ with $\sum_{x \in X} \mu(x) = 1$. $\mathrm{Distr}(X)$ denotes the set of all distributions on X. Let $\mathrm{Distr}_{\mathbb{Q}(V)}(X)$ denote the set of *parametric probability distributions* over X induced by the polynomials in $\mathbb{Q}(V)$, i.e., the set of functions $\mu\colon X \to \mathbb{Q}(V)$ such that $\sum_{x \in X} \mu(x)(\vec{u}) = 1$ and $\mu(x)(\vec{u}) \geq 0$ for all $\vec{u}$ within the parameter space R.

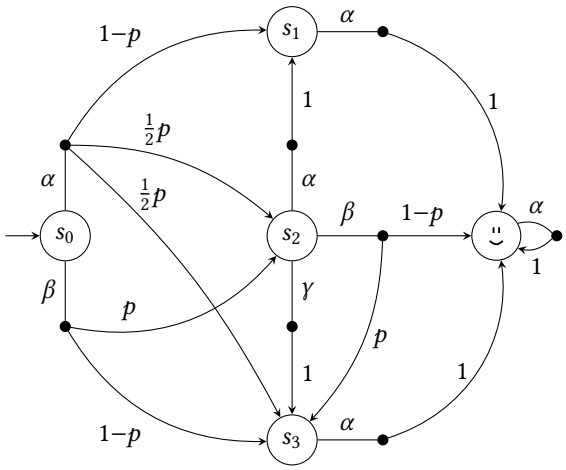


Figure 2.1: An example parametric Markov decision process $\mathcal{M}_1$

Example 2.3 Let $V = \{p\}$ and $X = \{a, b\}$. A parametric distribution is given by $\mu(a) = p$ and $\mu(b) = 1 - p$ with $p \in [0, 1]$.

2.2 Markov Models

In this section, we consider Markov models. A Markov model is a stochastic model consisting of states and (probabilistic) transitions. The probability distribution describing the next state, only depends on the current state, and not on previously seen states. We discuss the parametric version [Daw04] of Markov chains (MC) [Chu67; KS76] and Markov decision processes (MDP) [Bel57; How60; Put14].

parametric MDP

Definition 2.3 A *parametric Markov Decision process (pMDP)* is a tuple $\mathcal{M} = (S, s_I, T, Act, V, \mathbf{P})$ with a finite set S of *states*, an *initial state* $s_I \in S$, a finite set $T \subseteq S$ of *target states*, a finite set Act of *actions*, a finite set V of real-valued *variables* (*parameters*), and a *partial transition function* $\mathbf{P} \colon S \times Act \times S \nrightarrow \mathbb{Q}[V]$.

Let $Act(s) = \{\alpha \in Act \mid \mathbf{P}(s, \alpha, \cdot) \neq \bot\}$ denote the set of available actions at state s. We assume that at each state at least one action is available, i.e., $|Act(s)| \geq 1$ for all $s \in S$.

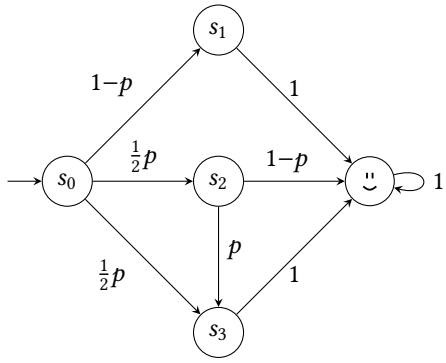


Figure 2.2: An example parametric Markov chain $\mathcal{D}_1$

Example 2.4 Figure 2.1 depicts pMDP $\mathcal{M}_1$ with set of actions $\{\alpha, \beta, \gamma\}$ and single parameter p. It consists of the states $\{s_0, s_1, s_2, s_3, \lightning\}$ and the transition functions $\{1/2p, p, 1-p, 1\}$. The initial state is state s_0, and the set of target states is $\{\lightning\}$.

A parametric Markov chain is a pMDP without non-determinism, i.e., there is exactly one action available for each state.

parametric MC

Definition 2.4 A *parametric Markov Chain (pMC)* is a pMDP such that $|Act(s)| = 1$ for all states $s \in S$. We denote a pMC as tuple $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ with S, s_I, T, V as in Definition 2.3 and *transition function* $\mathbf{P}\colon S \times S \to \mathbb{Q}[V]$.

Example 2.5 Figure 2.2 depicts pMC $\mathcal{D}_1$ with single parameter p. It consists of the states $\{s_0, s_1, s_2, s_3, \lightning\}$ and the transition functions $\{1/2p, p, 1-p, 1\}$. The initial state is state s_0, and the set of target states is $\{\lightning\}$.

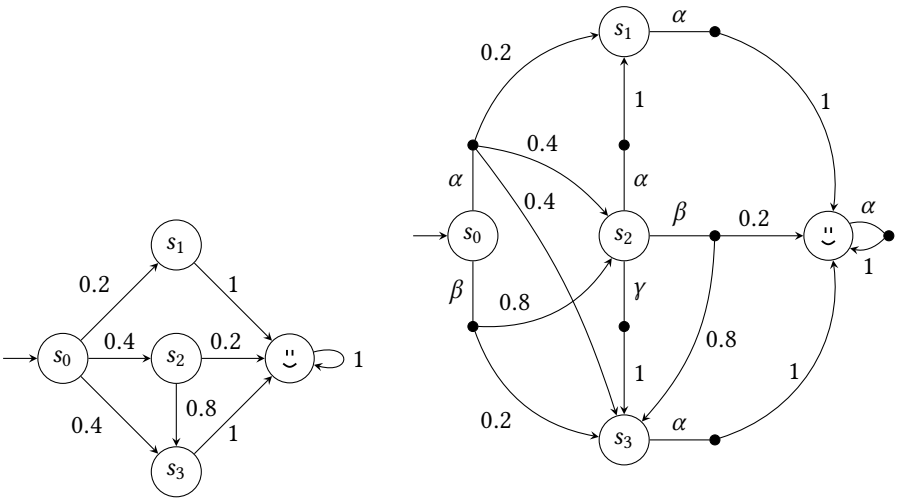
Assumption 2.1 We assume all target states to be absorbing, i.e., for $s \in T$ we have $\mathbf{P}(s, \alpha, s) = 1$ for all $\alpha \in Act(s)$.

2.2.1 Applying parameter instantiations

Remark 2.2 In this subsection, we introduce terminology for pMDPs; this terminology also transfers to pMCs as they are defined as a special case of pMDPs.

instantiation

Applying an *instantiation* $\vec{u}$ to a pMDP $\mathcal{M}$ yields $\mathcal{M}[\vec{u}]$ by replacing each $f \in \mathbb{Q}[V]$



(a) Markov chain $\mathcal{D}_2$ (b) Markov decision process $\mathcal{M}_2$

Figure 2.3: Example of parameter-free Markov models

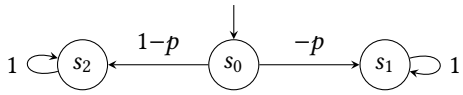


Figure 2.4: An example of a *not* well-defined parametric Markov chain

in $\mathcal{M}$ by $f(\vec{u})$. An instantiation $\vec{u}$ is *well-defined* (for $\mathcal{M}$) if for $\mathcal{M}[\vec{u}]$ all transitions probabilities are non-negative, i.e., for all states $s, s' \in S$ and actions $\alpha \in Act(s)$, we have $\mathbf{P}(s, \alpha, s')(\vec{u}) \geq 0$. Applying a well-defined instantiation $\vec{u}$ to a pMDP $\mathcal{M}$ yields MDP $\mathcal{M}[\vec{u}]$. A well-defined instantiation $\vec{u}$ is *graph-preserving* (for $\mathcal{M}$) if the topology is preserved, i.e., $\mathbf{P}(s, \alpha, s') \neq 0$ implies $\mathbf{P}(s, \alpha, s')(\vec{u}) \neq 0$ for all states $s, s' \in S$ and actions $\alpha \in Act(s)$.

well-defined

graph-preserving

Example 2.6 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1 and pMC $\mathcal{D}_1$ of Figure 2.2. Instantiation $\vec{u}(p) = 0.8$ is graph-preserving. Applying this instantiation to $\mathcal{M}_1$ ($\mathcal{D}_1$) yields MDP $\mathcal{M}_1[\vec{u}] = \mathcal{M}_2$ (MC $\mathcal{D}_1[\vec{u}] = \mathcal{D}_2$) of Figure 2.3b (Figure 2.3a). Instantiation $\vec{u}(p) = 0$ is well-defined but not graph-preserving, as, e.g., $\mathbf{P}(s_0, \alpha, s_2)(\vec{u}) = 0$, although $\mathbf{P}(s_0, \alpha, s_2) \neq 0$.$\vec{u}(p) = -0.1$ is a not well-defined

instantiation, as, e.g., $\mathbf{P}(s_0, \alpha, s_2)(\vec{u}) = -0.05$. Figure 2.4 depicts a pMC that is not well-defined as $\mathbf{P}(s, \cdot) \notin \text{Distr}(S)$.

Remark 2.3 For conciseness, we directly instantiate our parameters, e.g., we write $p = 0.1$ instead of $\vec{u}(p) = 0.1$.

region

A set of instantiations is called a *region* (see also Remark 2.1). A region R is *well-defined* (*graph-preserving*) if $\vec{u}$ is well-defined (graph-preserving) for all $\vec{u} \in R$.

Assumption 2.2 We only consider graph-preserving regions.

Example 2.7 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. First of all, let $R = [0.1, 0.9]$, this region is graph-preserving, as all non-zero transitions have a strictly positive probability. Secondly, consider $R' = [0, 0.9]$, this region is not graph-preserving, as for $p = 0$, e.g., for action β, the probability for the transition from s_2 to s_3 becomes 0. Finally, consider $R'' = [-0.1, 0.9]$, this region is not well-defined as, e.g., for action β, the transition from s_2 to s_3 has value -0.1 for $p = -0.1$.

2.2.2 Resolving non-determinism

To resolve non-determinism in pMDPs we introduce schedulers. Note that these schedulers are *memoryless*, i.e., they do not depend on states seen before. As schedulers might depend on the parameter instantiation, we discuss both independent and dependent schedulers. First, we focus on independent schedulers, which fix an action for each state $s \in S$, only based on the current state.

·independent scheduler

Definition 2.5 An *independent scheduler* for a pMDP $\mathcal{M} = (S, s_I, T, Act, V, \mathbf{P})$ is a function $\sigma \colon S \to Act$, where $\sigma(s) \in Act(s)$ for all $s \in S$.

Secondly, we consider dependent schedulers, the action chosen by the scheduler does not only depend on the state, but also the instantiation $\vec{u}$.

·dependent scheduler

Definition 2.6 A *dependent scheduler* for a pMDP $\mathcal{M} = (S, s_I, T, Act, V, \mathbf{P})$ is a function $\sigma \colon R \times S \to Act$, where $\sigma(\vec{u}, s) \in Act(s)$ for all $s \in S, \vec{u} \in R$.

Example 2.8 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. First of all, we observe that for states $s \in \{s_1, s_3, \lightning\}$, $|Act(s)| = 1$, so for both independent

and dependent schedulers we have $\sigma(s) = \alpha$. An example of an independent scheduler is

$$\sigma(s) = \begin{cases} \alpha & \text{if } s \in \{s_0, s_1, s_3, \text{␣}\} \\ \beta & \text{otherwise.} \end{cases}$$

The dependent scheduler

$$\sigma(p, s) = \begin{cases} \alpha & \text{if } s \in \{s_0, s_1, s_3, \text{␣}\} \\ \alpha & \text{if } s = s_2 \text{ and } p \geq \frac{1}{3} \\ \beta & \text{otherwise,} \end{cases}$$

always chooses action α for states $s_0, s_1, s_3, \text{␣}$. However, for state s_2, the action chosen by $\sigma(p, s)$ depends on the value of p.

We often refer to independent schedulers as schedulers. For conciseness, we often drop s from the scheduler notation. We let $Sched(\mathcal{M})$ ($Sched(\mathcal{M}, \vec{u})$) denote the set of all independent (dependent) schedulers for pMDP $\mathcal{M}$. Furthermore, we let $Dep(\sigma(\vec{u}))$ denote the set of independent schedulers on which $\sigma(\vec{u})$ depends, i.e., $Dep(\sigma(\vec{u})) = \{\sigma \in Sched(\mathcal{M}) \mid \exists \vec{u} \in R.\, \sigma(\vec{u}) = \sigma\}$.

Example 2.9 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. $Sched(\mathcal{M}_1)$ consists of the schedulers depicted in the table below.

	$\sigma(s_0)$	$\sigma(s_1)$	$\sigma(s_2)$	$\sigma(s_3)$	$\sigma(\text{␣})$
σ_1	α	α	α	α	α
σ_2	α	α	β	α	α
σ_3	α	α	γ	α	α
σ_4	β	α	α	α	α
σ_5	β	α	β	α	α
σ_6	β	α	γ	α	α

For dependent scheduler $\sigma(p)$ of Example 2.8, we have $Dep(\sigma(p)) = \{\sigma_1, \sigma_2\}$.

Definition 2.7 Let $\mathcal{M} = (S, s_I, T, Act, V, \mathbf{P})$ be a pMDP and σ be an independent scheduler on $\mathcal{M}$. The *induced* pMC $\mathcal{M}^\sigma = (S, s_I, T, V, \mathbf{P}^\sigma)$ where $\mathbf{P}^\sigma(s, s') = \mathbf{P}(s, \sigma(s), s')$ for $s, s' \in S$. ·induced

For a dependent scheduler $\sigma(\vec{u})$ and parameter instantiation $\vec{u} \in R$ we have induced parameter-free MC $\mathcal{M}[\vec{u}]^{\sigma(\vec{u})}$.

Example 2.10 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. The induced pMC $\mathcal{M}_1^{\sigma_2}$, obtained by applying scheduler σ_2 of Example 2.9 to $\mathcal{M}_1$, is depicted as pMC $\mathcal{D}_1$ in Figure 2.2.

2.2.3 Notation

In this subsection, we introduce notation for pMDPs, this notation also transfer to pMCs as they are a special case of pMDPs. First of all, we introduce notation for the set of successors of a state. Let $\text{succ}(s,\alpha) = \{s' \in S \mid \mathbf{P}(s,\alpha,s') \neq 0\}$ be the set of direct successors of s when taking action α. Let $\text{succ}(s) = \{s' \mid \exists \alpha \in Act(s).\, s' \in \text{succ}(s,\alpha)\}$ be the set of all possible direct successors of s.

Example 2.11 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. For state s_2, we have $\text{succ}(s_2,\alpha) = \{s_1\}$, $\text{succ}(s_2,\beta) = \{s_3, \circlearrowleft\}$, and $\text{succ}(s_2,\gamma) = \{s_3\}$. From this, we obtain $\text{succ}(s_2) = \{s_1, s_3, \circlearrowleft\}$ as set of all possible direct successors of s_2. Now, reconsider our running example pMC $\mathcal{D}_1$ of Figure 2.1. As s_1 is not a successor of s_2 in $\mathcal{D}_1$, we have $\text{succ}(s_2) = \{s_3, \circlearrowleft\}$.

For independent scheduler σ, we define $\text{succ}(s,\sigma) = \text{succ}(s,\sigma(s))$ as the set of all direct successors of s w.r.t. scheduler σ. Furthermore, we define the set of all possible direct successors of s w.r.t. dependent scheduler $\sigma(\vec{u})$ as $\text{succ}(s,\sigma(\vec{u})) = \{\text{ s' } \mid \exists \sigma \in Dep(\sigma(\vec{u})).\, \text{s'} \in \text{succ(s,}\sigma\text{(s))}\}$.

Example 2.12 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. For independent scheduler σ_1 and state s_2, we have $\text{succ}(s_2,\sigma_1) = \{s_1\}$. For dependent scheduler $\sigma(p)$ of Example 2.8 and state s_2, we have $\text{succ}(s_2,\sigma(p)) = \{s_1, s_3, \circlearrowleft\}$.

We let $\text{occur}(s,\alpha) = \{x \in V \mid \exists s' \in S.\, x \in \mathbf{P}(s,\alpha,s')\}$ denote the set of parameters occurring at state s for action α. Furthermore, we denote the set of all parameters occurring at state s by $\text{occur}(s) = \{x \mid \exists \alpha \in Act(s).\, x \in occur(s,\alpha)\}$. A state s is called *parametric*, if $\text{occur}(s) \neq \emptyset$; we write $\text{occur}(s) = x$ if $\{x\} = \text{occur}(s)$. A parameter p is *dependent* if it occurs at multiple states, i.e., there exist $s_1, s_2 \in S$ such that $s_1 \neq s_2$, $p \in \text{occur}(s_1)$, and $p \in \text{occur}(s_2)$.

·parametric
·dependent

Example 2.13 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. For s_2, we have $\text{occur}(s_2,\alpha) = \emptyset$, $\text{occur}(s_2,\beta) = \{p\}$, and $\text{occur}(s_2,\gamma) = \emptyset$. From this, we obtain $\text{occur}(s_2) = \{p\}$, so s_2 is parametric. Now, consider our running example pMC $\mathcal{D}_1$ of Figure 2.2 and induced pMC $\mathcal{M}^{\sigma_4}$ as depicted as $\mathcal{D}_3$ in Figure 2.5. State s_2 is parametric in $\mathcal{D}_1$, however, in $\mathcal{D}_3$ state s_2 is not parametric, as $\text{occur}(s_2) = \emptyset$ for $\mathcal{D}_3$

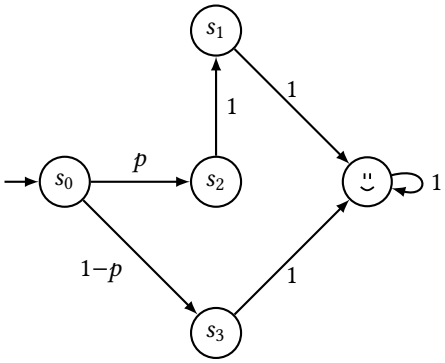


Figure 2.5: An example simple pMC $\mathcal{D}_3$

A pMDP $\mathcal{M} = (S, s_I, T, Act, V, \mathbf{P})$ is *simple* if for all $s, s' \in S$ and $\alpha \in Act(s)$ we have $\mathbf{P}(s, \alpha, s') \in \{p, 1-p\} \cup [0, 1]$, with $p \in V$. ·simple

Example 2.14 The pMDP $\mathcal{M}_1$ and pMC $\mathcal{D}_1$ of Figures 2.1 and 2.2 are not simple, as for state s_0 non-simple transitions occur. The pMC $\mathcal{D}_3$ as depicted in Figure 2.5 is simple.

2.3 Probabilities and Rewards

In this section, we first introduce the path notation for Markov models (Section 2.3.1). Then, in Section 2.3.2, we introduce reachability probabilities and expected total reward in parameter-free MCs. Next, we extend this to parameter-free MDPs in Section 2.3.3. Finally, we generalize this to pMCs and pMDPs in Section 2.3.4.

2.3.1 Paths

An *infinite path* of a pMDP $\mathcal{M}$ is a non-empty infinite sequence $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$ of states $s_i \in S$ such that $\mathbf{P}(s_i, \alpha_i, s_{i+1}) > 0$ for $i \geq 0$. Let $Paths^{\mathcal{M}}_{inf}(s)$ denote the set of all infinite paths in $\mathcal{M}$ starting from s. Let $pref(\pi)$ denote the set of finite prefixes of π. A *finite path* of a pMDP $\mathcal{M}$ is a finite, non-empty prefix of an infinite path in $\mathcal{M}$. Let $Paths^{\mathcal{M}}_{fin}(s)$ denote the set of all finite paths in $\mathcal{M}$ starting from s. We define the *length of a finite path* $|\hat{\pi}| = n$ for $\hat{\pi} = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{n-1}} s_n$, and let π_i be the i^{th} state of a path. Note that a path of length n consists of $n+1$ states. We let $Paths^{\mathcal{M}}_{fin} = \bigcup_{s \in S} Paths^{\mathcal{M}}_{fin}(s)$. ·infinite ·finite ·length

·prefix ·initial

We define $Paths_{fin}^{\mathcal{M},n}$ and $Paths_{inf}^{\mathcal{M}}$, analogously. The set of *prefixes* of infinite path $\pi \in Paths_{inf}^{\mathcal{M}}$ is denoted by $pref(\pi) = \{\hat{\pi} \in Paths_{fin}^{\mathcal{M}} \mid \pi_i = \hat{\pi}_i \text{ for all } 0 \leq i \leq n\}$. For pMDP $\mathcal{M} = (S, s_I, T, Act, V, \mathbf{P})$, a path π is *initial* if $\pi_0 = s_I$. We drop superscript $\mathcal{M}$ from the path notation whenever this is clear from the context.

Example 2.15 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. An initial infinite path for $\mathcal{M}_1$ is, e.g.,

$$\pi = s_0 \xrightarrow{\beta} s_2 \xrightarrow{\gamma} s_3 \xrightarrow{\alpha} \breve{\cup} \left(\xrightarrow{\alpha} \breve{\cup} \right)^{\omega} .$$

An initial finite path of length 4 for $\mathcal{M}_1$ is, e.g.,

$$\hat{\pi} = s_0 \xrightarrow{\beta} s_2 \xrightarrow{\gamma} s_3 \xrightarrow{\alpha} \breve{\cup} \xrightarrow{\alpha} \breve{\cup} .$$

Note that $\hat{\pi}$ is a prefix of π, i.e., $\hat{\pi} \in pref(\pi)$.

For pMDP $\mathcal{M}$ and state s, we have for infinite path $\pi \in Paths_{inf}^{\mathcal{M}}(s)$: $\pi \models \Diamond T$ if there exists an i such that $\pi_i \in T$. For finite path $\hat{\pi} \in Paths_{fin}^{\mathcal{M}}$ of length n, we have $\hat{\pi} \models \Diamond T$ if there exists an $i \leq n$ such that $\pi_i \in T$. Furthermore, we let set

$$Paths_{inf}^{\mathcal{M},\Diamond T}(s) = \{\pi \in Paths_{inf}^{\mathcal{M}}(s) \mid \pi \models \Diamond T\}$$

denote the set of all infinite paths in $\mathcal{M}$ that eventually reach a state in T.

Example 2.16 We continue from Example 2.15. We observe that $\pi \models \Diamond \breve{\cup}$ and $\hat{\pi} \models \Diamond \breve{\cup}$. As all infinite paths eventually reach $\breve{\cup}$, we have $Paths_{inf}^{\mathcal{M}_1}(s) = Paths_{inf}^{\mathcal{M}_1,\Diamond \breve{\cup}}(s)$.

The definitions for infinite and finite paths in pMCs can be obtained from the definitions above by omitting the actions.

Example 2.17 Reconsider our running example pMC $\mathcal{D}_1$ of Figure 2.2. For initial state s_0, we have

$$Paths_{inf}^{\mathcal{D}_1}(s_0) = \{s_0 s_1 \breve{\cup}^{\omega}, s_0 s_2 \breve{\cup}^{\omega}, s_0 s_2 s_3 \breve{\cup}^{\omega}, s_0 s_3 \breve{\cup}^{\omega}\} .$$

As all paths eventually reach $\breve{\cup}$, we have $Paths_{inf}^{\mathcal{D}_1}(s_0) = Paths_{inf}^{\mathcal{D}_1,\Diamond \breve{\cup}}(s_0)$. Further-

more, we have

$$Paths_{fin}^{\mathcal{D}_1}(s_0) = \{s_0, s_0s_1, s_0s_2, s_0s_3, s_0s_1\circlearrowleft, s_0s_2\circlearrowleft, s_0s_2s_3, s_0s_3\circlearrowleft, \ldots\} .$$

2.3.2 Parameter-Free Markov chains

In this section, we discuss reachability probabilities and expected total reward in parameter-free MCs.

Reachability probabilities | To be able to reason about the probability of eventually reaching a state in target set T, we first need to define the probability of a path. For a parameter-free MC, let

$$\Pr(\hat{\pi}) = \sum_{i=0}^{n-1} \mathbf{P}(s_i, s_{i+1})$$

denote the *probability of a finite path* $\hat{\pi} = s_0s_1 \ldots s_n$. ·probability

Example 2.18 Reconsider our running example MC $\mathcal{D}_2$ of Figure 2.3a. For finite path $\hat{\pi} = s_0s_2s_3$ we have $\Pr(\hat{\pi}) = 0.4 \cdot 0.8 = 0.32$.

We lift this to the *probability of an infinite path* via a cylinder set construction [BK08, Ch. 10].

Definition 2.8 The *cylinder set* of finite path $\hat{\pi} \in Paths_{fin}^{\mathcal{D}}$ is defined as: cylinder set

$$Cyl(\hat{\pi}) = \{\pi \in Paths_{inf}^{\mathcal{D}} \mid \hat{\pi} \in pref(\pi)\} .$$

The cylinder set for $\hat{\pi}$ thus contains all infinite paths that have $\hat{\pi}$ as a prefix. By applying classical probability theory [AD00; Fel01] to the smallest σ-algebra containing all cylinder set $\mathrm{Cyl}(\hat{\pi})$, we obtain for the set of infinite paths starting with $\hat{\pi}$:

$$\Pr(Cyl(\hat{\pi})) = \Pr(\pi) .$$

Example 2.19 We continue from Example 2.18 with $\hat{\pi} = s_0s_2s_3$. For this finite path, we have $Cyl(\hat{\pi}) = \{s_0s_2s_3\circlearrowleft^{\omega}\}$, and $\Pr(Cyl(\hat{\pi})) = 0.32$.

For a parameter-free MC $\mathcal{D}$, we let $\Pr_{\mathcal{D}}^{s}(\Diamond T) \in [0,1] \subseteq \mathbb{R}$ denote the probability that from state s the target T is reached.

reachability probability

Definition 2.9 The *reachability probability* for reaching T from $s \in S$ for an MC $\mathcal{D}$ is defined as follows:

$$\Pr^s_{\mathcal{D}}(\Diamond T) = \Pr(Paths^{\mathcal{D},\Diamond T}_{inf}) \ .$$

We drop superscript $\mathcal{D}$, whenever it is clear from the context. Furthermore, we let $\lightning$ and $\lightning$ denote all states $s \in S$ with $\Pr^s(\Diamond T) = 1$ and $\Pr^s(\Diamond T) = 0$, respectively.

Assumption 2.3 By a standard preprocessing [BK08], we may assume a single $\lightning$ and $\lightning$ state. Furthermore, we assume $\lightning$ and $\lightning$ to be absorbing.

Example 2.20 Reconsider our running example MC $\mathcal{D}_2$ of Figure 2.3a. For any state $s \in S$, we have $\Pr^s(\Diamond\lightning) = 1$. Now, let $T = \{s_3\}$, we have $\Pr^{s_0}(\Diamond s_3) = 0.4 \cdot 0.8 + 0.4 = 0.72$.

Expected total rewards | Next, we consider the expected total reward for eventually reaching a target state. Note that in literature, it is also referred to as expected total cost. Let *state reward* function $r\colon S \to \mathbb{R}_{\geq 0}$ associate a non-negative reward to each state in an MC. The *cumulative reward* for the finite path $\hat{\pi} = s_0 s_1 \dots s_n$ is defined by:

state reward

cumulative reward

$$r(\hat{\pi}) = r(s_0) + r(s_1) + \dots + r(s_{n-1}) \ .$$

Rewards are thus earned when leaving a state. For infinite path $\pi = s_0 s_1 s_2 \dots$, the expected total reward to eventually reach T in $\mathcal{D}$ is the cumulative reward until reaching a state in T for the first time. Formally:

$$r(\pi, \Diamond T) = \begin{cases} r(s_0 s_1 \dots s_n) & \text{if } s_i \notin T \text{ for } 0 \leq i < n \text{ and } s_n \in T \\ \infty & \text{if } \pi \not\models \Diamond T. \end{cases}$$

For finite path $\hat{\pi} = s_0 s_1 \dots s_n$, the expected total reward until eventually reaching T in $\mathcal{D}$ is defined slightly different, namely:

$$r(\hat{\pi}, \Diamond T) = \begin{cases} r(s_0 s_1 \dots s_m) & \text{if } s_i \notin T \text{ for } 0 \leq i < m \leq n \text{ and } s_m \in T \\ r(\hat{\pi}) & \text{if } \hat{\pi} \not\models \Diamond T. \end{cases}$$

Assumption 2.4 We often define the reward function for all states $s \in S \setminus \{T\}$. We assume $r(t) = 0$ for $t \in T$, as we only consider the cumulative reward until reaching T, thus we never need $r(t)$.

Example 2.21 Reconsider MC $\mathcal{D}_2$ of Figure 2.3a. Let the state reward function $r(s_i) = i$. For infinite path $\pi = s_0 s_2 s_3 (\circlearrowleft)^\omega$, we have $r(\pi, \Diamond \circlearrowleft) = 2 + 3 = 5$. For finite path $\hat{\pi}_1 = s_0 s_2 s_3$, we have $r(\hat{\pi}, \Diamond \circlearrowleft) = 2$. For finite path $\hat{\pi}_2 = s_0 s_2 s_3 \circlearrowleft\circlearrowleft$, we have $r(\hat{\pi}, \Diamond \circlearrowleft) = 5$.

expected total reward

Definition 2.10 The *expected total reward* until reaching T from $s \in S$ for an MC $\mathcal{D}$ is defined as follows:

$$\mathrm{ER}^s_{\mathcal{D}}(\Diamond T) = \sum_{\pi \in Paths_{inf}(s)} \Pr(\pi) \cdot r(\pi, \Diamond T).$$

We drop superscript $\mathcal{D}$ whenever this is clear from the context. As state rewards are non-negative, the expected total reward is in $\mathbb{R}_{\geq 0}$ or equal to ∞.

Example 2.22 We continue with Example 2.21. The expected total reward from initial state s_0 to target state $\circlearrowleft$ is given by

$$\mathrm{ER}^{s_0}(\Diamond \circlearrowleft) = \underbrace{0.2 \cdot 1}_{s_0 s_1 \circlearrowleft^\omega} + \underbrace{0.4 \cdot 0.2 \cdot 2}_{s_0 s_2 \circlearrowleft^\omega} + \underbrace{0.4 \cdot 0.8 \cdot 5}_{s_0 s_2 s_3 \circlearrowleft^\omega} + \underbrace{0.4 \cdot 3}_{s_0 s_3 \circlearrowleft^\omega} = 3.16\,.$$

We observe that we can write $\mathrm{ER}^s(\Diamond T)$ as the recursive equation:

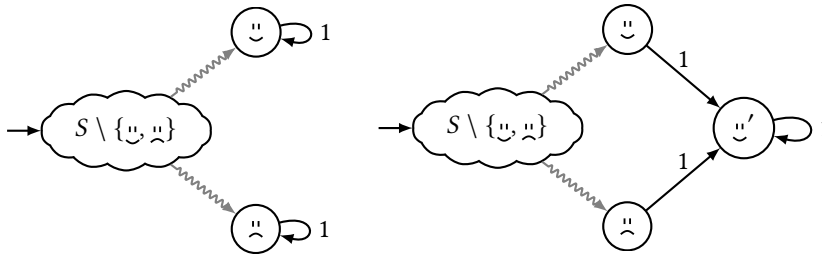
$$\mathrm{ER}^s(\Diamond T) = \begin{cases} 0 & \text{if } s \in T \\ r(s) + \sum_{s' \in \mathrm{succ}(s)} \mathbf{P}(s, s')(\vec{u}) \cdot \mathrm{ER}^{s'}(\Diamond T) & \text{otherwise}\,. \end{cases}$$

Example 2.23 We continue with Example 2.22. The expected total reward from initial state s_0 to target state $\circlearrowleft$ is given by

$$\mathrm{ER}^{s_0}(\Diamond \circlearrowleft) = 0 + 0.2 \cdot \underbrace{1}_{\mathrm{ER}^{s_1}(\Diamond \circlearrowleft)} + 0.4 \cdot \underbrace{(2 + 0.8 \cdot 3)}_{\mathrm{ER}^{s_2}(\Diamond \circlearrowleft)} + 0.4 \cdot \underbrace{3}_{\mathrm{ER}^{s_3}(\Diamond \circlearrowleft)}\,.$$

Assumption 2.5 W.l.o.g., we may assume there exists exactly one s such that $\mathrm{ER}^s(\Diamond \circlearrowleft) = 0$.

Assumption 2.6 For expected total reward properties, we restrict ourselves to MCs where all infinite paths eventually reach T. Otherwise, $\mathrm{ER}^{s_I}(\Diamond T) = \infty$.



(a) $\mathcal{D}$ (b) $\mathcal{D}'$

Figure 2.6: Construction to relate probabilities to rewards

From reachability probabilities to expected total rewards | We can use the expected total reward to obtain the reachability probability for reaching a state in T from state s in MC $\mathcal{D}$. Recall from Assumption 2.3 that we assume a single absorbing state ☺ representing T, and a single absorbing state ☹, representing the states for which $\Pr^s(\Diamond T) = 0$. We create an MC $\mathcal{D}'$ from $\mathcal{D}$ such that $\Pr^s_{\mathcal{D}}(\Diamond ☺) = \mathrm{ER}^s_{\mathcal{D}'}(\Diamond ☺')$ as follows:

1. Add a sink state ☺′ to S (yielding S') with for any $s \in S$, $\mathbf{P}(s, ☺') = 1$ if $\mathbf{P}(s, s) = 1$ and $\mathbf{P}(s, ☺') = 0$ otherwise. Furthermore, let $\mathbf{P}(☺, ☺) = 0$, all other probabilities from $\mathcal{D}$ remain unchanged.

2. Equip all states in $\mathcal{D}'$ with reward zero, except for ☺, i.e., $r(s) = 0$ if $s \in S' \setminus \{☺\}$ and $r(☺) = 1$.

The quantity $\mathrm{ER}^{s_0}_{\mathcal{D}'}(\Diamond ☺')$ now equals the reachability probability of eventually reaching ☺ in $\mathcal{D}$. Figure 2.6a shows the original MC $\mathcal{D}$, Figure 2.6b shows $\mathcal{D}'$.

> **Remark 2.4** Note that by eliminating state ☹ from $\mathcal{D}'$, we have exactly one state s such that $\mathrm{ER}^s(\Diamond ☺) = 0$, thus Assumption 2.5 holds.

2.3.3 Parameter-Free Markov decision processes

For MDPs, the reachability probabilities and expected total rewards depend on the actions chosen at the states. Instead of reasoning about the reachability probability or the expected total reward, we now reason about the minimal and maximal reachability probabilities and expected total rewards. The *minimal reachability probability* in MDP $\mathcal{M}$ of eventually reaching a state in T is defined as follows:

·minimal probability

$$\Pr^s_{min,\mathcal{M}}(\Diamond T) = \min\{\Pr^s_{\mathcal{M}^\sigma}(\Diamond T) \mid \sigma \in \mathit{Sched}(\mathcal{M})\}$$

In a similar way, we define the *minimal expected total reward* in $\mathcal{M}$ as follows:

·minimal expected total reward

$$\mathrm{ER}^s_{min,\mathcal{M}}(\Diamond T) = \min\{\mathrm{ER}^s_{\mathcal{M}^\sigma}(\Diamond T) \mid \sigma \in \mathit{Sched}(\mathcal{M})\}$$

We define the *maximal reachability probability* ($\Pr^s_{max,\mathcal{M}}(\Diamond T)$) and *maximal expected total reward* ($\mathrm{ER}^s_{max,\mathcal{M}}(\Diamond T)$), analogously. For conciseness, we typically omit $\mathcal{M}$ if $\mathcal{M}$ is clear from the context.

·maximal probability
·maximal expected total reward

Example 2.24 Reconsider our running example MDP $\mathcal{M}_2$ of Figure 2.3b and assume we are interested in the optimal expected total reward for reaching ⟓. Observe that for $s \in \{s_1, s_3, ⟓\}$, there is only one available action. In the following, we let $\sigma_i(s) = \alpha$ for $s \in \{s_1, s_3, ⟓\}$, we now get the following functions for expected total rewards for $\mathcal{M}_2^{\sigma_i}$:

	$\sigma(s_0)$	$\sigma(s_2)$	$\mathrm{ER}^{s_0}(\Diamond ⟓)$
σ_1	α	α	2.6
σ_2	α	β	3.16
σ_3	α	γ	3.4
σ_4	β	α	3
σ_5	β	β	4.12
σ_6	β	γ	4.6

From this table, we obtain $\mathrm{ER}^{s_0}_{min}(\Diamond T) = 2.6$ and $\mathrm{ER}^{s_0}_{max}(\Diamond T) = 4.6$.

We let $\sigma^{min,\mathtt{pr}}$ denote a *minimizing scheduler* such that $\Pr^s_{min}(\Diamond T) = \Pr^s_{\mathcal{M}^{\sigma^{min,\mathtt{pr}}}}(\Diamond T)$ and we define $\sigma^{min,\mathtt{rew}}$, $\sigma^{max,\mathtt{pr}}$, and $\sigma^{max,\mathtt{rew}}$ in a similar manner. We drop superscripts `pr` and `rew` if it is clear from the context.

·minimizing/ maximizing scheduler

Remark 2.5 Note that the minimizing and maximizing schedulers are not necessarily unique. However, if there are two schedulers σ_1 and σ_2 which are both minimizing, the obtained reachability probability or expected total reward is the same for both schedulers.

Example 2.25 We continue from Example 2.24. We observe that $\sigma^{min}(s) = \alpha$ for all $s \in S$ and $\sigma^{max}(s) = \alpha$ for $s \in \{s_1, s_3, ⟓\}$, $\sigma^{max}(s_0) = \beta$, and $\sigma^{max}(s_2) = \gamma$.

2.3.4 Lifting probabilities and rewards to parameters

Parametric Markov chains | For pMC $\mathcal{D}$, the reachability probability $\Pr^s_{\mathcal{D}}(\Diamond T)$ and expected total reward $\mathrm{ER}^s_{\mathcal{D}}(\Diamond T)$ are not constants, but rather functions

solution function

$\Pr^{s\to T}_{\mathcal{D}}\colon V \to [0,1]$ and $\mathsf{ER}^{s\to T}_{\mathcal{D}}\colon V \to \mathbb{R}$, such that $\Pr^{s\to T}_{\mathcal{D}}(\vec{u}) = \Pr^{s}_{\mathcal{D}[\vec{u}]}(\Diamond T)$ and $\mathsf{ER}^{s\to T}_{\mathcal{D}}(\vec{u}) = \mathsf{ER}^{s}_{\mathcal{D}[\vec{u}]}(\Diamond T)$. We call $\Pr^{s\to T}_{\mathcal{D}}(\vec{u})$ and $\mathsf{ER}^{s\to T}_{\mathcal{D}}(\vec{u})$ the *solution functions*, and for conciseness, we typically omit $\mathcal{D}$. On a graph-preserving region, the function $\Pr^{s\to T}(\vec{u})$ and $\mathsf{ER}^{s\to T}(\vec{u})$ are always continuously differentiable on R [QDJJ+16] and admit a closed-form as a rational function over V [Daw04; HHZ09]. As for $\mathsf{ER}^{s}(\Diamond T)$, we can write $\mathsf{ER}^{s\to T}(\vec{u})$ as a recursive equation, namely:

$$\mathsf{ER}^{s\to T}(\vec{u}) = \begin{cases} 0 & \text{if } s \in T \\ r(s) + \sum\limits_{s' \in \mathsf{succ}(s)} \mathbf{P}(s,s')(\vec{u}) \cdot \mathsf{ER}^{s'\to T}(\vec{u}) & \text{otherwise .} \end{cases}$$

Example 2.26 Reconsider our running example pMC $\mathcal{D}_1$ of Figure 2.2 with state reward function $r(s_i) = i$. The expected total reward from initial state s_0 to target state $\lightning$ is given by:

$$\begin{aligned}
\mathsf{ER}^{s_0\to \lightning}(p) &= 0 + (1-p)\cdot \mathsf{ER}^{s_1\to \lightning}(p) + \frac{1}{2}p \cdot \mathsf{ER}^{s_2\to \lightning}(p) + \frac{1}{2}p \cdot \mathsf{ER}^{s_3\to \lightning}(p) \\
\mathsf{ER}^{s_1\to \lightning}(p) &= 1 + 1 \cdot \mathsf{ER}^{\lightning\to \lightning}(p) \\
\mathsf{ER}^{s_2\to \lightning}(p) &= 2 + (1-p)\cdot \mathsf{ER}^{\lightning\to \lightning}(p) + p \cdot \mathsf{ER}^{s_3\to \lightning}(p) \\
\mathsf{ER}^{s_3\to \lightning}(p) &= 3 + 1 \cdot \mathsf{ER}^{\lightning\to \lightning}(p) \\
\mathsf{ER}^{\lightning\to \lightning}(p) &= 0
\end{aligned}$$

From this, we obtain $\mathsf{ER}^{s_0\to \lightning}(p) = \frac{3}{2}p^2 + \frac{3}{2}p + 1$.

Parametric Markov decision processes | As for pMCs, the reachability probability $\Pr^{s}_{min,\mathcal{M}}(\Diamond T)$ and expected total reward $\mathsf{ER}^{s}_{min,\mathcal{M}}(\Diamond T)$ (and there duals) are not constant, but solution functions $\Pr^{s\to T}_{min,\mathcal{M}}\colon V \to [0,1]$ and $\mathsf{ER}^{s\to T}_{min,\mathcal{M}}\colon V \to \mathbb{R}$, such that $\Pr^{s\to T}_{min,\mathcal{M}}(\vec{u}) = \Pr^{s}_{min,\mathcal{M}[\vec{u}]}(\Diamond T)$ and $\mathsf{ER}^{s\to T}_{min,\mathcal{M}}(\vec{u}) = \mathsf{ER}^{s}_{min,\mathcal{M}[\vec{u}]}(\Diamond T)$. We define the duals ($\Pr^{s\to T}_{max,\mathcal{M}}(\vec{u})$ and $\mathsf{ER}^{s\to T}_{max,\mathcal{M}}(\vec{u})$) in a similar manner. We drop superscript $\mathcal{M}$ whenever this is clear from the context.

Remark 2.6 Note that as for parameter-free MDPs, the optimal scheduler is not necessarily unique. Furthermore, for parametric MDP, the optimal scheduler might be dependent (Definition 2.6), i.e., for a region R, the optimal scheduler might depend on $\vec{u} \in R$, see also Example 2.27.

If σ^{min} is independent, then $\Pr^{s\to T}_{min}(\vec{u})$ and $\mathsf{ER}^{s\to T}_{min}(\vec{u})$ are continuously differentiable

on R and admit a closed-form as a rational function over V. If σ^{min} is dependent, then $\Pr^{s\to T}_{min}(\vec{u})$ and $\mathsf{ER}^{s\to T}_{min}(\vec{u})$ are not necessarily continuously differentiable on R, however, they are continuous.

Remark 2.7 For a dependent scheduler $\sigma(\vec{u})$, we often drop $\vec{u}$ and write $\mathsf{ER}^{s\to T}_{\mathcal{M}^\sigma}(\vec{u})$ to denote the dependent expected total reward function induced by $\sigma(\vec{u})$.

Example 2.27 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1 with state reward function $r(s_i) = i$. In the following, we let $\sigma_i(s) = \alpha$ for $s \in \{s_1, s_3, \lightning\}$, we now get the following functions for expected total rewards for $\mathcal{M}_1^{\sigma_i}$:

	$\sigma(s_0)$	$\sigma(s_2)$	$\mathsf{ER}^{s_0\to\lightning}$
σ_1	α	α	$2p+1$
σ_2	α	β	$\frac{3}{2}p^2+\frac{3}{2}p+1$
σ_3	α	γ	$3p+1$
σ_4	β	α	3
σ_5	β	β	$3p^2-p+3$
σ_6	β	γ	$2p+3$

The maximizing scheduler (σ_6) is independent and the maximizing expected total reward function $\mathsf{ER}^{s_0\to\lightning}_{max}(p)$ is given by $p\cdot(2+3)+(1-p)\cdot 3 = 2p+3$. The minimizing expected total reward function, however, depends the value of p. For region $R_1 = [0.8, 0.9]$, the minimizing scheduler is independent and given by σ_1. However, for region $R_2 = [0.1, 0.9]$, we have

$$\sigma^{min}(p) = \begin{cases} \sigma_1 & \text{if } p \geq \frac{1}{3} \\ \sigma_2 & \text{otherwise .} \end{cases}$$

Thus for the expected total reward we get:

$$\mathsf{ER}^{s_0\to\lightning}_{min}(p) = \begin{cases} 2p+1 & \text{if } p \geq \frac{1}{3} \\ \frac{3}{2}p^2+\frac{3}{2}p+1 & \text{otherwise .} \end{cases}$$

2.4 Problem Statements

In this section, we introduce the various problems we consider throughout this dissertation. In Section 2.4.1, we introduce the decision and synthesis problems for pMCs. In Section 2.4.2, we elaborate on the decision problems for pMDPs.

Table 2.1: Complexity landscape for feasibility in pMCs adapted from [JKPW21]

	Fixed # parameters	Arbitrary # parameters	
		well-defined	graph-preserving
$\mathsf{REACH}^{</>}$	in P	NP-hard	$\mathsf{REACH}^{>}_{\text{wd}}$-compete
$\mathsf{REACH}^{\leq/\geq}$	in P	ETR-complete	

2.4.1 Problems for pMCs and their complexity

First, we introduce the feasibility problem and the verification problem, which are both decision problems. Next, we introduce the optimal and ε-optimal synthesis problem.

Decision problems | We let $\bowtie \in \{<, \leq, \geq, >\}$, and consider the following decision problems for pMCs:

- ***Feasibility:*** *Given a pMC $\mathcal{D}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does there exist a $\vec{u} \in R$ such that $\mathsf{ER}^{s \to T}_{\mathcal{D}}(\vec{u}) \bowtie \lambda$?*
- ***Verification:*** *Given a pMC $\mathcal{D}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does $\mathsf{ER}^{s \to T}_{\mathcal{D}}(\vec{u}) \bowtie \lambda$ hold for all $\vec{u} \in R$?*

For the feasibility problem, we check if there exists an instantiation in our region such that our property holds, whereas for the verification problem we check if the property holds for all instantiations in our region. Any verification problem can be written as the negation of a feasibility problem.

Example 2.28 Reconsider our running example pMC $\mathcal{D}_1$ of Figure 2.2. Let $\bowtie = >$ and $\lambda = 2.7$. First of all, consider region $R_1 = [0.8, 0.9]$. For this region the answer to both the feasibility problem and the verification problem is "yes". Now, consider region $R_2 = [0.1, 0.9]$. The answer to the feasibility problem remains "yes", as for, e.g., $p = 0.8$, $\mathsf{ER}^{s_0 \to \{\!\{\}\!\}}(p) > 2.7$ holds. The answer to the verification problem is "no", as for, e.g., $p = 0.1$, we have $\mathsf{ER}^{s_0 \to \{\!\{\}\!\}}(p) = 1.165 \not> 2.7$.

In [JKPW21], Junges et al. discuss the complexity of the feasibility problem for reachability probabilities (REACH) in simple pMCs. They distinguish between a fixed number of parameters, and an arbitrary number of parameters. Furthermore, they distinguish for an arbitrary number of parameters both well-defined (wd) and graph-preserving regions. The results for this are depicted in Table 2.1.

Synthesis problems | Next, we consider (almost) optimal synthesis problems. The goal of these synthesis problems is not to answer the question with "yes" or "no", but

to find an (almost) optimal instantiation $\vec{u}$.

- ***Optimal synthesis:*** Find the instantiation $\vec{u}^*$ such that

$$\vec{u}^* = \arg\max_{\vec{u}\in R} \mathsf{ER}_{\mathcal{D}}^{s_I\to T}(\vec{u})$$

- ***ε-Optimal synthesis:*** Given tolerance $\varepsilon \geq 0$, find an instantiation $\vec{u}^*$ such that

$$\max_{\vec{u}\in R} \mathsf{ER}_{\mathcal{D}}^{s_I\to T}(\vec{u}) \cdot (1-\varepsilon) \leq \mathsf{ER}_{\mathcal{D}}^{s_I\to T}(\vec{u}^*) \leq \max_{\vec{u}\in R} \mathsf{ER}_{\mathcal{D}}^{s_I\to T}(\vec{u}) .$$

Note that in the problems above, we assume we want to maximize the expected total reward for $\vec{u}$, analogously, we can define the problems when minimizing.

Example 2.29 We continue from Example 2.28. Let $\varepsilon = 0.1$ and region $R_1 = [0.8, 0.9]$. The answer to the optimal synthesis problem is: $p = 0.9$. For the ε-optimal synthesis problem, any $p > 0.8124$ is a solution.

The optimal synthesis problem is ETR-hard [JKPW21], therefore the ε-optimal synthesis problem is ETR-hard as well.

2.4.2 Problems for pMDPs and their complexity

Now, we consider the decision problems for pMDPs. As for pMCs, we let $\bowtie \in \{<, \leq, \geq, >\}$. Furthermore, let $Q \in \{\exists, \forall\}$. We consider the following decision problems for pMDPs:

- **Feasibility:** *Given a pMDP $\mathcal{M}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does there exist a $\vec{u} \in R$ such that $Q\sigma \in Sched(\mathcal{M})$. $\mathsf{ER}_{\mathcal{M}^\sigma}^{s\to T}(\vec{u}) \bowtie \lambda$?*

- **Verification:** *Given a pMDP $\mathcal{M}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does $Q\sigma \in Sched(\mathcal{M})$ such that $\mathsf{ER}_{\mathcal{M}^\sigma}^{s\to T}(\vec{u}) \bowtie \lambda$ hold for all $\vec{u} \in R$?*

Example 2.30 Reconsider our running example pMDP $\mathcal{M}_1$ of Figure 2.1. As in Example 2.28, let $\bowtie = >$ and $\lambda = 2.7$. First of all, consider region $R_1 = [0.8, 0.9]$. For $Q = \exists$, we observe that, if we take, e.g., scheduler σ_4 from Example 2.27, then $\mathsf{ER}^{s_0\to \lightning}(p) = 3$ for any $p \in R_1$, so the answer to both the feasibility problem and the verification problem is "yes". For $Q = \forall$, the answer to the feasibility problem is "yes", as for all possible schedulers at $p = 0.9$ we have $\mathsf{ER}^{s_0\to \lightning}(p) > 2.7$. If we consider the verification problem for $Q = \forall$, we observe that for scheduler σ_1 from Example 2.27 and $p = 0.8$, we have $\mathsf{ER}^{s_0\to \lightning}(p) = 2.6 \not> 2.8$, so the answer is "no". Now, consider region $R_2 = [0.1, 0.9]$. For both $Q = \exists$ and $Q = \forall$, the answer

Table 2.2: Complexity landscape for feasibility in pMDPs adapted from [JKPW21]

	Fixed # parameters	Arbitrary # parameters	
		well-defined	graph-preserving
$\exists\mathsf{REACH}^{\leq/\geq}$	in NP	ETR-complete	
$\exists\mathsf{REACH}^{>}$	in NP	$\mathsf{REACH}^{>}_{\text{wd}}$-complete	
$\exists\mathsf{REACH}^{<}$	in NP	$\mathsf{REACH}^{<}_{\text{wd}}$-complete	$\mathsf{REACH}^{>}_{\text{wd}}$-hard
$\forall\mathsf{REACH}^{\bowtie}$	in NP	ETR-complete	

feasibility problem remains "yes" as for all schedulers picking $p = 0.9$ is a solution. The answer to the verification problems is for $Q = \exists$ "yes", as we can pick, e.g., scheduler σ_4. The answer to the verification problem for $Q = \forall$ is "no".

In [JKPW21], Junges et al. also discuss the complexity of the feasibility problem for reachability probabilities ($Q\mathsf{REACH}$) in simple pMDPs, with $Q \in \{\exists, \forall\}$. They distinguish between a fixed number of parameters, and an arbitrary number of parameters. Furthermore, they distinguish for an arbitrary number of parameters both well-defined (wd) and graph-preserving regions. The results for this are depicted in Table 2.2. Note that the pMDPs in this work are not necessarily simple.

3 Monotonicity for Markov Chains

Summary | In this section, we introduce both global and local monotonicity for pMCs. As analyzing monotonicity is in co-ETR, we consider a different approach to obtain monotonicity based on a pre-order on the states, the so-called reachability or reward order. Although determining such an order is as hard as pMC verification, we explain how we can algorithmically build a sufficient order to obtain monotonicity. Finally, we support our work by empirically evaluating the approach. We show that, even though our approach is not complete, we can find monotonicity in cases where the approach using the expected total reward solution has a time-out.

Origins | The presentation is mainly based on [1; 5]. The proofs in this chapter are more extensive, and more examples are provided. We also generalize the parts of [1] which did not make it into [5].

Background | We consider pMCs and the reachability and expected total reward properties as introduced in Sections 2.2 and 2.3. Furthermore, we consider monotonicity as defined in Section 2.1.

3.1 Related Work

Monotonicity in Markov chains | Daley [Dal68] uses a stochastic ordering $\leq_{st}$ on the rows in the transition matrix $\mathbf{P}$ to compare MCs. MCs can be compared if $\mathbf{P}$ is monotone w.r.t. $\leq_{st}$ for all probability vectors. Haddad and Pekergin [HP09] use these orderings for multi-valued model checking of interval MCs. This has, however, not been applied to pMCs. The main difference to our approach is that we order the states, whereas [Dal68] orders the rows of the transition matrix.

Hutschenreiter et al. [HBK17] show that the complexity of model checking (a monotone fragment of) PCTL on monotonic pMC is lower than on general pMCs. Pathak et al. [PÁJT+15] provided an efficient greedy approach to repair monotonic pMCs.

Monotonicity in partially observable Markov decision processes | Miehling et al. [MT20] investigate monotonicity in two-action partially observable Markov decision processes (POMDPs). They propose a stochastic order on the state space, which imposes additional conditions on the beliefs of incomparable states. They obtain

a set of sufficient conditions that ensure the existence of an optimal policy that is monotone on the belief space w.r.t. the proposed stochastic order. This work is similar to our work in the sense that we can transform POMDPs into pMCs to obtain optimal policies (see Appendix C), and we can check for monotonicity of these pMCs. If the pMC is monotonic, the optimal policy is monotonic.

Monotonicity in continuous-time Markov chains | Research on monotonicity has also been done for continuous-time MCs (CTMCs). Dai Pra et al. [DLM10] formalize stochastic monotonicity and realizable monotonicity in CTMCs. Gouberman et al. [GST19] use monotonicity for hitting probabilities in perturbed CTMCs.

State orderings | Bharadwaj et al. [BRPT17] and Roux and Pérez [RP18] define a never-worse relation (NWR) on MDP states. This NWR is similar in spirit to our orders: states are ordered according to their maximal reachability probabilities without considering the exact probabilities. The computation of the NWR is also based on the graph structure. However, its usage is different from our setting. The goal of the NWR is to reduce the size of the MDP prior to model checking. The coNP-completeness [RP18] indicates that checking this order is simpler than our reward order unless coETR and coNP coincide.

Local monotonicity | Parameter lifting [QDJJ+16] exploits local monotonicity by removing the parameter dependencies in the parametric Markov model. The resulting parametric Markov model without parameter dependencies is then replaced by an MDP (or stochastic game for pMDPs) that over-approximates the original model. Global monotonicity is, however, not used in parameter lifting. Furthermore, Brim et al. [BCDS13] use local monotonicity to analyze continuous-time MCs. For interval MCs [JL91; CSH08], all parameters are independent by definition. Thus all states are locally monotonic. However, whether the parameters are locally monotonic increasing or decreasing remains unclear.

3.2 Defining Monotonicity

We start by defining global monotonicity, i.e., monotonicity of the expected total reward solution function of a pMC.

global monotonicity

Definition 3.1 The pMC $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ is *globally monotonic increasing* in parameter $p \in V$ on graph-preserving region R, written $\mathcal{D}{\uparrow}_p^R$, if $\mathsf{ER}^{s_I \to T}(\vec{u}){\uparrow}_p^R$. *Globally monotonic decreasing*, written $\mathcal{D}{\downarrow}_p^R$, is defined analogously.

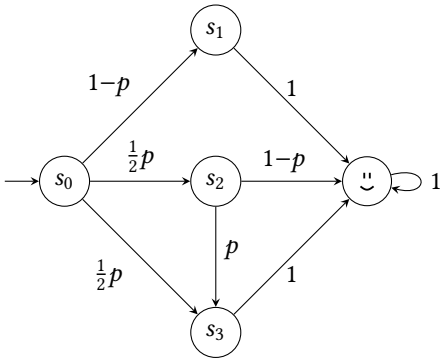


Figure 3.1: An example parametric Markov chain $\mathcal{D}_1$ with $r(s_i) = i$

Recall the recursive equation for the expected total reward solution function for $s \in S$: $\mathsf{ER}^{s \to T}(\vec{u}) = r(s) + \sum_{s' \in \mathsf{succ}(s)} \mathbf{P}(s,s')(\vec{u}) \cdot \mathsf{ER}^{s' \to T}(\vec{u})$, from this we obtain:

$$\mathcal{D}\uparrow_p^R \quad \text{iff} \quad \frac{\partial}{\partial p}\left(r(s_I) + \sum_{s' \in \mathsf{succ}(s_I)} \mathbf{P}(s_I, s') \cdot \mathsf{ER}^{s' \to T} \right)(\vec{u}) \;\geq\; 0 \quad \text{for all } \vec{u} \in R\,.$$

Remark 3.1 Note that the derivative of state reward $r(s_I)$ vanishes, as $r(s_I)$ is constant, thus $\frac{\partial}{\partial p} r(s_I) = 0$. We thus have

$$\mathcal{D}\uparrow_p^R \quad \text{iff} \quad \frac{\partial}{\partial p}\left(\sum_{s' \in \mathsf{succ}(s_I)} \mathbf{P}(s_I, s') \cdot \mathsf{ER}^{s' \to T} \right)(\vec{u}) \;\geq\; 0 \text{ for all } \vec{u} \in R\,.$$

Example 3.1 Reconsider our running example pMC $\mathcal{D}_1$, also depicted in Figure 3.1. Let the state rewards be given by: $r(s_i) = i$. Recall from Example 2.26 that $\mathsf{ER}^{s_0 \to \text{☺}}(p) = 1 + \frac{3}{2}p + \frac{3}{2}p^2$. From this, we obtain

$$\frac{\partial}{\partial p}\mathsf{ER}^{s_0 \to \text{☺}}(p) = \frac{3}{2} + \frac{6}{2}p\,.$$

For any well-defined and graph-preserving region R, we now obtain $\frac{\partial}{\partial p}\mathsf{ER}^{s_0 \to \text{☺}}(p) \geq 0$, thus $\mathcal{D}\uparrow_p^R$.

Global monotonicity takes the derivative over the entire expected total reward solution function. Local monotonicity is defined for a single state $s \in S$ and only takes into account the derivative of the transition probabilities to the direct successors of s. As for global monotonicity (see Remark 3.1), the reward $r(s)$ vanishes from the derivative as it is constant.

local monotonicity

Definition 3.2 $\mathsf{ER}^{s\to T}(\vec{u})$ is *locally monotonic increasing* in parameter p (at s) on region R, denoted $\mathsf{ER}^{s\to T}(\vec{u})\uparrow_p^{\ell,R}$, if:

$$\left(\sum_{s'\in\mathsf{succ}(s)}\left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T}\right)(\vec{u}) \;\geq\; 0 \quad \text{for all } \vec{u}\in R\,. \tag{3.1}$$

Locally monotonic decreasing, denoted $\mathsf{ER}^{s\to T}(\vec{u})\downarrow_p^R$, is defined analogously.

For conciseness, we typically omit superscript R for $\uparrow_p^R$ and $\uparrow_p^{\ell,R}$ if R is clear from the context. Furthermore, we drop subscript p whenever there is monotonicity in for all parameters.

Example 3.2 Reconsider our running example pMC $\mathcal{D}_1$. Observe that for any probability p we have:

$$\begin{aligned}
&\overbrace{\mathsf{ER}^{s_1\to \circ}(p) < \mathsf{ER}^{s_2\to \circ}(p) = 2+3\cdot p}^{=1}\\
&\mathsf{ER}^{s_1\to \circ}(p) < \mathsf{ER}^{s_3\to \circ}(p)\\
&\underbrace{\mathsf{ER}^{\circ\to \circ}(p)}_{=0} < \mathsf{ER}^{s_3\to \circ}(p)
\end{aligned}$$

From Definition 3.2 we obtain: $\mathsf{ER}^{s_0\to \circ}(p)\uparrow_p^\ell$ and $\mathsf{ER}^{s_2\to \circ}(p)\uparrow_p^\ell$.

If a parameter p is locally monotonic increasing (decreasing) at all states, the reward function $\mathsf{ER}^{s_i\to T}$ is globally monotonic increasing (decreasing) in p.

Theorem 3.1

$$\left(\forall s\in S.\, \mathsf{ER}^{s\to T}(\vec{u})\uparrow_p^\ell\right) \quad \text{implies} \quad \mathsf{ER}^{s_I\to T}(\vec{u})\uparrow_p\,.$$

Proof. In order to prove Theorem 3.1, we first lift the notion of local monotonicity (Definition 3.2) to local monotonicity for n steps (Definition 3.3). Secondly, we show that local monotonicity implies local monotonicity for n steps for each n. Moreover, we show that we have local monotonicity for n steps with n in the limit going to ∞ (Lemma 3.2). Next, we prove Lemma 3.3, stating that global monotonicity follows from Lemma 3.2. The proof of Theorem 3.1 now directly follows from the proofs of Lemmas 3.2 and 3.3.

Local monotonicity for n steps | Let $Paths_n(s)$ be the set of all finite paths with length n starting from state $s \in S$, i.e., $Paths_n(s) = \{\hat{\pi} \mid \hat{\pi} \in Paths_{fin}(s) \wedge |\hat{\pi}| = n\}$. We extend the definition of local monotonicity to local monotonicity for n steps. For $n = 1$, the definition below corresponds to Definition 3.2.

Definition 3.3 $\mathsf{ER}^{s \to T}(\vec{u})$ is *locally monotonic increasing for n steps* in parameter p (at s) on region R, denoted $\mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell,n,R}$, iff for all $\vec{u} \in R$: ·for n steps

$$\left(\sum_{\hat{\pi} \in Paths_n(s)} \left(\frac{\partial}{\partial p} \Pr(\hat{\pi}) \right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_n \to T} \right) \right) (\vec{u}) \;\geq\; 0 \,.$$

If R is clear from the context, we omit superscript R from $\uparrow_p^{\ell,n,R}$.

Local monotonicity implies local monotonicity for n steps | The following lemma claims that local monotonicity implies local monotonicity for n steps in the limit of n to ∞.

Lemma 3.2

$$\left(\forall s \in S. \, \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell} \right) \quad \text{implies} \quad \lim_{n \to \infty} \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell,n} \,.$$

As for any $s \in S$, $\Pr^{s \to T}(\vec{u}) = 1$, we know that all infinite paths eventually reach a state in T; from this, we observe that:

$$\left(\forall n. \, \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell,n} \right) \implies \lim_{n \to \infty} \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell,n} \,.$$

In order to prove Lemma 3.2, we use induction on n to show:

$$\left(\forall s \in S. \, \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell} \right) \implies \forall s \in S. \, \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell,n} \,. \tag{3.2}$$

Base case: $n = 1$. For $n = 1$, we only consider paths of length 1 (i.e., $|\hat{\pi}| = 1$). Recall from Section 2.3.1 that there are $n + 1$ states on a path of length n. So, considering paths of length 1 is equivalent to only considering the direct successors of a state. We thus obtain for $n = 1$ that Definition 3.3 is equivalent to Definition 3.2. So the base case trivially holds.

Induction step. Assume for $n \leq k$, the following holds:

$$\left(\forall s \in S.\, \mathsf{ER}^{s \to T}(\vec{u}){\uparrow_p^\ell}\right) \implies \forall s \in S.\, \mathsf{ER}^{s \to T}(\vec{u}){\uparrow_p^{\ell,n}} \,. \tag{IH 3.2}$$

We show for $n = k + 1$:

$$\left(\forall s \in S.\, \mathsf{ER}^{s \to T}(\vec{u}){\uparrow_p^\ell}\right) \implies \forall s \in S.\, \mathsf{ER}^{s \to T}(\vec{u}){\uparrow_p^{\ell,k+1}} \,. \tag{P 3.2}$$

Let $\hat{\pi} \in \mathit{Paths}_k$ and $s \in S$ such that $\mathbf{P}(\hat{\pi}_k, s) \neq 0$. As T is absorbing, we observe that we can split the cumulative reward until reaching a state in T for the first time for path $\hat{\pi}s$ into this reward for path $\hat{\pi}$ and this reward for path $\hat{\pi}_k s$, i.e.,

$$r(\hat{\pi}s, \lozenge T) = r(\hat{\pi}, \lozenge T) + r(\hat{\pi}_k s, \lozenge T) \,. \tag{3.3}$$

To prove that Equation (P 3.2) holds we first observe that we can rewrite local monotonicity for $k + 1$ steps as follows (note that we leave out the instantiation $\vec{u} \in R$ for readability):

$$\begin{aligned}
&\sum_{\hat{\pi} \in \mathit{Paths}_{k+1}(s)} \left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \lozenge T) + \mathsf{ER}^{\hat{\pi}_{k+1} \to T}\right) \\
&= \sum_{\hat{\pi} \in \mathit{Paths}_k(s)} \sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p}\Pr(\hat{\pi}s')\right) \cdot \left(r(\hat{\pi}s', \lozenge T) + \mathsf{ER}^{s' \to T}\right) \\
&= \sum_{\hat{\pi} \in \mathit{Paths}_k(s)} \sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p}\left(\Pr(\hat{\pi}) \cdot \mathbf{P}(\hat{\pi}_k, s')\right)\right) \cdot \left(r(\hat{\pi}s', \lozenge T) + \mathsf{ER}^{s' \to T}\right) \\
&\overset{\text{Chain rule}}{=} \sum_{\hat{\pi} \in \mathit{Paths}_k(s)} \sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \left(\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \mathbf{P}(\hat{\pi}_k, s') + \Pr(\hat{\pi}) \cdot \left(\frac{\partial}{\partial p}\mathbf{P}(\hat{\pi}_k, s')\right)\right) \\
&\qquad \cdot \left(r(\hat{\pi}s', \lozenge T) + \mathsf{ER}^{s' \to T}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\hat{\pi}\in Paths_k(s)} \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \mathbf{P}\left(\hat{\pi}_k, s'\right) \cdot \left(r(\hat{\pi}s', \Diamond T) + \mathsf{ER}^{s'\to T}\right) \\
&\quad + \sum_{\hat{\pi}\in Paths_k(s)} \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \Pr(\hat{\pi}) \cdot \left(\frac{\partial}{\partial p}\mathbf{P}\left(\hat{\pi}_k, s'\right)\right) \cdot \left(r(\hat{\pi}s', \Diamond T) + \mathsf{ER}^{s'\to T}\right) \\
&\overset{Eq.\ (3.3)}{=} \sum_{\hat{\pi}\in Paths_k(s)} \left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \\
&\qquad\qquad \cdot \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \mathbf{P}\left(\hat{\pi}_k, s'\right) \cdot \left(r(\hat{\pi}, \Diamond T) + r(\hat{\pi}_k s', \Diamond T) + \mathsf{ER}^{s'\to T}\right) \\
&\quad + \sum_{\hat{\pi}\in Paths_k(s)} \Pr(\hat{\pi}_k) \cdot \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p}\mathbf{P}\left(\hat{\pi}_k, s'\right)\right) \cdot \left(r(\hat{\pi}s', \Diamond T) + \mathsf{ER}^{s'\to T}\right) \\
&= \sum_{\hat{\pi}\in Paths_k(s)} \left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \\
&\qquad\qquad \cdot \left(r(\hat{\pi}, \Diamond T) + \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \mathbf{P}\left(\hat{\pi}_k, s'\right) \cdot \left(r(\hat{\pi}_k s', \Diamond T) + \mathsf{ER}^{s'\to T}\right)\right) \qquad (3.4) \\
&\quad + \sum_{\hat{\pi}\in Paths_k(s)} \Pr(\hat{\pi}) \cdot \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p}\mathbf{P}\left(\hat{\pi}_k, s'\right)\right) \cdot r(\hat{\pi}s', \Diamond T) \qquad (3.5) \\
&\quad + \sum_{\hat{\pi}\in Paths_k(s)} \Pr(\hat{\pi}) \cdot \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p}\mathbf{P}\left(\hat{\pi}_k, s'\right)\right) \cdot \mathsf{ER}^{s'\to T} \qquad (3.6)
\end{aligned}
$$

We observe that we can rewrite the right-hand side of Equation (P 3.2) as

$$\Big(Eq.\ (3.4) + Eq.\ (3.5) + Eq.\ (3.6)\Big)(\vec{u}) \geq 0\ .$$

We now argue that for all $\vec{u} \in R$, we have that Equations (3.4) through (3.6) are at least 0, thus Equation (P 3.2) holds.

- Equation (3.4): By distinguishing between whether or not we reach a state in T on $\hat{\pi}$, we show that we can rewrite

$$\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \Diamond T) + \sum_{s'\in \mathrm{succ}(\hat{\pi}_k)} \mathbf{P}\left(\hat{\pi}_k, s'\right) \cdot \left(r(\hat{\pi}_k s', \Diamond T) + \mathsf{ER}^{s'\to T}\right)\right)$$

as

$$\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_k \to T}\right) .$$

– $\hat{\pi} \models T$: As T is absorbing (see Assumption 2.3) and $\hat{\pi} \models T$, we have $r(\hat{\pi}_k s', \Diamond T) = 0$, $\mathsf{ER}^{\hat{\pi}_k \to T}(\vec{u}) = 0$, and $\mathsf{ER}^{s' \to T}(\vec{u}) = 0$ for all $\vec{u} \in R$ and $s' \in \mathrm{succ}(\hat{\pi}_k)$ as T. We observe that we can rewrite Equation (3.4) as:

$$\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_k \to T}\right) .$$

– $\hat{\pi} \not\models T$: We obtain $r(\hat{\pi}_k s', \Diamond T) = r(\hat{\pi}_k)$. We thus have for any $\hat{\pi} \in Paths_k(s)$:

$$\begin{aligned}&\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \Diamond T) + r(\hat{\pi}_k) + \sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \mathbf{P}(\hat{\pi}_k, s') \cdot \left(\mathsf{ER}^{s' \to T}\right)\right) \\ &= \left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_k \to T}\right)\end{aligned}$$

From the induction hypothesis (Equation (IH 3.2)), we now obtain for all $\vec{u} \in R$:

$$\underbrace{\left(\sum_{\hat{\pi} \in Paths_k(s)} \left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_n \to T}\right)\right)}_{=\ \textit{Eq. (3.4)}}(\vec{u}) \geq 0 .$$

- Equation (3.5): we, first of all, observe that by the definition of the cumulative reward of a finite path, $r(\hat{\pi}s', \Diamond T)$ (page 20) is independent of s', thus we can threat this as a constant:

$$c = \begin{cases} r(\hat{\pi}_0) + r(\hat{\pi}_1) + \cdots + r(\hat{\pi}_k) & \text{if } \forall 0 \leq i \leq k.\, \hat{\pi}_i \not\models T \\ r(\hat{\pi}s', \Diamond T) & \text{otherwise .} \end{cases}$$

Secondly, we observe that $\sum_{s' \in \mathrm{succ}(s)} \mathbf{P}(s, s')(\vec{u}) = 1$ for all $\vec{u} \in R$, thus for its

derivative we have for all $\vec{u} \in R$:

$$\frac{\partial}{\partial p} \sum_{s' \in \mathrm{succ}(s)} \mathbf{P}(s, s')(\vec{u}) = \sum_{s' \in \mathrm{succ}(s)} \frac{\partial}{\partial p} \mathbf{P}(s, s')(\vec{u}) = 0 \,. \tag{3.7}$$

We thus obtain:

$$\sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p} \mathbf{P}\left(\hat{\pi}_k, s'\right) \right) \cdot c = 0 \cdot c \cdot |\mathrm{succ}(\hat{\pi}_k)| = 0 \,.$$

- Equation (3.6): we have $\Pr(\hat{\pi}) \geq 0$ for all $\hat{\pi} \in \mathit{Paths}_k(s)$, as we consider graph-preserving regions. Furthermore, we observe that the rightmost part of Equation (3.6) is at least 0, as we assume local monotonicity for all states (see Definition 3.2), so for all $\vec{u} \in R$:

$$\left(\sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p} \mathbf{P}\left(\hat{\pi}_k, s'\right) \right) \cdot \mathsf{ER}^{s' \to T} \right) (\vec{u}) \geq 0 \,.$$

We thus obtain for all $\vec{u} \in R$:

$$\left(\sum_{\hat{\pi} \in \mathit{Paths}_k(s)} \Pr(\hat{\pi}) \cdot \sum_{s' \in \mathrm{succ}(\hat{\pi}_k)} \left(\frac{\partial}{\partial p} \mathbf{P}\left(\hat{\pi}_k, s'\right) \right) \cdot \mathsf{ER}^{s' \to T} \right) (\vec{u}) \geq 0 \,.$$

From this, we conclude that Equation (3.2) holds, so Lemma 3.2 holds.

Obtaining global monotonicity | The following lemma claims that local monotonicity for n steps in the limit of n to ∞ implies global monotonicity.

Lemma 3.3

$$\lim_{n \to \infty} \mathsf{ER}^{s \to T}(\vec{u}) \uparrow_p^{\ell, n} \quad \text{implies} \quad \mathsf{ER}^{s_I \to T}(\vec{u}) \uparrow_p \,.$$

We can write the set of paths from $s \in S$ as the union of two disjoint sets, namely the set of paths that already reached a state in T, and those where T is not yet reached.

Proposition 3.4 Let $Paths_n(s) = Paths_n^T(s) \cup Paths_n^?(s)$ with

- $Paths_n^T(s) = \{\hat{\pi} \mid \hat{\pi} \in Paths_n(s) \wedge \hat{\pi} \models \Diamond T\}$ and
- $Paths_n^?(s) = \{\hat{\pi} \mid \hat{\pi} \in Paths_n(s) \wedge \hat{\pi} \not\models \Diamond T\}$.

$Paths_n^T(s)$ and $Paths_n^?(s)$ are disjoint.

Recall from Section 2.3.2 that for finite path $\hat{\pi} = s_0 s_1 \dots s_n$, the expected total reward until eventually reaching T in $\mathcal{D}$ is defined as:

$$r(\hat{\pi}, \Diamond T) = \begin{cases} r(s_0 s_1 \dots s_m) & \text{if } s_i \notin T \text{ for } 0 \leq i < m \leq n \text{ and } s_m \in T \\ r(\hat{\pi}) & \text{if } \hat{\pi} \not\models \Diamond T. \end{cases}$$

So we let, w.l.o.g., for all $\hat{\pi} \in Paths_n^T(s)$ and $\vec{u} \in R$: $\mathsf{ER}^{\hat{\pi}_n \to T}(\vec{u}) = 0$ as $\hat{\pi} \models \Diamond T$. Furthermore, from $\Pr^{s_I \to T}(\vec{u}) = 1$ we obtain for all $\vec{u} \in R$:

$$\lim_{n\to\infty} \left(\sum_{\hat{\pi} \in Paths_n^?(s)} \left(\frac{\partial}{\partial p} \Pr(\hat{\pi}) \right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_n \to T} \right) \right) (\vec{u}) = 0 \,. \tag{3.8}$$

As T is absorbing, we have for $n \to \infty$ for any $s \in S$ and $\hat{\pi}_n \in Paths_n^T(s)$:

$$\mathsf{ER}^{\hat{\pi}_n \to T} = 0 \,. \tag{3.9}$$

As the reward function is constant, we observe that:

$$\begin{aligned} &\lim_{n\to\infty} \left(\sum_{\hat{\pi} \in Paths_n^T(s)} \left(\frac{\partial}{\partial p} \Pr(\hat{\pi}) \right) \cdot r(\hat{\pi}, \Diamond T) \right) (\vec{u}) \\ &= \lim_{n\to\infty} \left(\sum_{\hat{\pi} \in Paths_n^T(s)} \left(\frac{\partial}{\partial p} \Pr(\hat{\pi}) \cdot r(\hat{\pi}, \Diamond T) \right) \right) (\vec{u}) \end{aligned} \tag{3.10}$$

Thus we obtain for all $\vec{u} \in R$:

$$\lim_{n\to\infty} \left(\sum_{\hat{\pi} \in Paths_n(s)} \left(\frac{\partial}{\partial p} \Pr(\hat{\pi}) \right) \cdot \left(r(\hat{\pi}, \Diamond T) + \mathsf{ER}^{\hat{\pi}_n \to T} \right) \right) (\vec{u}) \geq 0$$

$$\iff \lim_{n\to\infty}\left(\sum_{\hat{\pi}\in Paths_n^T(s)}\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right)\cdot\left(r(\hat{\pi},\lozenge T)+\mathsf{ER}^{\hat{\pi}_n\to T}\right)\right.$$
$$\left.+\sum_{\hat{\pi}\in Paths_n^?(s)}\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right)\cdot\left(r(\hat{\pi},\lozenge T)+\mathsf{ER}^{\hat{\pi}_n\to T}\right)\right)(\vec{u})\geq 0$$
$$\overset{\text{Eq. (3.8)}}{\iff} \lim_{n\to\infty}\left(\sum_{\hat{\pi}\in Paths_n^T(s)}\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right)\cdot\left(r(\hat{\pi},\lozenge T)+\mathsf{ER}^{\hat{\pi}_n\to T}\right)\right)(\vec{u})\geq 0$$
$$\overset{\text{Eq. (3.9)}}{\iff} \lim_{n\to\infty}\left(\sum_{\hat{\pi}\in Paths_n^T(s)}\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\right)\cdot r(\hat{\pi},\lozenge T)\right)(\vec{u})\geq 0$$
$$\overset{\text{Eq. (3.10)}}{\iff} \lim_{n\to\infty}\left(\sum_{\hat{\pi}\in Paths_n^T(s)}\left(\frac{\partial}{\partial p}\Pr(\hat{\pi})\cdot r(\hat{\pi},\lozenge T)\right)\right)(\vec{u})\geq 0$$
$$\iff \sum_{\pi\in Paths_{inf}(s)}\left(\left(\frac{\partial}{\partial p}\Pr(\pi)\right)\cdot r(\pi,\lozenge T)\right)(\vec{u})\geq 0$$
$$\iff \underbrace{\frac{\partial}{\partial p}\left(\sum_{\pi\in Paths_{inf}(s)}\Pr(\pi)\cdot r(\pi,\lozenge T)\right)(\vec{u})}_{=\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})}\geq 0$$

Lemma 3.3 follows from the above and the cylinder set construction for the expected total reward solution function (see Section 2.3).
Now Theorem 3.1 directly follows from Lemmas 3.2 and 3.3. ■

Example 3.3 Reconsider our running example pMC $\mathcal{D}_1$ of Figure 3.1. Recall from Example 3.2 that for $s \in \{s_0, s_2\}$, $\mathsf{ER}^{s\to\ddot{\smile}}(p)\uparrow_p^\ell$. For all $s \in S \setminus \{s_0, s_2\}$, we obtain $\left(\sum_{s'\in succ(s)}\left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot\mathsf{ER}^{s'\to T}\right)(p) = 0$ as p does not occur at the transition probabilities of s. Therefore, $\mathsf{ER}^{s\to\ddot{\smile}}(p)\uparrow_p^\ell$ for all $s \in S$. Thus, we obtain $\mathsf{ER}^{s_I\to\ddot{\smile}}(p)\uparrow_p$.

Remark 3.2 Here, the state rewards are constants. Monotonicity in the presence of parametric rewards can be accommodated by extending the notion of local monotonicity by including $\frac{\partial}{\partial p} r(s)$. I.e., we replace Equation (3.1) of Definition 3.2 by

$$\left(\frac{\partial}{\partial p} r(s) + \sum_{s' \in \mathrm{succ}(s)} \left(\frac{\partial}{\partial p} \mathbf{P}(s,s') \right) \cdot \mathsf{ER}^{s' \to T} \right) (\vec{u}) \;\geq\; 0 \quad \text{for all } \vec{u} \in R \,.$$

3.3 A Sufficient Condition for Monotonicity

In Section 3.4, we show we could solve monotonicity checking with the help of an co-ETR formula (see Example 3.10) representing the expected total reward solution function. However, this is often intractable [BHHJ+20]. Therefore, we want to find sufficient conditions for monotonicity at s. These conditions are based on constructing a pre-order on the states of the pMC, the so-called *reward order*.

3.3.1 Reward order

reward order

Definition 3.4 An ordering relation $\preceq^{\mathsf{rew}}_{R,T} \subseteq S \times S$ is a *reward order* w.r.t. target set $T \subseteq S$ and region R if for all $s, t \in S$:

$$s \preceq^{\mathsf{rew}}_{R,T} t \quad \text{implies} \quad \mathsf{ER}^{s \to T}(\vec{u}) \;\leq\; \mathsf{ER}^{t \to T}(\vec{u}) \quad \text{for all } \vec{u} \in R \,.$$

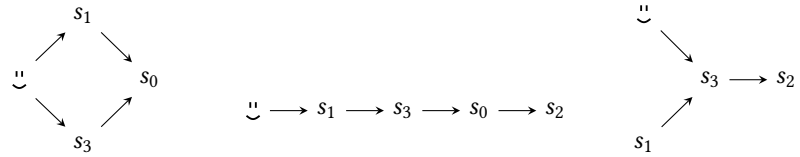
·exhaustive

The order $\preceq^{\mathsf{rew}}_{R,T}$ is called *exhaustive* if the reverse implication holds too.

For conciseness, we omit R and T from the ordering relation if they are clear from the context. We often denote the order as a Hasse-diagram, i.e., $G = (S, E)$ with $E = \{(s,t) \mid s,t \in S \wedge s \neq t \wedge s \preceq^{\mathsf{rew}} t \wedge (\nexists s' \in S \setminus \{s,t\}.\, s \preceq^{\mathsf{rew}} s' \preceq^{\mathsf{rew}} t)\}$. For reachability properties, we often refer to the order as the *reachability order*.

reachability order

Example 3.4 Reconsider our running example pMC $\mathcal{D}_1$ of Figure 3.1. Let region $R_1 = [0.8, 0.9]$, Figures 3.2a and 3.2b show reward orders for this pMC and region. Note that for this region we have:



(a) Reward Order (b) Exhaustive Order (c) Sufficient Order

Figure 3.2: Examples of orders for the pMC $\mathcal{D}_1$ in Figure 3.1

s	$\mathsf{ER}^{s\to \Cup}$
s_0	$[3.163, 3.565]$
s_1	1
s_2	$[4.4, 4.7]$
s_3	3
$\Cup$	0

For both s_0 and s_2, intervals occur. This is because we do not apply a single instantiation $\vec{u}$ to our function, but the region R_1, yielding the intervals.
The order in Figure 3.2a is not exhaustive, e.g., s_1 and s_3 are not ordered, although $\mathsf{ER}^{s_3\to \Cup} \leq \mathsf{ER}^{s_1\to \Cup}$. Also s_2 is missing, thus its ordering to s_0, s_1, s_3 and $\Cup$ is not contained in the order. The order in Figure 3.2b is exhaustive, as all states are fully ordered.

If s has two successors which are ordered, we can conclude local monotonicity of s in a region R:

Proposition 3.5 Let $s \in S$ and $\mathrm{succ}(s) = \{s_1, s_2\}$, with $\mathbf{P}(s, s_1) = p$ and $\mathbf{P}(s, s_2) = 1-p$. Then:

$$\text{for each region } R: \qquad s_2 \preceq^{\mathsf{rew}} s_1 \quad \text{implies} \quad \mathsf{ER}^{s\to T}(\vec{u})\uparrow_p^{\ell,R} .$$

Example 3.5 We continue with Example 3.4. For state s_2, we have $\Cup \preceq^{\mathsf{rew}} s_3$, thus $\mathsf{ER}^{\Cup\to\Cup}(p) \leq \mathsf{ER}^{s_3\to\Cup}(p)$ for any $p \in R_1$. Recall from Definition 3.2:

$$\text{if } \left(\sum_{s'\in \mathrm{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T}\right)(p) \;\geq\; 0 \text{ for all } p \in R, \text{ then } \mathsf{ER}^{s\to T}(p)\uparrow_p^{\ell} .$$

For s_2, we have for all $p \in R_1$:

$$\left(\sum_{s' \in \mathrm{succ}(s_2)} \left(\frac{\partial}{\partial p}\mathbf{P}(s_2, s')\right) \cdot \mathsf{ER}^{s' \to T}\right)(p) = 1 \cdot \underbrace{\mathsf{ER}^{s_3 \to \lightning}(p)}_{\geq 0} - 1 \cdot \underbrace{\mathsf{ER}^{\lightning \to \lightning}(p)}_{=0} .$$

Clearly, this is at least 0, thus $\mathsf{ER}^{s_2 \to T}(p)\uparrow_p^\ell$.

We extend this proposition to the case of several direct successor states of s. If those successors are ordered, and the transition functions are monotonic, we can obtain local monotonicity of s in a region R:

Lemma 3.6 Let $\mathrm{succ}(s) = \{s_1, \ldots, s_n\}$, $\mathbf{P}(s, s_i) = f_i$, and $\forall j > i.\, s_j \preceq^{\mathsf{rew}} s_i$. Then for each region R:

$$\exists i \in [1, \ldots, n].\, \left(\forall j \leq i.\, f_j(\vec{u})\uparrow_p^R \text{ and } \forall j > i.\, f_j(\vec{u})\downarrow_p^R\right)$$

$$\text{implies} \quad \mathsf{ER}^{s \to T}(\vec{u})\uparrow_p^{\ell,R} .$$

Proof. Let $\mathrm{succ}(s) = \{s_1, \ldots, s_n\}$ and $\mathbf{P}(s, s_i) = f_i$ for all $s_i \in \mathrm{succ}(s)$. Furthermore, let $s_j \preceq^{\mathsf{rew}} s_i$ for all $1 \leq i < j \leq n$, so we have $s_n \preceq^{\mathsf{rew}} s_{n-1} \preceq^{\mathsf{rew}} \ldots \preceq^{\mathsf{rew}} s_1$. Assume there exists $i \in [1, \ldots, n]$ such that for all $1 \leq j \leq i$ we have $f_j\uparrow_p^R$ and for all $i < j \leq n$ we have $f_j\downarrow_p^R$. Thus for all $\vec{u} \in R$ we obtain the weaker statement:

$$\sum_{j=0}^{i} \frac{\partial}{\partial p} f_j(\vec{u}) \geq 0, \text{ and } \sum_{j=i+1}^{n} \frac{\partial}{\partial p} f_j(\vec{u}) \leq 0 . \tag{3.11}$$

Furthermore, recall from Equation (3.7) that for the derivative of the transitions to successor states we have for all $\vec{u} \in R$:

$$\frac{\partial}{\partial p} \sum_{j=0}^{n} f_j(\vec{u}) = \sum_{j=0}^{n} \frac{\partial}{\partial p} f_j(\vec{u}) = 0 . \tag{3.12}$$

From Equations (3.11) and (3.12) we obtain for all $\vec{u} \in R$:

$$\sum_{j=0}^{i} \frac{\partial}{\partial p} f_j(\vec{u}) = -\sum_{j=i+1}^{n} \frac{\partial}{\partial p} f_j(\vec{u}) . \tag{3.13}$$

Recall from Definition 3.2:

$$\text{if } \left(\sum_{s' \in \mathrm{succ}(s)} \left(\frac{\partial}{\partial p} \mathbf{P}(s, s') \right) \cdot \mathsf{ER}^{s' \to T} \right)(\vec{u}) \;\geq\; 0 \text{ for all } \vec{u} \in R, \text{ then } \mathsf{ER}^{s \to T} \uparrow_p^{\ell, R}.$$

We now derive for arbitrary $\vec{u} \in R$:

$$\begin{aligned}
&\left(\sum_{s' \in \mathrm{succ}(s)} \left(\frac{\partial}{\partial p} \mathbf{P}(s, s') \right) \cdot \mathsf{ER}^{s' \to T} \right)(\vec{u}) \\
&= \left(\sum_{j=0}^{i} \left(\frac{\partial}{\partial p} f_j \right) \cdot \mathsf{ER}^{s_j \to T} + \sum_{j=i+1}^{n} \left(\frac{\partial}{\partial p} f_j \right) \cdot \mathsf{ER}^{s_j \to T} \right)(\vec{u}) \\
&\geq \left(\sum_{j=0}^{i} \left(\frac{\partial}{\partial p} f_j \right) \cdot \mathsf{ER}^{s_i \to T} + \sum_{j=i+1}^{n} \left(\frac{\partial}{\partial p} f_j \right) \cdot \mathsf{ER}^{s_n \to T} \right)(\vec{u}) && (3.14) \\
&= \left(\sum_{j=0}^{i} \left(\frac{\partial}{\partial p} f_j \right) \cdot \mathsf{ER}^{s_i \to T} - \sum_{j=0}^{i} \left(\frac{\partial}{\partial p} f_j \right) \cdot \mathsf{ER}^{s_n \to T} \right)(\vec{u}) && (3.15) \\
&= \left(\sum_{j=0}^{i} \left(\frac{\partial}{\partial p} f_j \right) \left(\mathsf{ER}^{s_i \to T} - \mathsf{ER}^{s_n \to T} \right) \right)(\vec{u}) \\
&\geq 0 && (3.16)
\end{aligned}$$

Equation (3.14) follows from $s_n \preceq^{\mathsf{rew}} s_j$ and $\frac{\partial}{\partial p} f_j(\vec{u}) \leq 0$ for all $1 \leq j < n$. Equation (3.15) directly follows from Equation (3.13) and $\mathsf{ER}^{s_n \to T}(\vec{u})$ being independent of j. Equation (3.16) follows from $s_n \preceq^{\mathsf{rew}} s_i$ for all $1 \leq i < n$, thus $\mathsf{ER}^{s_n \to T}(\vec{u}) \leq \mathsf{ER}^{s_i \to T}(\vec{u})$. ■

Example 3.6 We continue from Example 3.5. For state s_0, we have $s_1 \preceq^{\mathsf{rew}} s_3 \preceq^{\mathsf{rew}} s_2$, thus $\mathsf{ER}^{s_1 \to \breve{\cup}}(p) \leq \mathsf{ER}^{s_3 \to \breve{\cup}}(p) \leq \mathsf{ER}^{s_2 \to \breve{\cup}}(p)$. Let $f_i = \mathbf{P}(s_0, s_i)$ for $i \in \{1, 2, 3\}$. We have $f_1(p)\uparrow_p$, $f_2(p)\uparrow_p$, and $f_1\downarrow_p$. Applying Lemma 3.6 gives us $\mathsf{ER}^{s_0 \to \breve{\cup}}(p)\uparrow_p^{\ell}$.

Proposition 3.5 and Lemma 3.6 hint that it is sufficient for local monotonicity to only have the direct successors of the states ordered. Therefore, we focus on a so-called "sufficient" reward order:

·sufficient

Definition 3.5 Reward order $\preceq^{\mathsf{rew}}$ is *sufficient* for $s \in S$ if for all $s_1, s_2 \in \mathsf{succ}(s)$: $(s_1 \preceq^{\mathsf{rew}} s_2 \vee s_2 \preceq^{\mathsf{rew}} s_1)$ holds. The reward order is *sufficient* for pMC $\mathcal{D}$ if it is sufficient for all parametric states in $\mathcal{D}$.

Phrased differently, the reward order $\preceq^{\mathsf{rew}}$ is sufficient for p if $(\mathsf{succ}(s), \preceq^{\mathsf{rew}})$ is a total order for all $s \in S$ with $p \in \mathsf{occur}(s)$.

Example 3.7 Figure 3.2c shows a sufficient reward order for our running example pMC on region $R = [0.8, 0.9]$.

Definition 3.5 and Lemma 3.6 yield:

Corollary 3.7 Let pMC $\mathcal{D}$ be such that for each $s \in S$, $|\mathsf{succ}(s)| \leq 2$ and $\mathbf{P}(s, s')$ is monotonic for each $s' \in \mathsf{succ}(s)$. Then: $\preceq^{\mathsf{rew}}$ is sufficient for s implies $\mathsf{ER}^{s \to T}(\vec{u})$ is locally monotonic increasing/decreasing on any region R in all parameters.

3.4 Complexity

This section builds on [JKPW21]; Junges et al. describe the complexity landscape for the feasibility problem for reachability in pMCs (see also Section 2.4.1). In this section, we show that both monotonicity checking and computing a reward order are at least as hard as pMC verification. We also show that monotonicity checking is at least as hard as computing the reward order. Also, we show that all problems are in co-ETR.

Recall the decision problems as introduced in Section 2.4. To obtain the complexity of monotonicity checking and the complexity of computing a reward order, we restrict ourselves in the remainder of this section to $\bowtie := <$ for pMC-F and $\bowtie := \geq$ for pMC-V. Yielding the following problems:

- **(pMC-F)** *Given a pMC $\mathcal{D}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does there exist a $\vec{u} \in R$ such that $\mathsf{ER}_{\mathcal{D}}^{s \to T}(\vec{u}) < \lambda$?*
- **(pMC-V)** *Given a pMC $\mathcal{D}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does $\mathsf{ER}_{\mathcal{D}}^{s \to T}(\vec{u}) \geq \lambda$ hold for all $\vec{u} \in R$?*

Note that computing $\mathsf{ER}^{s \to \ddot{\smile}}(\vec{u})$ and deriving this function symbolically is intractable: the function can be exponentially large in the number of parameters [BHHJ+20]. We describe the decision problem whether two states are ordered by the expected total reward (pMC-RO) and the decision problem whether a pMC is monotonic (pMC-Mon) as follows:

- **(pMC-Mon)** *Given a pMC $\mathcal{D}$, a parameter $p \in V$, and a graph-preserving region R, does $\mathcal{D}{\uparrow}_p^R$ hold?*

- **(pMC-RO)** *Given a pMC $\mathcal{D}$, a graph-preserving region R, and two states s_1 and s_2, does $s_1 \preceq^{\mathsf{rew}}_{R,T} s_2$ hold, where $\preceq^{\mathsf{rew}}_{R,T}$ denotes the exhaustive reward order?*

PMC-V is in co-ETR | [JKPW21] provide an encoding for pMC-F for reachability properties. We can easily modify this encoding for expected total reward. An example is given in Example 3.8. From this, we obtain that pMC-F is in ETR. As pMC-V is the complement of pMC-F, we obtain that pMC-V is in co-ETR.

Example 3.8 Reconsider our running example pMC $\mathcal{D}_1$ of Figure 3.1, with reward function $r(s_i) = i$, target state $\lightning$, and region $R_1 = [0.8, 0.9]$. To encode the pMC-F problem with $\lambda = 3.2$ as an ETR problem, we use variables $\{x_s \mid s \in S\} \cup V$ where x_i encodes the expected total reward from state s_i and V are the parameters of the pMC. We encode the pMC by the conjunction of:

$$
\begin{aligned}
x_{\lightning} &= 0 \\
x_0 &= 0 + (1-p) \cdot x_1 + \frac{1}{2}p \cdot x_2 + \frac{1}{2}p \cdot x_3 \\
x_1 &= 1 + 1 \cdot x_{\lightning} \\
x_2 &= 2 + (1-p) \cdot x_{\lightning} + p \cdot x_3 \\
x_3 &= 3 + 1 \cdot x_{\lightning}
\end{aligned}
$$

We encode the region by:

$$0.8 \leq p \leq 0.9 \,.$$

Finally, we encode the bound for the pMC-F problem by:

$$x_0 < 3.2 \,.$$

If checking for satisfiability yields a solution, we have found a solution to pMC-F. If checking for satisfiability yields no solution, we have the co-ETR problem as for pMC-V.

PMC-RO is at least as hard as pMC-V | Determining whether two states are ordered by the exhaustive reward order is as hard as pMC verification and can be expressed in co-ETR (see Example 3.9).

Theorem 3.8 The pMC-V decision problem is polynomial-time reducible to pMC-RO, the decision problem whether two states are ordered by the exhaustive reward order.

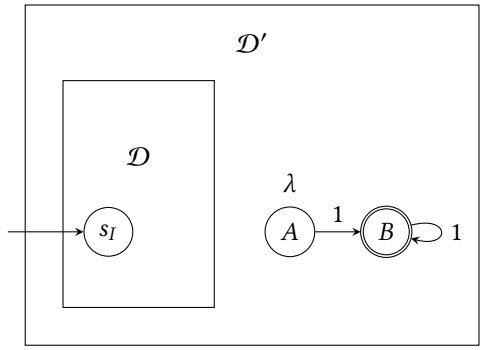


Figure 3.3: Outline of the construction for the proof of Theorem 3.8

Proof. Given an instance of pMC-V, with threshold λ and graph-preserving region R, we construct an instance of pMC-RO as follows (see also Figure 3.3): From $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ with reward function r, we construct a new $\mathcal{D}' = (S, s_I, T, V, \mathbf{P}')$ with reward function r' and additional states A, B. Let B be a target state. We add a transition from A to B with probability 1, and give A reward λ. Formally:

- $S' = S \uplus \{A, B\}$
- $s_I' = s_I$
- $T' = T \cup \{B\}$
- $V' = V$
- $\mathbf{P}'(s, s') = \begin{cases} \mathbf{P}(s, s') & \text{if } s, s' \in S \\ 1 & \text{if } s \in \{A, B\}, s' = B \\ 0 & \text{otherwise} \end{cases}$
- $r'(s) = \begin{cases} r(s) & \text{if } s \in S \\ \lambda & \text{if } s = A \\ 0 & \text{otherwise} \end{cases}$

Furthermore, we keep region R from pMC-V. This transformation clearly is in polynomial time.
Take $s_1 = A$ and $s_2 = s_I$ in pMC-RO. Note that the expected total reward for reaching the target from A in $\mathcal{D}'$ is λ for any parameter instantiation. Thus, $A \preceq^{\mathsf{rew}} s_I$ if and only if $\mathsf{ER}_{\mathcal{D}}^{s_I \to T}(\vec{u}) \geq \lambda$ for all $\vec{u} \in R$. ∎

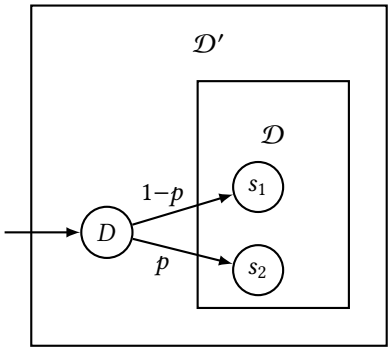


Figure 3.4: Outline of the construction for the proof of Lemma 3.9

Example 3.9 Reconsider our running example. Consider pMC-RO to check $s_1 \prec^{\mathsf{rew}} s_3$. We encode the pMC and the region as in Example 3.8. We encode the check for $s_1 \prec^{\mathsf{rew}} s_3$ as:

$$x_1 \geq x_3 \ .$$

If checking for satisfiability yields no solution, $s_1 \prec^{\mathsf{rew}} s_3$ follows.

PMC-Mon is at least as hard as pMC-RO (and pMC-V) | We show that pMC-RO is polynomial-time reducible to pMC-Mon. From this and Theorem 3.8, we obtain that pMC-V is polynomial-time reducible to pMC-Mon.

Lemma 3.9 The pMC-RO decision problem is polynomial-time reducable to pMC-Mon, the decision problem whether a pMC is monotonic.

Proof. Given an instance of pMC-RO for states s_1, s_2, with a graph-preserving region R, and reward function r, we construct an instance of pMC-Mon as follows. For $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$, we construct $\mathcal{D}' = (S', s_I', T, V', \mathbf{P}')$ by extending $\mathcal{D}$, see also Figure 3.4. We introduce a *fresh* parameter p and a fresh initial state D, and connect D with s_1 with probability $1 - p$, and D with s_2 with probability p. The reward for D is set to 0. Formally:

- $S' = S \uplus \{D\}$
- $s_I' = D$

- $V' = V \uplus \{p\}$
- $\mathbf{P}'(s,s') = \begin{cases} \mathbf{P}(s,s') & \text{if } s,s' \in S \\ 1-p & \text{if } s = D, s' = s_1 \\ p & \text{if } s = D, s' = s_2 \\ 0 & \text{otherwise} \end{cases}$
- $r'(s) = \begin{cases} r(s) & \text{if } s \in S \\ 0 & \text{otherwise} \end{cases}$

We extend region R by taking the product of R with any set of graph-preserving instantiations for fresh parameter p. This transformation clearly is in polynomial time.
We obtain the following equalities:

$$\begin{aligned} \mathsf{ER}^{D\to T}_{\mathcal{D}'}(\vec{u}) &= \left((1-p)\cdot \mathsf{ER}^{s_1\to T}_{\mathcal{D}'} + p\cdot \mathsf{ER}^{s_2\to T}_{\mathcal{D}'}\right)(\vec{u}) \\ &= \left((1-p)\cdot \mathsf{ER}^{s_1\to T}_{\mathcal{D}} + p\cdot \mathsf{ER}^{s_2\to T}_{\mathcal{D}}\right)(\vec{u}) \end{aligned}$$

As p is a fresh parameter, the expected total rewards from s_1 and s_2 until reaching a target state are constant in p on R, i.e., for $s \in \{s_1, s_2\}$ we have $\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}_{\mathcal{D}}(\vec{u}) = 0$ for all $\vec{u} \in R$. Thus,

$$\begin{aligned} \frac{\partial}{\partial p}\mathsf{ER}^{D\to T}_{\mathcal{D}'}(\vec{u}) &= \frac{\partial}{\partial p}\left((1-p)\cdot \mathsf{ER}^{s_1\to T}_{\mathcal{D}} + p\cdot \mathsf{ER}^{s_2\to T}_{\mathcal{D}}\right)(\vec{u}) \\ &= \left(-\mathsf{ER}^{s_1\to T}_{\mathcal{D}} + \mathsf{ER}^{s_2\to T}_{\mathcal{D}}\right)(\vec{u}) \end{aligned}$$

If $\mathcal{D}'$ is monotonic increasing in p on R, then $\mathsf{ER}^{D\to T}_{\mathcal{D}'}(\vec{u}) \geq 0$ for all $\vec{u} \in R$.
Whether or not $s_1 \preceq^{\mathsf{rew}} s_2$ can now be obtained as follows:

- If $\mathcal{D}'$ is monotonic increasing in p on R, then $s_1 \preceq^{\mathsf{rew}} s_2$ as for all $\vec{u} \in R$ we have $\left(-\mathsf{ER}^{s_1\to T}_{\mathcal{D}} + \mathsf{ER}^{s_2\to T}_{\mathcal{D}}\right)(\vec{u}) \geq 0$. Thus the expected total reward for s_1 is always at most the expected total reward for s_2.
- If $\mathcal{D}'$ is not monotonic increasing in p on R, then there must exist an $\vec{u} \in R$ such that $\left(-\mathsf{ER}^{s_1\to T}_{\mathcal{D}} + \mathsf{ER}^{s_2\to T}_{\mathcal{D}}\right)(\vec{u}) < 0$, thus $s_1 \not\preceq^{\mathsf{rew}} s_2$. ■

Corollary 3.10 The pMC-V decision problem is polynomial-time reducible to pMC-Mon, the decision problem whether a pMC is monotonic.

Corollary 3.10 is an immediate corollary to Theorem 3.8 and Lemma 3.9. Deciding whether a pMC is monotonic can also be expressed as a co-ETR problem.

Example 3.10 Reconsider our running example. Assume we have the pMC-Mon problem and want to check $\mathcal{D}{\uparrow_p}$. We encode the pMC and the region as in Example 3.8. We let d_i encode the derivative of the expected total reward for state s_i. We encode the derivatives as follows:

$$
\begin{aligned}
d_{\lightning} &= 0\\
d_0 &= (1-p)\cdot d_1 - x_1 + \frac{1}{2}p\cdot d_2 + \frac{1}{2}\cdot x_2 + \frac{1}{2}p\cdot\frac{\partial}{\partial p}x_0 + \frac{1}{2}\cdot x_3\\
d_1 &= d_{\lightning}\\
d_2 &= (1-p)\cdot d_{\lightning} - d_{\lightning} + p\cdot d_3 + d_3\\
d_3 &= d_{\lightning}
\end{aligned}
$$

We encode the monotonicity check as:

$$d_0 < 0\,.$$

If checking for satisfiability yields no solution, $\mathcal{D}{\uparrow_p}$ follows.

3.5 Automatically Proving Monotonicity

Our aim is to construct a sufficient reward order. The construction is inspired by 1) an observation, 2) graph rules, 3) state assumptions, and 4) rules for bounds on states. To be able to construct the reward order, let us introduce some notation.

Let $\prec^{\mathsf{rew}}_{R,T}$ denote the irreflexive variant of $\preceq^{\mathsf{rew}}_{R,T}$ and let $s \equiv^{\mathsf{rew}}_{R,T} s'$ denote $\mathsf{ER}^{s\to T}(\vec{u}) = \mathsf{ER}^{s'\to T}(\vec{u})$ for all $\vec{u} \in R$, i.e., $s \equiv^{\mathsf{rew}}_{R,T} s'$ iff $s \preceq^{\mathsf{rew}}_{R,T} s'$ and $s' \preceq^{\mathsf{rew}}_{R,T} s$.

We let $s \prec^{\mathsf{rew}}_{R,T} X$ ($X \prec^{\mathsf{rew}}_{R,T} s$) for $X \subseteq S$ be shorthand notation for $s \prec^{\mathsf{rew}}_{R,T} s'$ ($s' \prec^{\mathsf{rew}}_{R,T} s$) for all $s' \in X$. Equivalently, we define $s \equiv^{\mathsf{rew}}_{R,T} X$, $X \equiv^{\mathsf{rew}}_{R,T} s$, $s \preceq^{\mathsf{rew}}_{R,T} X$, and $X \preceq^{\mathsf{rew}}_{R,T} s$. Let $[s]^{\mathsf{rew}}_{R,T}$ denote the set of all $s' \in S$ such that $s \equiv^{\mathsf{rew}}_{R,T} s'$. We again omit R, T for conciseness. For $X \subseteq S$, $\mathsf{ub}(X) = \{s \in S \mid X \preceq^{\mathsf{rew}} s\}$ and $\mathsf{lb}(X) = \{s \in S \mid s \preceq^{\mathsf{rew}} X\}$ denote the upward and downward closure of X w.r.t. reward order $\preceq^{\mathsf{rew}}$. Furthermore, let $\min(X) = \{x \in X \mid \nexists x' \in X.\, x' \preceq^{\mathsf{rew}} x\}$, and $\max(X) = \{x \in X \mid \nexists x' \in X.\, x \preceq^{\mathsf{rew}} x'\}$.

3.5.1 An observation

By the definition of the expected total reward, we have $\mathsf{ER}^{t\to T}(\vec{u}) = 0$ for all $t \in T$. Furthermore, as rewards (and probabilities) are non-negative, we obtain through the recursive definition of the expected total reward function (see page 24): $\mathsf{ER}^{s\to T}(\vec{u}) \geq 0$ for all $s \in S \setminus T$. From this and Assumption 2.5 (page 21), we obtain for all $s \in S \setminus T, t \in T$:

$$t \prec^{\mathsf{rew}} s \,. \tag{3.17}$$

Note that, as all states have the states in T as downward closure and $T \neq \emptyset$, $\mathsf{lb}(X) \neq \emptyset$ for each $X \subseteq S$.

Remark 3.3 For a pMC $\mathcal{D}$ with a reachability property, the upward closure is non-empty, i.e., $\mathsf{ub}(X) \neq \emptyset$ for each $X \subseteq S$. Recall the transformation of $\mathcal{D}$ with target state $\lightning$ (see Section 2.3.2) into $\mathcal{D}'$ with target state $\lightning'$. By construction we have for all $s \in S' : \lightning \in \mathsf{ub}(\{s\})$.

Example 3.11 Reconsider the pMC of Figure 3.1 with target state $\lightning$. From Equation (3.17) we obtain for all $i \in \{0, 1, 2, 3\} : \lightning \prec^{\mathsf{rew}} s_i$.

3.5.2 Graph rules

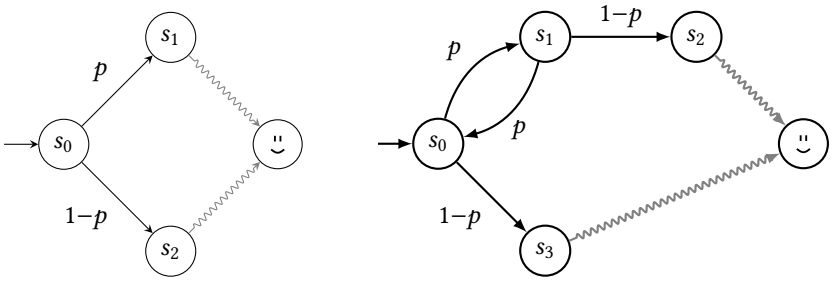
The graph structure of the pMC can be used to obtain the reward order. First of all, we reason about acyclic pMCs. Afterwards, we focus on cyclic ones.

Acyclic pMCs | For acyclic pMCs, we try to order states in a backward manner, i.e., given that the ordering of the successor states of s are known, we try to order s.

Lemma 3.11 For $s \in S \setminus T$:

1. if $r(s) > 0$, then $\mathsf{lb}(\mathsf{succ}(s)) \prec^{\mathsf{rew}} s$,
2. if $r(s) = 0$ and $\mathsf{ub}(\mathsf{succ}(s)) \neq \emptyset$, then
   either $\mathsf{succ}(s) \subseteq [s]^{\mathsf{rew}}$ or $\mathsf{lb}(\mathsf{succ}(s)) \prec^{\mathsf{rew}} s \prec^{\mathsf{rew}} \mathsf{ub}(\mathsf{succ}(s))$.

Proof. We show the proof for Lemma 3.11.1. Assume $s_1 \in \mathsf{lb}(\mathsf{succ}(s))$. By definition of the downward closure, we have for all $s_2 \in \mathsf{succ}(s) \setminus \{s_1\}$: $s_1 \preceq^{\mathsf{rew}} s_2$. Let $r(s) > 0$. We now have:



(a) Acyclic pMC $\mathcal{D}_3$ (b) Cyclic pMC $\mathcal{D}_4$

Figure 3.5: Sample pMCs to clearify the graph rules

$$\begin{aligned}
\mathsf{ER}^{s\to T}(\vec{u}) &= r(s) + \left(\sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathsf{ER}^{s'\to T}\right)(\vec{u}) \\
&\geq r(s) + \left(\sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathsf{ER}^{s_1\to T}\right)(\vec{u}) \\
&= r(s) + \mathsf{ER}^{s_1\to T}(\vec{u}) \\
&> \mathsf{ER}^{s_1\to T}(\vec{u})
\end{aligned}$$

From this Lemma 3.11.1 directly follows. Lemma 3.11.2 is obtained by a similar reasoning. ■

If $\mathsf{lb}(\mathsf{succ}(s)) \subseteq \mathsf{succ}(s)$, Lemma 3.11.1 states that if the reward obtained for our current state s is strictly greater than 0, s will lie above the successor state with the lowest expected total reward. Furthermore, Lemma 3.11.2 states that if the reward obtained for our current state s is 0 and the upward closure of the successor states of s is non-empty, then either s lies in the order at the successor states (in case all successors have the same expected total reward), or s lies between the downward and upward closure of the successors.

Example 3.12 Consider the pMC $\mathcal{D}_3$ of Figure 3.5a. Let $r_1(s_0) = 0$. Assume $s_2 \prec^{\mathsf{rew}} s_1$. Then, $s_2 \in \mathsf{lb}(\mathsf{succ}(s_0))$ and $s_1 \in \mathsf{ub}(\mathsf{succ}(s_0))$, thus by Lemma 3.11.2, we obtain $s_2 \prec^{\mathsf{rew}} s_0 \prec^{\mathsf{rew}} s_1$. Now assume $s_2 \equiv^{\mathsf{rew}} s_1$, we have $\mathsf{lb}(\mathsf{succ}(s_0)) = \{s_1, s_2\}$ and $\mathsf{ub}(\mathsf{succ}(s_0)) = \{s_1, s_2\}$. Therefore, $\mathsf{succ}(s_0) \subseteq [s_0]^{\mathsf{rew}}$ and $s_0 \equiv^{\mathsf{rew}} s_1 \equiv^{\mathsf{rew}} s_2$.

Algorithm 1 Construction of a reward order

Input: acyclic pMC $\mathcal{D} = (S, s_I, \{\lightning\}, V, \mathbf{P})$
Output: a reward order $\preceq^{\mathsf{rew}}$

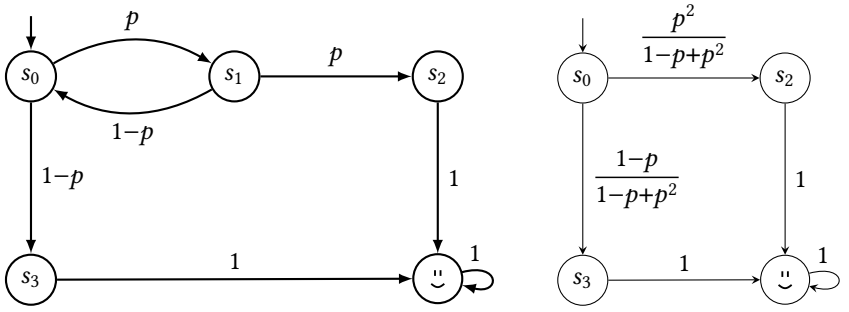
1: $S' \leftarrow S \setminus \{\lightning\}$, $\preceq^{\mathsf{rew}} \leftarrow \{\lightning\} \times S'$
2: **while** $S' \neq \emptyset$ **do**
3: select $s \in S'$ with s topologically last w.r.t. $\lightning$
4: **if** $\preceq^{\mathsf{rew}}$ is a total order for $\mathsf{succ}(s)$ **then**
5: **if** $r(s) = 0$ **then**
6: **if** $\exists s' \in \mathsf{succ}(s)$ s.t. $\mathsf{succ}(s) \subseteq [s']^{\mathsf{rew}}$ **then**
7: extend $\preceq^{\mathsf{rew}}$ with: $s \equiv^{\mathsf{rew}} \mathsf{succ}(s)$ *Lemma 3.11.2*
8: **else**
9: extend $\preceq^{\mathsf{rew}}$ with:
$\max(\mathsf{lb}(\mathsf{succ}(s))) \prec^{\mathsf{rew}} s$ and $s \prec^{\mathsf{rew}} \min(\mathsf{ub}(\mathsf{succ}(s)))$ *Lemma 3.11.2*
10: **else**
11: extend $\preceq^{\mathsf{rew}}$ with: $\max(\mathsf{lb}(\mathsf{succ}(s))) \prec^{\mathsf{rew}} s$ *Lemma 3.11.1*
12: $S' \leftarrow S' \setminus \{s\}$
13: **return** $\preceq^{\mathsf{rew}}$

Now let $r_2(s_0) = 2$. Assume $s_2 \prec^{\mathsf{rew}} s_1$, we then have $s_2 \in \mathsf{lb}(\mathsf{succ}(s_0))$ and $s_1 \in \mathsf{ub}(\mathsf{succ}(s_0))$. By Lemma 3.11.1, we obtain $s_2 \prec^{\mathsf{rew}} s_0$. However, we do not obtain $s_0 \prec^{\mathsf{rew}} s_1$ or $s_1 \prec^{\mathsf{rew}} s_0$, as we do not know the exact value of $\mathsf{ER}^{s_1 \to \lightning}(p)$ and $\mathsf{ER}^{s_2 \to \lightning}(p)$. The table below shows the intervals for the expected total reward given region $R = [0.1, 0.5]$, given fixed values for $\mathsf{ER}^{s_1 \to \lightning}(p)$ and $\mathsf{ER}^{s_2 \to \lightning}(p)$.

$\mathsf{ER}^{s_1 \to \lightning}$	$\mathsf{ER}^{s_2 \to \lightning}$	$\mathsf{ER}^{s_0 \to \lightning}$	Rew. Order
5	0	$[2.5, 4.5]$	$s_0 \prec^{\mathsf{rew}} s_1$
5	4	$[6.1, 6.5]$	$s_1 \prec^{\mathsf{rew}} s_0$
5	2	$[4.3, 5.5]$	Unknown

As can be seen from the table, $\mathsf{ER}^{s_0 \to \lightning}(p)$ heavily depends on the expected total reward for s_1 and s_2. Ordering $\mathsf{ER}^{s_0 \to \lightning}(p)$ with $r(s_0) > 0$ is therefore not trivial.

The reasoning of Equation (3.17) and Lemma 3.11 suggests Algorithm 1. It takes as input an acyclic pMC and computes a reward order $\preceq^{\mathsf{rew}}$ by going over all the states of the pMC. Initially, the reward order consists of the trivial order of Equation (3.17) combined with Assumption 2.5, and S' consists of all states of the pMC except for the target state $\lightning$. We now iteratively pick a state s topologically last w.r.t. $\lightning$ (line 3). A simple breadth-first search can compute this. Lemma 3.11 is then used to build the



(a) pMC $\mathcal{D}_5$ (b) $\mathcal{D}_5$ after SCC elimination

Figure 3.6: An example pMC with a single non-trivial SCC

reward order. Note that in line 4, we check if the ordering is a total order for $\mathsf{succ}(s)$. This ensures that we order s relative to its successors, i.e., if $\preceq^{\mathsf{rew}}$ is a total order for $\mathsf{succ}(s)$, then $\max(\mathsf{lb}(\mathsf{succ}(s))), \min(\mathsf{ub}(\mathsf{succ}(s))) \in \mathsf{succ}(s)$. We do this as we want to create an order that is *as dense as possible*, i.e., if the order contains s_1, s_2, s_3 and we could obtain an order $\prec^{\mathsf{rew}}$ with $s_1 \prec^{\mathsf{rew}} s_2 \prec^{\mathsf{rew}} s_3$, then the order $\prec^{\mathsf{rew}'}$ containing $s_1 \prec^{\mathsf{rew}'} s_2$ and $s_1 \prec^{\mathsf{rew}'} s_3$ is less dense as $\prec^{\mathsf{rew}}$. A denser order increases the chances that for a next state the successor states are ordered. If we now order s to a state s' that is not a direct successor of s, the chances that there is a state already contained in the order that should be between s and s' increases. Of course a drawback of this approach is that we might not be able to add s to $\preceq^{\mathsf{rew}}$ even though $\mathsf{lb}(\mathsf{succ}(s)$ and $\mathsf{ub}(\mathsf{succ}(s)$ are non-empty. We make up for this drawback in Sections 3.5.3 and 3.5.4.

Cyclic pMCs | The backward reasoning of Lemma 3.11 might not work if we have a cycle. The successor states of s with s on a cycle do not necessarily lie topologically after s, i.e., if you apply breadth-first search starting from the target (☹) you might find s before finding all of its successors.

One way to deal with this is to apply SCC elimination [JCVW+14]. Each SCC is merged into a set of states, one for each entry state of the SCC.

Example 3.13 Figure 3.6a shows a pMC $\mathcal{D}_5$ with a single non-trivial SCC. For reward function $r(s_i) = i$, Figure 3.6b shows $\mathcal{D}_5$ after SCC elimination. Note that the reward function also changes, i.e., we now have $r'(s_0) = p/1-p(1-p)$, $r'(s_2) = 2 \cdot (1-p)+p^2/1-p(1-p)$, and $r'(s_3) = 3 \cdot (1-p)+p^2/1-p(1-p)$. If $r(s_0) > 0$, we additionally get $r(s_0)$ in the changed reward: $r'(s_0) = r(s_0)+p/1-p(1-p)$.

The problem with SCC elimination is twofold. First of all, for pMCs with expected

reward properties, SCC elimination might introduce non-constant rewards. Even though we can handle this, as Remark 3.2 suggests, we want to avoid it as it makes checking local monotonicity cumbersome. Secondly, if the pMC consists of a single SCC, SCC elimination boils down to calculation of the solution function, which we want to avoid as it is intractable [BHHJ+20].

The following example suggests a graph-based rule for cyclic pMCs.

Example 3.14 Consider pMC $\mathcal{D}_4$ of Figure 3.5b. Let s_1 be the state we want to order. Based on the reasoning seen so far, we cannot order s_1 w.r.t. its successors, as s_0 lies topologically after s_1. Now assume, $s_2 \prec^{\mathsf{rew}} s_1$ is given. If $r(s_1) = 0$, we obtain $s_1 \prec^{\mathsf{rew}} s_0$:

$$\begin{aligned}
&\mathsf{ER}^{s_1 \to \ddot{\smile}}(p) = r(s_1) + p \cdot \mathsf{ER}^{s_0 \to \ddot{\smile}}(p) + (1-p) \cdot \mathsf{ER}^{s_2 \to \ddot{\smile}}(p) \\
&\implies \mathsf{ER}^{s_1 \to \ddot{\smile}}(p) < p \cdot \mathsf{ER}^{s_0 \to \ddot{\smile}}(p) + (1-p) \cdot \mathsf{ER}^{s_1 \to \ddot{\smile}}(p) \\
&\iff p \cdot \mathsf{ER}^{s_1 \to \ddot{\smile}}(p) < p \cdot \mathsf{ER}^{s_0 \to \ddot{\smile}}(p) \\
&\iff \mathsf{ER}^{s_1 \to \ddot{\smile}}(p) < \mathsf{ER}^{s_0 \to \ddot{\smile}}(p).
\end{aligned}$$

As in Example 3.12, we cannot order s_1 w.r.t. s_0 if $r(s_1) > 0$.

Based on the example above, we introduce the following Lemma:

Lemma 3.12 For any state s with $\mathrm{succ}(s) = \{s_1, s_2\}$ the following holds:

1. if $s_1 \equiv^{\mathsf{rew}} s$ and $r(s) = 0$, then $s_2 \equiv^{\mathsf{rew}} s$,
2. if $s_1 \equiv^{\mathsf{rew}} s$ and $r(s) > 0$, then $s_2 \prec^{\mathsf{rew}} s$,
3. if $s \prec^{\mathsf{rew}} s_1$, then $s_2 \prec^{\mathsf{rew}} s$, and
4. if $s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0$, then $s \prec^{\mathsf{rew}} s_2$.

Proof. The proof follows directly from Definition 3.4. ∎

Remark 3.4 Eliminating SCCs is a viable approach if the SCC is a small fragment of the pMC.

To deal with cycles, we extend Algorithm 1 by Algorithm 2, which implements the different cases of Lemma 3.12.

Algorithm 2 Cycle extension (put after line 11 in Algorithm 1)

1: **if** succ(s) = $\{s_1, s_2\}$ and $\preceq^{\mathsf{rew}}$ is not a total order for $\{s_1, s_2\}$ **then**
2: **if** $s_1 \equiv^{\mathsf{rew}} s$ **then**
3: **if** $r(s) = 0$ **then**
4: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \equiv^{\mathsf{rew}} s$ *Lemma 3.12.1*
5: **else**
6: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$ *Lemma 3.12.2*
7: **else if** $s \prec^{\mathsf{rew}} s_1$ **then**
8: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$ *Lemma 3.12.3*
9: **else if** $s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0$ **then**
10: extend $\preceq^{\mathsf{rew}}$ with: $s \prec^{\mathsf{rew}} s_2$ *Lemma 3.12.4*

Example 3.15 We continue from Example 3.14. If we cannot order s_1 w.r.t. either s_0 or s_2 and we cannot order s_0 and s_2 we cannot obtain a sufficient reward order from Lemmas 3.11 and 3.12.

3.5.3 State assumptions

As Example 3.15 suggests, Algorithms 1 and 2 do not necessarily yield a sufficient reward order. Therefore, we introduce state assumptions for the reward order.

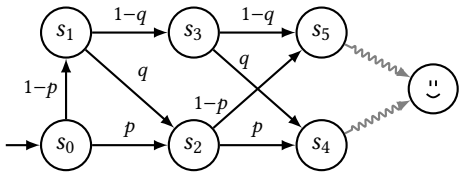
Making state assumptions | The following example motivates the use of state assumptions to obtain a sufficient reward order.

Example 3.16 Consider pMC $\mathcal{D}_6$ in Figure 3.7a, with $r(s_i) = 0$. Assume we somehow know $s_4 \prec^{\mathsf{rew}} s_5$. Following the lines of Lemma 3.11 we get the reward order as in Figure 3.7b. Clearly, we cannot order s_2 and s_3, however any region R can be partitioned into three (potentially empty) sub-regions: A region with $s_2 \prec^{\mathsf{rew}} s_3$, a region with $s_2 \equiv^{\mathsf{rew}} s_3$, and a region with $s_3 \prec^{\mathsf{rew}} s_2$. We create three copies of the order, each containing one of the state assumptions. In the copy assuming $s_2 \prec^{\mathsf{rew}} s_3$, we can order s_1 as in Figure 3.7c. The other copies in Figure 3.7c reflect $s_3 \prec^{\mathsf{rew}} s_2$ and $s_2 \equiv^{\mathsf{rew}} s_3$, respectively.

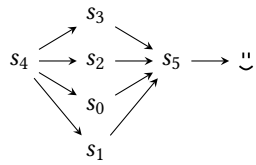
Let $\mathcal{A} = (\mathcal{A}_{\prec}, \mathcal{A}_{\equiv})$ be a pair of sets of state assumptions such that $(s,t) \in \mathcal{A}_{\prec}$ means $s \prec^{\mathsf{rew}} t$ while $(s,t) \in \mathcal{A}_{\equiv}$ means $s \equiv^{\mathsf{rew}} t$.

Definition 3.6 Let $\preceq^{\mathsf{rew}}$ be a reward order, and $\mathcal{A} = (\mathcal{A}_{\prec^{\mathsf{rew}}}, \mathcal{A}_{\equiv^{\mathsf{rew}}})$ with *state assumptions* $\mathcal{A}_{\prec^{\mathsf{rew}}}, \mathcal{A}_{\equiv^{\mathsf{rew}}} \subseteq S \times S$. Then $\preceq^{\mathsf{rew},\mathcal{A}}$ is a reward order with state

·state assumption



(a) Example pMC $\mathcal{D}_6$ with $s_4 \prec^{\mathsf{rew}} s_5$



(b) Reward order following Lemma 3.11

Assume $s_2 \prec^{\mathsf{rew}} s_3$: $s_4 \longrightarrow s_2 \longrightarrow s_0 \longrightarrow s_1 \longrightarrow s_3 \longrightarrow s_5 \longrightarrow \text{☹}$

Assume $s_3 \prec^{\mathsf{rew}} s_2$: $s_4 \longrightarrow s_3 \longrightarrow s_1 \longrightarrow s_0 \longrightarrow s_2 \longrightarrow s_5 \longrightarrow \text{☹}$

Assume $s_2 \equiv^{\mathsf{rew}} s_3$: $s_4 \longrightarrow \{s_0, s_1, s_2, s_3\} \longrightarrow s_5 \longrightarrow \text{☹}$

(c) Reward orders with assumptions

Figure 3.7: Illustrating the use of state assumptions

assumptions where $\preceq^{\mathsf{rew},\mathcal{A}} = \left(\preceq^{\mathsf{rew}} \cup \mathcal{A}_{\prec^{\mathsf{rew}}} \cup \mathcal{A}_{\equiv^{\mathsf{rew}}}\right)$.

Lemma 3.13 If state assumptions $\mathcal{A} = (\mathcal{A}_{\prec^{\mathsf{rew}}}, \mathcal{A}_{\equiv^{\mathsf{rew}}})$ satisfy:

$$(s,t) \in \mathcal{A}_{\prec^{\mathsf{rew}}} \quad \text{implies} \quad \forall \vec{u} \in R.\, \mathsf{ER}^{s \to T}(\vec{u}) < \mathsf{ER}^{t \to T}(\vec{u}) \text{ and}$$
$$(s,t) \in \mathcal{A}_{\equiv^{\mathsf{rew}}} \quad \text{implies} \quad \forall \vec{u} \in R.\, \mathsf{ER}^{s \to T}(\vec{u}) = \mathsf{ER}^{t \to T}(\vec{u}),$$

·valid then $\preceq^{\mathsf{rew},\mathcal{A}}$ is a reward order, and we call $\mathcal{A}$ *(globally) valid*.

We now generalize Algorithms 1 and 2 to use state assumptions. The result is given in Algorithm 3. First, we introduce a queue (Queue) consisting of the annotated reward order, state assumptions, and states we still need to handle for this order. Instead of looping over the states (Algorithm 1 lines 2-12), we now loop over the items in the queue. We pop an order from the queue (Algorithm 3 line 3) and try to extend as in Algorithms 1 and 2. If this extension is successful, we push the extended order to the queue together with the states we still need to handle for the order and the already existing state assumptions. Otherwise, we create the three state assumptions and push three copies of the order to the queue, each with its own new assumption. Once the set of states we need to consider is empty, we annotated the order with the state assumptions made (Algorithm 3 line 5), such that we can distinguish between states ordered by the graph rules and states ordered through assumptions.

Algorithm 3 Construction of a reward order with assumptions

Input: pMC $\mathcal{D} = (S, s_I, \{\lightning\}, V, \mathbf{P})$
Output: Result = set of annotated orders $\preceq^{\mathsf{rew},\mathcal{A}}$

Result $\leftarrow \emptyset$, Queue $\leftarrow \big(S' : S \setminus \{\lightning\}, \mathcal{A} : \emptyset, \prec^{\mathsf{rew}}: \{\lightning\} \times S'\big)$
while Queue not empty **do**
  $S', \mathcal{A}, \preceq^{\mathsf{rew}} \leftarrow$ Queue.pop()
  if $S' = \emptyset$ **then**
    Result $\leftarrow$ Result $\cup \{\preceq^{\mathsf{rew},\mathcal{A}}\}$
  else
    select $s \in S'$ with s topologically last w.r.t. $\lightning$
    if $\preceq^{\mathsf{rew}}$ is a total order for succ(s) **then**
      Algorithm 1 lines 5-11
      Queue.push$(S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}})$
    else if succ$(s) = \{s_1, s_2\}$ and
        $\preceq^{\mathsf{rew}}$ is not a total order for $\{s_1, s_2\}$ and
        $\big(s_1 \equiv^{\mathsf{rew}} s$ or $s \prec^{\mathsf{rew}} s_1$ or $(s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0)\big)$ **then**
      Algorithm 2 lines 2-10
      Queue.push$(S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}})$
    else
      pick $s_1, s_2 \in$ succ(s) s.t. neither $s_1 \preceq^{\mathsf{rew}} s_2$ nor $s_2 \preceq^{\mathsf{rew}} s_1$
      Queue.push$(S', (\mathcal{A}_\prec \cup \{(s_1, s_2)\}, \mathcal{A}_\equiv), \preceq^{\mathsf{rew}}$ extended with $s_1 \prec^{\mathsf{rew}} s_2)$
      Queue.push$(S', (\mathcal{A}_\prec, \mathcal{A}_\equiv \cup \{(s_1, s_2)\}), \preceq^{\mathsf{rew}}$ extended with $s_1 \equiv^{\mathsf{rew}} s_2)$
      Queue.push$(S', (\mathcal{A}_\prec \cup \{(s_2, s_1)\}, \mathcal{A}_\equiv), \preceq^{\mathsf{rew}}$ extended with $s_2 \prec^{\mathsf{rew}} s_1)$
return Result

Theorem 3.14 For every order with assumptions $(\preceq^{\mathsf{rew},\mathcal{A}}, \mathcal{A})$ computed by Algorithm 3 the following holds:

If $\preceq^{\mathsf{rew},\mathcal{A}}$ is a reward order, then it is sufficient .

Discharging state assumptions | Even for simple pMCs (see page 17) the number of orders is worst-case exponential in the number of states [CJJK+18; JJWQ+18]. For states in pMCs obtained from POMDPs, the number of successor states is not bounded (see Appendix C). Therefore, we want to find ways to discharge state assumptions.

The following result states that state assumptions can sometimes be ignored.

Theorem 3.15 If all orders computed by Algorithm 3 are witnesses for a parameter to be monotonic increasing (decreasing), then the parameter is indeed monotonic increasing (decreasing).

Note that the orders are not necessarily globally valid for this theorem. Assume Algorithm 3 yields n sets of state assumptions $\mathcal{A}_i$ for $i \in [1 \dots n]$. These assumption sets partition a region R such that $R = \bigcup_i^n R_{\mathcal{A}_i}$. Note that the $R_{\mathcal{A}}$ might be empty, i.e., $\mathcal{A}$ does not hold in any sub-region. If for each $i \in [1 \dots n]$, $\preceq_i^{\mathsf{rew},\mathcal{A}}$ is a witness for monotonicity, and the witnesses agree on whether a parameter is increasing (decreasing), then the parameter is monotonic increasing (decreasing) on R.

Theorem 3.15 does not always apply, plus it still yields an exponential number of orders, as we can only discharge them after computations. Therefore, we introduce an on-the-fly method to validate state assumptions. Let $(\preceq^{\mathsf{rew},\mathcal{A}}, \mathcal{A})$ be the current order, and suppose we want to order s_1 and s_2. We go over all possible state assumptions, starting with checking whether $s_1 \prec^{\mathsf{rew}} s_2$ is valid on our region R. If $s_1 \prec^{\mathsf{rew}} s_2$ holds on the entire region, then we extend $\preceq^{\mathsf{rew},\mathcal{A}}$ with $s_1 \prec^{\mathsf{rew}} s_2$, and do not add this assumption to $\mathcal{A}$. We also ignore the possibilities $s_1 \equiv^{\mathsf{rew}} s_2$ and $s_2 \prec^{\mathsf{rew}} s_1$, as these cannot hold since $s_1 \prec^{\mathsf{rew}} s_2$ holds on the entire region. If $s_1 \not\prec^{\mathsf{rew}} s_2$ for the *entire* region, we do not assume $s_1 \prec^{\mathsf{rew}} s_2$ (and ignore the corresponding order). Both cases prune the number of orders. In case of an inconclusive result, $s_1 \prec^{\mathsf{rew}} s_2$ is added to $\mathcal{A}_{\prec^{\mathsf{rew}}}$. To check whether $s_1 \prec^{\mathsf{rew}} s_2$, we describe three techniques.

Using a local NLP. To refute a state assumption to be valid on the entire region, a single instantiation $\vec{u}$ refuting the state assumption suffices. This suggests to let a solver prove the absence of such a $\vec{u}$ by considering a fragment of the pMC. The idea is to locally (at s_1 and s_2) consider the pMC and its characterizing non-linear program (NLP) [BK08; BGKR+11; DJJC+15; CJJK+18], together with the inequalities encoded by $\preceq^{\mathsf{rew},\mathcal{A}}$.

Example 3.17 Reconsider pMC $\mathcal{D}_6$ of Figure 3.7a and let $s_4 \prec^{\mathsf{rew},\mathcal{A}} s_5$ and $r(s_i) = 0$ for $i \in \{0,1,2,3\}$. Assume we want to check $s_2 \prec^{\mathsf{rew}} s_3$ for $R = (0.5, 0.8) \times (0.1, 0.3)$. We encode this as the following conjunction, where x_i encodes the expected total reward for state s_i:

$$0.5 < p < 0.8 \ \wedge \ 0.1 < q < 0.3 \tag{3.18}$$

$$x_2 = p \cdot x_4 + (1-p) \cdot x_5 \tag{3.19}$$

$$x_3 = q \cdot x_4 + (1-q) \cdot x_5 \tag{3.20}$$

$$0 \leq x_4 < \infty \ \wedge \ 0 \leq x_5 < \infty \tag{3.21}$$
$$x_4 < x_5 \tag{3.22}$$
$$x_2 \geq x_3 \tag{3.23}$$

Equation (3.18) describes the region. Equations (3.19) and (3.20) encode the expected total reward of s_2 and s_3, respectively, note that $r(s)$ is left out, as it is 0. For states s_4, s_5, we know that the expected total reward is non-negative and finite (Equation (3.21)), and $s_4 \prec^{\mathsf{rew}} s_5$ (Equation (3.22)). Finally, to validate the state assumption $s_2 \prec^{\mathsf{rew}} s_3$, we add constraint $\neg(s_2 \prec^{\mathsf{rew}} s_3)$ (Equation (3.23)). If checking for satisfiability yields no solution, it follows $s_2 \prec^{\mathsf{rew}} s_3$. Otherwise, we do not know: the obtained instantiation $\vec{u}$ might be spurious, as bounds for x_4 and x_5 are not provided.

Using sampling and model checking. This approach targets to cheaply disprove state assumptions for the entire region R. The benefit of this, is that we can limit the use of the local NLP, if we already know there exists an $\vec{u} \in R$ such that our state assumption does not hold, the local NLP will also find an $\vec{u}$.

We use model checking in the following manner: we instantiate the pMC with instantiations $\vec{u}$ from a set U, and evaluate the (parameter-free) MC $\mathcal{D}[\vec{u}]$ using standard model checking. We reduce the amount of solver runs, similar to [DJJC+15]. We determine U by sampling the parameter space at suitable points as in, e.g., [CHHK+13; CCGK+18]. This sampling yields expected total reward $\mathrm{ER}^{s}_{\mathcal{D}[\vec{u}]}(\lozenge T)$ for every state s, and allows to disprove a state assumption, say $s_1 \prec^{\mathsf{rew}} s_2$, by looking up whether $\mathrm{ER}^{s_1}_{\mathcal{D}[\vec{u}]}(\lozenge T) \geq \mathrm{ER}^{s_2}_{\mathcal{D}[\vec{u}]}(\lozenge T)$ for some $u \in U$.

Example 3.18 Reconsider pMC $\mathcal{D}_6$ of Figure 3.7a. Let $R(p) = (0.5, 0.8)$ and $R(q) = (0.1, 0.3)$ and $r(s_i) = 0$ for $i \in \{0, 1, 2, 3\}$. Assume the expected total reward for s_4 and s_5 are independent of R, e.g., let $\mathsf{ER}^{s_4 \to \lightning}(p, q) = 15$ and $\mathsf{ER}^{s_5 \to \lightning}(p, q) = 5$. To check $s_2 \prec^{\mathsf{rew}} s_3$ we sample over p and q, e.g., sampling over the corners of the region gives us:

p	q	$\mathsf{ER}^{s_2 \to \lightning}$	$\mathsf{ER}^{s_3 \to \lightning}$
0.5	0.1	10	6
0.5	0.3	10	8
0.8	0.1	13	6
0.8	0.3	13	8

From this, it is easy to observe that $s_2 \prec^{\mathsf{rew}} s_3$ does not hold on the entire region.

Using region checking. Region verification procedures (see Section 5.2) consider a region R, and obtain for each state s bounds $L_R(s)$ and $U_R(s)$ such that:

$$L_R(s) \;\leq\; \mathsf{ER}^{s\to T}(\vec{u}) \;\leq\; U_R(s) \quad \text{for all } \vec{u} \in R\,.$$

Assumption $s_1 \prec^{\mathsf{rew}} s_2$ can be proven by checking $U_R(s_1) < L_R(s_2)$.

3.5.4 Using bounds

Instead of using bounds to discharge and prove state assumptions, we can also directly use them to order states. We assume for state s and region R to have bounds $L_R(s)$ and $U_R(s)$ at our disposal satisfying:

$$L_R(s) \;\leq\; \mathsf{ER}^{s\to T}(\vec{u}) \;\leq\; U_R(s) \quad \text{for all } \vec{u} \in R\,.$$

Such bounds can be trivially assumed to be 0 and ∞ respectively, but the idea is to obtain tighter bounds by exploiting parameter lifting (see Section 5.2). The following simple observation yields a cheap rule to order states.

Lemma 3.16 For $s_1, s_2 \in S$ and region R:

$$U_R(s_2) \leq L_R(s_1) \quad \text{implies} \quad s_2 \preceq^{\mathsf{rew}} s_1\,.$$

Note the similarity to using region checking to discharge state assumptions. If we additionally take into account the reward $r(s)$ at state s, we can also order s w.r.t. its successors.

Lemma 3.17 For any state s with $\mathsf{succ}(s) = \{s_1, s_2\}$, $f = \mathbf{P}(s, s_1)$, region R and $s_1 \preceq^{\mathsf{rew}} s_2$:

1. if $r(s) \geq f(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ for all $\vec{u} \in R$, then $s_2 \preceq^{\mathsf{rew}} s$ and
2. if $r(s) \leq f(\vec{u}) \cdot (L_R(s_2) - U_R(s_1))$ for all $\vec{u} \in R$, then $s \preceq^{\mathsf{rew}} s_2$.

Proof. We show the proof for Lemma 3.17.1. Let $f' = \mathbf{P}(s, s_2) = 1 - f$. We derive for all $\vec{u} \in R$:

$$\begin{aligned} r(s) &\geq f(\vec{u}) \cdot (U_R(s_2) - L_R(s_1)) \\ &\geq f(\vec{u}) \cdot \left(\mathsf{ER}^{s_2\to T}(\vec{u}) - \mathsf{ER}^{s_1\to T}(\vec{u})\right) \\ &= f(\vec{u}) \cdot \mathsf{ER}^{s_2\to T}(\vec{u}) - f(\vec{u}) \cdot \mathsf{ER}^{s_1\to T}(\vec{u}) \end{aligned}$$

$$= (1-f'(\vec{u})) \cdot \mathsf{ER}^{s_2 \to T}(\vec{u}) - f(\vec{u}) \cdot \mathsf{ER}^{s_1 \to T}(\vec{u})$$
$$= \mathsf{ER}^{s_2 \to T}(\vec{u}) - f'(\vec{u}) \cdot \mathsf{ER}^{s_2 \to T}(\vec{u}) - f(\vec{u}) \cdot \mathsf{ER}^{s_1 \to T}(\vec{u})$$

From this, it immediately follows for all $\vec{u} \in R$:

$$\underbrace{r(s) + f(\vec{u}) \cdot \mathsf{ER}^{s_1 \to T}(\vec{u}) + f'(\vec{u}) \cdot \mathsf{ER}^{s_2 \to T}(\vec{u})}_{=\mathsf{ER}^{s \to T}(\vec{u})} \geq \mathsf{ER}^{s_2 \to T}(\vec{u}) \,,$$

so $s_2 \preceq^{\mathsf{rew}} s$, thus Lemma 3.17.1 holds. Lemma 3.17.2 is obtained by a similar reasoning. ■

Example 3.19 Reconsider pMC $\mathcal{D}_6$ of Figure 3.7a, let $r(s_2) = 5$ and $R(p) = [0.1, 0.7]$. Assume we have the following bounds for s_4 and s_5: $L(s_4) = 3$, $U(s_4) = 4$, $L(s_5) = 5$, and $U(s_5) = 10$. From this, we obtain $L(s_2) = 5 + \min\{0.1 \cdot 3 + 0.9 \cdot 5, 0.7 \cdot 3 + 0.3 \cdot 5\} = 8.6$, $U(s_2) = 5 + \max\{0.1 \cdot 4 + 0.9 \cdot 10, 0.7 \cdot 4 + 0.3 \cdot 10\} = 14.4$. Observe that we cannot order s_2 w.r.t. s_5 with the help of Lemma 3.11, as $r(s_2) > 0$ and $s_4 \prec^{\mathsf{rew}} s_5$. Moreover, we cannot apply Lemma 3.16, as $U(s_5) > L(s_2)$ and $U(s_2) > L(s_5)$. However, by Lemma 3.17.1, we obtain $s_2 \preceq^{\mathsf{rew}} s_5$, as

$$\underbrace{r(s)}_{=5} \geq \underbrace{f(\vec{u})}_{=p \in [0.1,0.7]} \cdot \left(\underbrace{U_R(s_5)}_{=10} - \underbrace{L_R(s_4)}_{=3} \right) .$$

We generalize Lemma 3.17 to n successors as follows:

Theorem 3.18 Let $\mathrm{succ}(s) = \{s_1, \ldots, s_n\}$ with $\mathbf{P}(s, s_i) = f_i$ and $s_i \preceq^{\mathsf{rew}} s_j$ for all $1 \leq i < j < n$. For $1 \leq j \leq n$ we now have:

$$\text{if } r(s) \geq (1-f_j(\vec{u})) \cdot (U_R(s_j) - L_R(s_1)) \text{ for all } \vec{u} \in R, \text{ then } s_j \preceq^{\mathsf{rew}} s \,.$$

Proof. Let $f_i = \mathbf{P}(s, s_i)$ for any $1 \leq i \leq n$. Furthermore, let

$$s_i \preceq^{\mathsf{rew}} s_j \quad \text{for all } 1 \leq i < j < n \,. \tag{3.24}$$

We derive for all $\vec{u} \in R$:

$$\begin{aligned}
r(s) &\geq (1-f_j(\vec{u})) \cdot (U_R(s_j) - L_R(s_1)) \\
&= (1-f_j(\vec{u})) \cdot U_R(s_j) - (1-f_j(\vec{u})) \cdot L_R(s_1) \\
&= (1-f_j(\vec{u})) \cdot U_R(s_j) - \sum_{i \in \{1,\dots,n\} \setminus \{j\}} f_i(\vec{u}) \cdot L_R(s_1) \\
&\overset{\text{Eq. (3.24)}}{\geq} (1-f_j(\vec{u})) \cdot U_R(s_j) - \sum_{i \in \{1,\dots,n\} \setminus \{j\}} f_i(\vec{u}) \cdot L_R(s_i) \\
&\geq (1-f_j(\vec{u})) \cdot \mathsf{ER}^{s_j \to T}(\vec{u}) - \sum_{i \in \{1,\dots,n\} \setminus \{j\}} f_i(\vec{u}) \cdot \mathsf{ER}^{s_i \to T}(\vec{u}) \\
&= \mathsf{ER}^{s_j \to T}(\vec{u}) - \sum_{i=1}^{n} f_i(\vec{u}) \cdot \mathsf{ER}^{s_i \to T}(\vec{u})
\end{aligned}$$

From this, it immediately follows for all $\vec{u} \in R$:

$$\underbrace{r(s) + \sum_{i=1}^{n} f_i(\vec{u}) \cdot \mathsf{ER}^{s_i \to T}(\vec{u})}_{= \mathsf{ER}^{s \to T}(\vec{u})} \geq \mathsf{ER}^{s_j \to T}(\vec{u}) \,,$$

so $s_j \preceq^{\mathsf{rew}} s$. ∎

Note that the dual of Theorem 3.18, i.e.,

$$r(s) \leq (1-f_j(\vec{u})) \cdot (L_R(s_j) - U_R(s_n)) \quad \text{implies} \quad s_j \preceq^{\mathsf{rew}} s \,,$$

does not help building the order. As $s_j \preceq^{\mathsf{rew}} s_n$, we obtain $(L_R(s_j) - U_R(s_n)) \leq 0$ yielding a reward of at most 0.

Algorithm 4 implements Lemmas 3.16 and 3.17 and Theorem 3.18. We extend Algorithm 3 by adding Algorithm 4 after state selection (Algorithm 3 line 7). The final result is found in Algorithm 10 (page 184) in Appendix A.1.

3.6 Empirical Evaluation

In this section, we evaluate the automatic inference of monotonicity for pMCs as described in this chapter. We implemented the monotonicity checking in the probabilistic model checker Storm [HJKQ+22]. To obtain bounds for each state on the reachability probability and expected total reward we use parameter lifting (see also

Algorithm 4 Bounds extension (put after line 7 in Algorithm 3)).

1: **for all** $(s_1, s_2) \in U^2$ with $U = \text{succ}(s) \cup \{s\}$ s.t. $s_1 \not\preceq^{\text{rew}} s_2$ and $s_2 \not\preceq^{\text{rew}} s_1$ **do**
2: **if** $U_R(s_2) \leq L_R(s_1)$ **then**
3: extend $\preceq^{\text{rew}}$ with: $s_2 \preceq^{\text{rew}} s_1$ *Lemma 3.16*
4: **if** $\text{succ}(s) = \{s_1, s_2\}$ and $\forall \vec{u} \in R.\, r(s) \geq \mathbf{P}(s, s_1)(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ **then**
5: extend $\preceq^{\text{rew}}$ with: $s_2 \preceq^{\text{rew}} s$ *Lemma 3.17.1*
6: **else if** $\text{succ}(s) = \{s_1, s_2\}$ and $\forall \vec{u} \in R.\, r(s) \leq \mathbf{P}(s, s_1)(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ **then**
7: extend $\preceq^{\text{rew}}$ with: $s \preceq^{\text{rew}} s_2$ *Lemma 3.17.2*
8: **if** $\text{succ}(s) = \{s_1, \ldots, s_n\}$ and $n > 2$ **then**
9: **for all** $j \in \{1, \ldots, n\}$ **do**
10: **if** $\forall\, 1 \leq i < j.\, s_i \preceq^{\text{rew}} s_j$ and $r(s) \geq \mathbf{P}(s, s_j)(\vec{u}) \cdot \big(U_R(s_j) - L_R(s_1)\big)$ **then**
11: extend $\preceq^{\text{rew}}$ with: $s_j \preceq^{\text{rew}} s$ *Theorem 3.18*

Chapter 5) as already implemented in Storm. We took *all* benchmarks sets with pMCs and non-trivial reachability properties and expected reward properties as described in Appendix B.2.1. The model characteristics of the pMCs are found in Table B.3 (page 193) and Table B.4 (page 194). The benchmark set consists of simple pMCs (brp, crowds, gambler, message authentication, and zeroconf) and non-simple ones obtained from POMDPs (see also Appendix C (page 199)). We use the setup as described in Appendix B.1 and use the graph-preserving region $[0.1, 0.9]$. To evaluate our approach, we answer the following question:

Q3.1 How does automatically inferring monotonicity with help of the order compare to checking the solution function?

Q3.2 How does automatically inferring monotonicity with help of the order perform on different properties?

Q3.3 How do model characteristics influence the performance of automatically inferring monotonicity with help of the order?

Q3.4 How do state assumptions influence the performance of automatically inferring monotonicity with help of the order?

Non-monotonicity | The benchmark sets with reachability properties for egl, craps, nand and Herman andt he benchmark sets with expected total reward properties for 4x4-grid are not monotonic. Their non-monotonicity can be shown by uniformly taking 100 samples on the parameter space.

Table 3.1: Automatically inferring monotonicity in pMCs

	benchmark	instance	time in seconds: bounds check	mon. check	sol. func.	#mon. in $\lvert V\rvert$: mon. check	sol. func.	asm. used
Reachability Probabilities	brp	(2,16)	<1	<1	<1	2	2	no
		(15,4096)	<1	6	MO	2	-	
		(15,8192)	2	20	MO	2	-	
		(15,16384)	4	MO	MO	-	-	
	crowds	(3,5)	<1	<1	<1	2	2	no
		(6,5)	<1	1	<1	2	2	
	evade	(1,2,0,1)	<1	TO	TO	-	-	yes
	gambler	(6400)	2	2	TO	1	-	no
		(102400)	547	627	TO	1	-	
		(204800)	TO	TO	TO	-	-	
	maze	(25)	<1	<1	TO	24	-	no
		(1000)	<1	1	TO	999	-	
		(10000)	<1	700	TO	9999	-	
	mes. auth.	(30)	<1	<1	<1	2	2	no
		(15360)	<1	<1	<1	2	2	
	nrp	(5,1)	<1	<1	<1	1	5	no
		(10,1)	<1	<1	<1	1	10	
		(15,1)	<1	<1	<1	1	15	
	zeroconf	(6400)	<1	<1	1459	2	2	no
		(12800)	<1	3	TO	2	-	
		(409600)	4	2099	MO	2	-	
Expected Total Reward	egl	(5,2)	<1	<1	<1	9	10	yes
		(6,2)	<1	<1	<1	11	12	
		(7,2)	<1	<1	<1	13	14	
	4x4-grid	(1)	<1	TO	46	-	0	yes
		(2)	<1	TO	TO	-	-	
	maze	(1)	<1	62	TO	6	-	yes
		(2)	<1	TO	TO	-	-	
	network_dp	(2,1,1)	<1	TO	TO	-	-	yes
	network_ps	(2,1,1)	<1	<1	<1	14	14	no
		(2,5,1)	<1	TO	TO	-	-	yes
	samplerocks	(1,1)	<1	TO	TO	-	-	yes

3.6.1 Results

Table 3.1 shows the results for monotonicity checking for the considered benchmarks. It lists the benchmark and their instances. Furthermore, it lists whether it is acylic (A) or cyclic (C), the number of parameters $|\boldsymbol{V}|$, and the size of the pMC. **Model building** includes the time for construction, default preprocessing by Storm, and bisimulation minimisation. **Bounds check** shows the timings for obtaining bounds for the model. **Mon. check** shows timings for inferring monotonicity from the built model. To place these numbers in perspective, the column **sol. func.** shows the time to obtain monotonicity from the solution function from the built model by Storm (default settings, same preprocessing, based on the implementation in [DJJC+15]). Furthermore, we report the number of monotonic parameters found (#mon. in $|V|$) for both using the order and using the solution function. Finally, we report whether or not assumptions (**asm. used**) were needed to sort states in the order.

3.6.2 Discussion

Comparison to solution function | From Table 3.1, we observe that inferring monotonicity with the help of the order outperforms checking the solution function for brp, gambler, zeroconf, egl, and maze. The results for monotonicity checking with/without the solution function are similar for crowds, evade, message authentication, network_dp, network_ps, and samplerocks. The explanation is two-fold. First of all, for smaller models calculating the solution function is still viable. This occurs, e.g., for crowds after SCC-elimination and at message authentication after the default preprocessing. Secondly, for the larger models, calculating the solution function is not viable. However, using the order approach does not perform much better, as we need assumptions to build it, and larger models have a higher risk of needing more assumptions. This occurs, e.g., for evade and network_dp. For nrp, we see that monotonicity checking with the solution function outperforms monotonicity checking on the order. The timings are similar, however, the solution function approach can find all monotonic parameters. As some of the parameters are locally monotonic increasing and locally monotonic decreasing at different states, global monotonicity cannot be obtained (see also Theorem 3.1).

Answer to Q3.1 (How does automatically inferring monotonicity with help of the order compare to checking the solution function?)

Inferring monotonicity with the help of the order outperforms checking the solution function for models with a larger state space.

Reachability vs. expected total reward properties | If we compare the reachability properties to expected total reward properties, we observe from Table 3.1 that for reachability properties, we obtain better results. This can be explained as follows: for reachability properties there is only one state with a strictly positive reward, so all other states have reward 0. Therefore, we can order more states (see, e.g., Lemma 3.11). As we can order more states, less assumptions are needed for reachability properties. Thus the chances of getting an explosion in the number of orders (from which we infer monotonicity) are smaller.

Answer to Q3.2 (How does automatically inferring monotonicity with help of the order perform on different properties?)

Inferring monotonicity with the help of an order performs better for reachability properties than for expected total reward properties.

Model characteristics | Note that the model characteristics of the pMCs are found in Table B.1 (page 190) and Table B.2 (page 191). We discuss the influence of cyclicity of the model, and the influence of the number of parameters in the model. First of all, we compare acyclic (e.g., brp and nrp) and cyclic (e.g., crowds and evade) models. For these model types, we do not observe a difference in performance. This is can be explained by Lemma 3.11 and Lemma 3.12 being quite effective. Next, we discuss the influence of the number of parameters on the performance of Algorithm 10. We observe from Tables B.1 and B.2 that models with more parameters are less likely to become smaller during preprocessing, as, e.g., less bisimilar states are present. From Table 3.1, we observe that the performance at these models is (in general) slightly worse, which can be explained by a larger state space. However, having a small number of parameters is not a recipe for success, see, e.g., gambler and network_dp. Finally, the empirical evaluation shows that for simple pMCs (e.g., brp and zeroconf), we can find monotonicity for more instance than for non-simple pMCs. This can be explained by the way we build the order (Section 3.5). In the best case, we order a state between two of its successor states (see, e.g., Lemma 3.11). In simple pMCs, there are at most two successors, thus the order is more likely to be sufficient.

Answer to Q3.3 (How do model characteristics influence the performance of automatically inferring monotonicity with help of the order?)

- There is no performance difference between acyclic and cyclic models.
- The performance on models with less parameters is slightly better.
- Simple pMCs are easier to solve.

State assumptions | First of all, we observe from Table 3.1 that for both egl and maze, assumptions are used and monotonicity is found. This implies that Theorem 3.15 is applied and all assumptions are a witness for monotonicity. However, for other benchmarks (e.g., evade and network_dp) assumptions are used and no monotonicity is found due to an explosion in the number of orders.

Answer to Q3.4 (How do state assumptions influence the performance of automatically inferring monotonicity with help of the order?)

State assumptions may help finding monotonicity.

3.7 Conclusion

This chapter defined global and local monotonicity for pMCs. It showed a method to automatically infer monotonicity of pMCs for both reachability and expected total reward properties, based on a pre-order on the states. We evaluated the approach by running several benchmarks. First of all, the empirical evaluation showed that if monotonicity can be found by calculating the solution function, more monotonic parameters are found, as this method is complete. However, in the cases where the solution function has a memory-out, our approach often still works. Secondly, the evaluation shows that reachability properties are easier to analyze the expected total reward properties. Furthermore, it shows that there is no performance difference between acyclic and cyclic models, and the number of parameters only has a small influence on the performance. Finally, the evaluation shows that assumptions may help finding monotonicity, but only if a few assumptions are needed. Directions for future work include improvement of building the reward order, by, e.g., eliminating states which cannot be order, and finding monotonicity on the input modeling language.

4 Monotonicity for Markov Decision Processes

Summary | In the previous chapter, we have seen how we can obtain automatically infer monotonicity for a pMC with the help of a reward order. The next step is to focus on parametric Markov Decision Processes (pMDPs). First, we define both global and local monotonicity for pMDPs on the expected total reward solution function. As expected total reward properties for pMDPs have an optimization direction, we also introduce this for monotonicity. Moreover, we focus on monotonicity under minimization, monotonicity under maximization functions akin. Next, we extend the approach of Chapter 3 to pMDPs, by focusing on determining a (minimizing) scheduler. If such a scheduler can be determined, obtaining monotonicity for pMDPs is as hard as for pMCs. We explain how we can algorithmically compute such a scheduler on-the-fly while using the reward order. We reuse and extend the results of Chapter 3 to obtain monotonicity. Finally, we conclude our work by empirically evaluating the approach.

Origins | The chapter is an extension for pMDPs of the automatic inference of monotonicity for pMCs as described in Chapter 3 and [1; 5]. The main difference to Chapter 3 is the computation of the minimizing scheduler on-the-fly and the introduction of action assumptions.

Background | We consider pMDPs and reachability and expected total reward properties as introduced in Section 2.2. Furthermore, we consider monotonicity, as introduced in Section 2.1, and the automatic inference of monotonicity, as presented in Chapter 3.

Assumption 4.1 In this chapter, we describe monotonicity and reward orders w.r.t. the pMDP and its minimizing expected total reward solution function. Clearly, the definitions also transfer to the maximizing expected total reward solution function.

Assumption 4.2 In this chapter, we assume parameter $p \in V$, pMDP $\mathcal{M}$, and graph-preserving region R are given. Moreover, for conciseness, we drop superscript R in the definitions of global and local monotonicity, and we drop R and T

in the definition of the reward order. We only use them if they are not clear from the context, their usage is similar as in Chapter 3.

4.1 Related Work

Monotonicity in Markov chains | As pMCs are a special case of pMDPs, all work related to Chapter 3 as discussed in Section 3.1, also relates to this chapter.

Monotonicity in Markov decision processes | Pittenger [Pit88] shows a proof for concavity of the optimal expected total reward and monotonicity for optimal actions in an MDP in which the state space and action space are ordered, but the expected total reward function does not possess the normal properties used to establish monotonicity on MDPs. Similar to our work, Cleaveland et al. [CRSL22] define a partial order between states in MDPs. The "monotonic safety" Cleaveland et al. assume defines a partial order between states based on their safety probability. These assumptions are then used to remove non-determinism from the MDP. The assumptions do not hold in general, i.e., there may be cases where the partial order does not hold, whereas we have graph rules that ensure the reward order's validity.

4.2 Defining Monotonicity

4.2.1 Global monotonicity

global monotonicity

Definition 4.1 Scheduler $\sigma \in \mathit{Sched}(\mathcal{M})$ is *globally monotonic increasing*, written $\sigma{\uparrow_p}$, if $\mathcal{M}^{\sigma}{\uparrow_p}$.
We define $\sigma{\downarrow_p}$ analogously for *globally monotonic decreasing*, and $\sigma(\vec{u}){\uparrow_p}$ and $\sigma(\vec{u}){\downarrow_p}$ analogously for monotonicity of a dependent scheduler.

Remark 4.1 Although for dependent scheduler $\sigma(\vec{u})$, $\mathcal{M}^{\sigma(\vec{u})}$ is not properly defined, $\mathcal{M}^{\sigma(\vec{u})}{\uparrow_p}$ is defined as $\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{u})$ is defined (see Remark 2.7 and Definition 3.1).

Definition 4.2 The pMDP $\mathcal{M}$ is *globally monotonic increasing*, written $\mathcal{M}{\uparrow_p}$, if $\sigma{\uparrow_p}$ for all $\sigma \in \mathit{Sched}(\mathcal{M})$.
Globally monotonic decreasing, written $\mathcal{M}{\downarrow_p}$, is defined analogously.

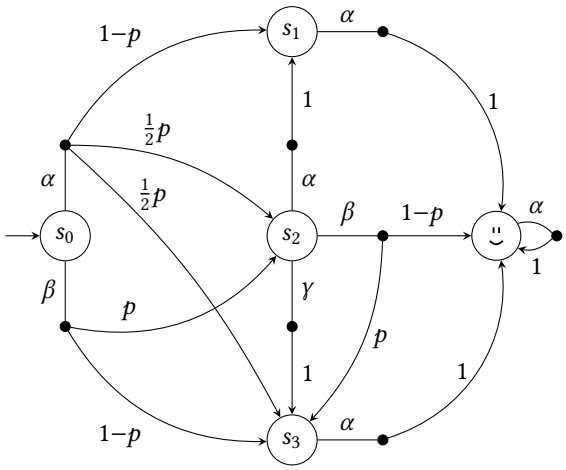


Figure 4.1: An example parametric Markov decision process $\mathcal{M}_1$ with $r(s_i) = i$

Example 4.1 Reconsider our running example pMDP $\mathcal{M}_1$, also depicted in Figure 4.1. Let the state rewards be given by: $r(s_i) = i$. Recall the schedulers of Example 2.27 and the corresponding expected total reward function:

	$\sigma(s_0)$	$\sigma(s_2)$	$\mathsf{ER}^{s_0 \to ☹}$
σ_1	α	α	$2p+1$
σ_2	α	β	$\frac{3}{2}p^2 + \frac{3}{2}p + 1$
σ_3	α	γ	$3p+1$
σ_4	β	α	3
σ_5	β	β	$3p^2 - p + 3$
σ_6	β	γ	$2p+3$

Consider region $R_1 = [0.8, 0.9]$. For all schedulers σ_i, we have $\sigma_i{\uparrow_p}$ on R_1 (see Example 2.24), thus we obtain $\mathcal{M}_1{\uparrow_p}$. For region $R_2 = [0.1, 0.9]$, however, σ_5 is not monotonic, so global monotonicity cannot be obtained.

Theorem 4.1 Let $\mathcal{M}{\uparrow_p}$ and $\sigma(\vec{u}) \in \mathit{Sched}(\mathcal{M}, \vec{u})$.

$$\text{If } \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{u}) \text{ is continuous on } R, \text{ then } \sigma(\vec{u}){\uparrow_p}\,.$$

Proof. Let $\vec{e}_p$ be the unit vector with a 1 at the position of parameter p. Following Definition 2.2, we have to show that for any $\vec{v} \in R$, and $b \in \mathbb{R}_{>0}$ such that $\vec{v'} = \vec{v} + b \cdot \vec{e}_p \in R$:

$$\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v}) \leq \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v'}) \,. \tag{4.1}$$

We distinguish two cases based on whether $\vec{v}$ and $\vec{v'}$ agree on a scheduler, i.e., $\vec{v}$ and $\vec{v'}$ agree iff $\sigma(\vec{v}) = \sigma(\vec{v'}) = \sigma \in \mathit{Sched}(\mathcal{M})$.
First of all, let $\vec{v}$ and $\vec{v'}$ agree on a scheduler, so $\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v}) = \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma}}(\vec{v})$ and $\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v'}) = \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma}}(\vec{v'})$. As $\mathcal{M}\uparrow_p$, we have $\sigma\uparrow_p$, thus

$$\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma}}(\vec{v}) \leq \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma}}(\vec{v'}) \,.$$

From this, we obtain:

$$\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v}) \leq \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v'}) \,.$$

Secondly, let $\vec{v}$ and $\vec{v'}$ disagree on a scheduler. Let $\vec{v}$ $\sigma_1 = \sigma(\vec{v})$ and $\sigma_2 = \sigma(v')$ such that $\sigma_1 \neq \sigma_2$. We show by contra-position that Equation (4.1) holds. Assume Equation (4.1) does not hold, so

$$\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma_1}}(\vec{v}) > \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma_2}}(\vec{v'}) \,. \tag{4.2}$$

By Definition 4.2 and $\mathcal{M}\uparrow_p$, we have $\sigma_1\uparrow_p$ and $\sigma_2\uparrow_p$. As $\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{u})$ is continuous, there exists $\sigma' \in \mathit{Sched}(\mathcal{M})$ such that not $\sigma'\uparrow_p$ and there exists $b_1, b_2 \in \mathbb{R}_{>0}$ with

- $b_1 < b_2 < b$,
- $\vec{w}_i = \vec{v} + b_i \cdot \vec{e}_p$ for $i \in \{1, 2\}$,
- $\sigma(w_1) = \sigma(w_2) = \sigma'$, and
- for all $b_1 < b' < b_2$ with $\vec{w'} = \vec{v} + b' \cdot \vec{e}_p$ we have $\sigma(\vec{w'}) = \sigma'$.

However, $\mathcal{M}\uparrow_p$ so by Definition 4.2 all independent schedulers are monotone increasing. This is a contradiction, therefore Equation (4.2) cannot hold, and we have $\sigma(\vec{u})\uparrow_p$ as

$$\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v}) \leq \mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{v'}) \,.$$

So for both cases we have $\sigma(\vec{u})\uparrow_p$. ∎

The following corollary follows directly from Theorem 4.1

Corollary 4.2 Let $\sigma(\vec{u}) \in \mathit{Sched}(\mathcal{M}, \vec{u})$ with $\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{u})$ continuous on R. Then:

$$\sigma\uparrow^R_p \text{ for all } \sigma \in \mathit{Dep}(\sigma(\vec{u})) \quad \text{implies} \quad \sigma(\vec{u})\uparrow^R_p .$$

In the following, we discuss monotonicity under minimization. For this, we use the minimizing scheduler σ^{min}. Note that this scheduler is not necessarily independent.

Definition 4.3 The pMDP $\mathcal{M}$ is *globally monotonic increasing under minimization*, written $\mathcal{M}\uparrow^{min}_p$, if $\sigma^{min}\uparrow_p$.
We define $\mathcal{M}\downarrow^{min}_p$ analogously for monotonic decreasing.

·under minimization

Example 4.2 We continue Example 4.1. For region R_1, σ^{min} is independent. We obtain $\mathcal{M}_1\uparrow^{min}_p$ directly from the induced pMCs $\mathcal{M}_1^{\sigma^{min}}$.
Recall from Example 2.27 that σ^{min} is dependent for region R_2. Following Remark 4.1, we check if $\mathsf{ER}^{s_0 \to}_{min}{}^{\cup}(p)$ is monotonic. Recall that

$$\mathsf{ER}^{s_0 \to}_{min}{}^{\cup}(p) = \begin{cases} 2p+1 & \text{if } p \geq \frac{1}{3} \\ \frac{3}{2}p^2 + \frac{3}{2}p + 1 & \text{otherwise} . \end{cases}$$

Even though $\mathsf{ER}^{s_0 \to}_{min}{}^{\cup}(p)$ is not continuously differentiable, it is monotonic increasing, thus for R_2 we also have $\mathcal{M}_1\uparrow^{min}_p$.

The following lemma states that if pMDP $\mathcal{M}$ is globally monotonic then $\mathcal{M}$ is also globally monotonic under minimization.

Lemma 4.3

$$\mathcal{M}\uparrow_p \quad \text{implies} \quad \mathcal{M}\uparrow^{min}_p .$$

Proof. In order to prove Lemma 4.3, we need to distinguish on the dependency of σ^{min}. If σ^{min} is independent, Lemma 4.3 directly follows from Definitions 4.2 and 4.3. If σ^{min} is dependent, Lemma 4.3 directly follows from Corollary 4.2. ■

Example 4.3 We continue from Example 4.1. Instead of obtaining $\mathcal{M}_1\uparrow^{min}_p$ by checking the induced pMC $\mathcal{M}_1^{\sigma^{min}}$ as in Example 4.2, we now obtain $\mathcal{M}_1\uparrow^{min}_p$ from Lemma 4.3 and $\mathcal{M}_1\uparrow_p$.

For region R_2, we cannot obtain $\mathcal{M}_1{\uparrow}_p^{min}$ from Lemma 4.3 as there is no global monotonicity, e.g., σ_5 is not monotonic on R_2.

4.2.2 Local monotonicity

First of all, we extend the notation for local monotonicity in pMCs. For pMC $\mathcal{D}$, we let $\mathcal{D}{\uparrow}_{p,s}^{\ell}$ denote $\mathsf{ER}_{\mathcal{D}}^{s\to T}(\vec{u}){\uparrow}_p^{\ell}$. We now lift the notion of local monotonicity for pMCs to local monotonicity for pMDPs.

local monotonicity

Definition 4.4 Scheduler $\sigma \in \mathit{Sched}(\mathcal{M})$ is *locally monotonic increasing at $s \in S$*, written $\sigma{\uparrow}_{p,s}^{\ell}$, if $\mathcal{M}^{\sigma}{\uparrow}_{p,s}^{\ell}$.
We define $\sigma{\downarrow}_{p,s}^{\ell}$ analogously for *locally monotonic decreasing*, and $\sigma(\vec{u}){\uparrow}_{p,s}^{\ell}$ and $\sigma(\vec{u}){\downarrow}_{p,s}^{\ell}$ analogously for local monotonicity of a dependent scheduler.

Remark 4.2 Although $\mathcal{M}^{\sigma(\vec{u})}$ is not properly defined, $\mathcal{M}^{\sigma(\vec{u})}{\uparrow}_{p,s}^{\ell}$ is (see also Remark 4.1).

Definition 4.5 The pMDP $\mathcal{M}$ is *locally monotonic increasing at $s \in S$*, denoted $\mathcal{M}{\uparrow}_{p,s}^{\ell}$, if $\sigma{\uparrow}_{p,s}^{\ell}$ for all $\sigma \in \mathit{Sched}(\mathcal{M})$.
Locally monotonic decreasing at $s \in S$, denoted $\mathcal{M}{\downarrow}_{p,s}^{\ell}$, is defined analogously.

Example 4.4 We continue from Example 4.1. Recall from Definition 3.2 that $\mathcal{D}{\uparrow}_{p,s}^{\ell}$, if:

$$\left(\sum_{s' \in \mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right) \cdot \mathsf{ER}^{s'\to T}\right)(\vec{u}) \;\geq\; 0 \quad \text{for all } \vec{u} \in R\,.$$

For any (possibly dependent) scheduler σ of $\mathcal{M}_1$ we have $\mathsf{ER}_{\mathcal{M}_1^{\sigma}}^{s_1\to}{}^{\cup}_{\cap}(p) = 1$ and $\mathsf{ER}_{\mathcal{M}_1^{\sigma}}^{s_3\to}{}^{\cup}_{\cap}(p) = 3$. For s_1 and s_3, we trivially have local monotonicity, independent of the scheduler. Thus, we obtain $\mathcal{M}_1{\uparrow}_{p,s_1}^{\ell}$ and $\mathcal{M}_1{\uparrow}_{p,s_3}^{\ell}$.
We now consider state s_2, for which we obtain the following expected total reward functions for the different independent schedulers on $\mathcal{M}_1$:

	$\mathsf{ER}^{s_2 \to \breve{\cup}}$
σ_1, σ_4	$2 + \mathsf{ER}^{s_1 \to \breve{\cup}}(p)$
σ_2, σ_5	$2 + p \cdot \mathsf{ER}^{s_3 \to \breve{\cup}}(p) + (1-p) \cdot \mathsf{ER}^{\breve{\cup} \to \breve{\cup}}(p)$
σ_3, σ_6	$2 + \mathsf{ER}^{s_3 \to \breve{\cup}}(p)$

Recall that for any $\sigma \in \mathit{Sched}(\mathcal{M}_1)$, graph-preserving region R, and $p \in R$ we have $\mathsf{ER}^{s_3 \to \breve{\cup}}(p) \geq \mathsf{ER}^{\breve{\cup} \to \breve{\cup}}(p)$. From the above, we conclude that for any $\sigma \in \mathit{Sched}(\mathcal{M}_1)$: $\sigma \uparrow^{\ell}_{p,s_2}$. From Definition 4.5, we obtain $\mathcal{M}_1 \uparrow^{\ell}_{p,s_2}$.
We now consider state s_0 and region R_1. For all $\sigma \in \mathit{Sched}(\mathcal{M}_1)$, we obtain $\sigma \uparrow^{\ell}_{p,s_0}$, thus, $\mathcal{M}_1 \uparrow^{\ell}_{p,s_0}$.
Now consider state s_0 and region R_2. As we cannot obtain local monotonicity for σ_5, local monotonicity at s_0 for $\mathcal{M}_1$ does not exist on R_2.

Theorem 4.4 Let $\mathcal{M} \uparrow^{\ell}_{p,s}$ and $\sigma(\vec{u}) \in \mathit{Sched}(\mathcal{M}, \vec{u})$. Then"

$$\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}(\vec{u}) \text{ is continuous on } R \quad \text{implies} \quad \sigma(\vec{u}) \uparrow^{\ell}_{p,s} .$$

Proof. The proof of Theorem 4.4 is analogous to the proof of Theorem 4.1. ■

Corollary 4.5 Let $\sigma(\vec{u}) \in \mathit{Sched}(\mathcal{M}, \vec{u})$ with $\mathsf{ER}^{s_I \to T}_{\mathcal{M}^{\sigma(\vec{u})}}$ continuous on R, and $s \in S$

$$\text{if } \sigma \uparrow^{\ell}_{p,s} \text{ for all } \sigma \in \mathit{Dep}(\sigma(\vec{u})), \text{ then} \quad \sigma(\vec{u}) \uparrow^{\ell}_{p,s} .$$

As for global monotonicity, we now consider local monotonicity under minimization.

Definition 4.6 The pMDP $\mathcal{M}$ is *locally monotonic increasing under minimization* at $s \in S$, denoted $\mathcal{M} \uparrow^{\ell,min}_{p,s}$, if $\sigma^{min} \uparrow^{\ell}_{p,s}$
We define $\mathcal{M} \downarrow^{\ell,min}_{p,s}$ analogously for monotone decreasing.

·under minimization

Example 4.5 We continue with Example 4.4. For region R_1, σ^{min} is independent. For any $s \in S$, we have $\mathcal{M}_1 \uparrow^{\ell}_{p,s}$, so for any independent scheduler $\sigma \in \mathit{Sched}(\mathcal{M}_1)$ we obtain $\sigma \uparrow^{\ell}_{p,s}$. Therefore, $\sigma^{min} \uparrow^{\ell}_{p,s}$ and we get $\sigma_1 \uparrow^{\ell}_{p,s}$.
For region R_2, σ^{min} is dependent. $\mathit{Dep}(\sigma^{min}) = \{\sigma_1, \sigma_2\}$. For any $s \in S$, we have $\sigma_1 \uparrow^{\ell}_{p,s}$ and $\sigma_2 \uparrow^{\ell}_{p,s}$. From Corollary 4.5, we now obtain $\sigma^{min} \uparrow^{\ell}_{p,s}$.

The following proposition states that if pMDP $\mathcal{M}$ is locally monotonic then $\mathcal{M}$ is

also locally monotonic under minimization.

Proposition 4.6

$$\mathcal{M}\uparrow^{\ell}_{p,s} \quad \text{implies} \quad \mathcal{M}\uparrow^{\ell,min}_{p,s} .$$

As for pMCs, local monotonicity implies global monotonicity.

Lemma 4.7

$$\left(\forall s \in S.\, \mathcal{M}\uparrow^{\ell}_{p,s}\right) \quad \text{implies} \quad \mathcal{M}\uparrow_{p} \tag{4.3}$$

$$\left(\forall s \in S.\, \mathcal{M}\uparrow^{\ell,min}_{p,s}\right) \quad \text{implies} \quad \mathcal{M}\uparrow^{min}_{p} . \tag{4.4}$$

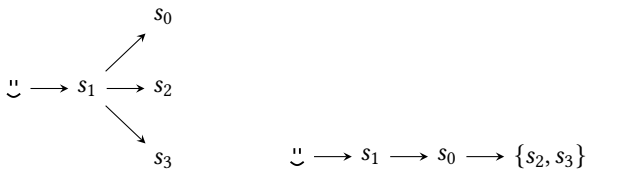
Proof. We show the proof of Equation (4.3), the proof of Equation (4.4) is obtained in a similar manner.

$$\begin{aligned}
&\forall s \in S.\, \mathcal{M}\uparrow^{\ell}_{p,s} \\
&\overset{\text{Def. 4.5}}{\iff} \forall s \in S.\, \forall \sigma \in Sched(\mathcal{M}).\, \sigma\uparrow^{\ell}_{p,s} \\
&\overset{\text{Def. 4.4}}{\iff} \forall s \in S.\, \forall \sigma \in Sched(\mathcal{M}).\, \mathcal{M}^{\sigma}\uparrow^{\ell}_{p,s} \\
&\iff \forall \sigma \in Sched(\mathcal{M}).\, \forall s \in S.\, \mathcal{M}^{\sigma}\uparrow^{\ell}_{p,s} \\
&\overset{\text{Thm. 3.1}}{\iff} \forall \sigma \in Sched(\mathcal{M}).\, \mathcal{M}^{\sigma}\uparrow_{p} \\
&\overset{\text{Def. 4.2}}{\iff} \mathcal{M}\uparrow_{p}
\end{aligned}$$

Example 4.6 We continue from Example 4.5. For region R_1, we have $\mathcal{M}_1\uparrow^{\ell}_{p,s}$ for all $s \in S$, therefore, we obtain for R_1: $\mathcal{M}_1\uparrow_p$. Furthermore, for both region R_1 and R_2 we have $\mathcal{M}_1\uparrow^{\ell,min}_{p,s}$ for all $s \in S$, therefore, we obtain for both regions: $\mathcal{M}_1\uparrow^{min}_{p}$.

4.3 Obtaining a Reward Order

As for pMCs, we can find monotonicity of a pMDP with the help of an ETR formula (see also Section 4.4), however, the scheduler choices make this even less viable, as the number of scheduler choices is (also) worst-case exponential in the number of states.



(a) Reward order (b) Reward order under minimimization

Figure 4.2: Examples of orders for the pMDP in Figure 4.1

Therefore, we again focus on constructing a pre-order on the states of the pMDP, the reward order.

Definition 4.7 An ordering relation $\preceq^{\mathsf{rew}} \subseteq S \times S$ is a *reward order* if for all $\sigma \in \mathit{Sched}(M)$, and $s, t \in S$:

reward order

$$s \preceq^{\mathsf{rew}} t \quad \text{implies} \quad \mathsf{ER}^{s \to T}_{\mathcal{M}^\sigma}(\vec{u}) \leq \mathsf{ER}^{t \to T}_{\mathcal{M}^\sigma}(\vec{u}) \text{ for all } \vec{u} \in R\,.$$

Definition 4.8 An ordering relation $\preceq^{\mathsf{rew}}_{min} \subseteq S \times S$ is a *reward order under minimization* if for all $s, t \in S$:

·under minimization

$$s \preceq^{\mathsf{rew}}_{min} t \quad \text{implies} \quad \mathsf{ER}^{s \to T}_{\mathcal{M}^{\sigma^{min}}}(\vec{u}) \leq \mathsf{ER}^{t \to T}_{\mathcal{M}^{\sigma^{min}}}(\vec{u}) \text{ for all } \vec{u} \in R\,.$$

As for pMCs, reward orders can be *exhaustive* and sufficient, see Definitions 3.4 and 3.5.

·exhaustive/ sufficient

Example 4.7 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1. Recall from Example 4.1, that there are six different schedulers. For this region, we thus have:

$\mathsf{ER}^{s \to \lightning}$

s	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
s_0	$[2.6, 2.8]$	$[4.248, 4.942]$	$[3.4, 3.7]$	3	$[4.12, 4.53]$	$[4.6, 4.8]$
s_1	1	1	1	1	1	1
s_2	3	$[4.4, 4.7]$	5	3	$[4.4, 4.7]$	5
s_3	3	3	3	3	3	3
$\lightning$	0	0	0	0	0	0

Since we do not apply a single instantiation $\vec{u}$ to our function, but the region $R_1 = [0.8, 0.9]$, intervals occur at s_0 and s_2 (see also Example 3.4).
Figure 4.2a shows a reward order for this pMDP and region. States s_0, s_2, and s_3

are not ordered, as for different schedulers the order differs. Although this order is exhaustive, it is not sufficient for state s_0, as s_2 and s_3 are not sorted.
Figure 4.2b shows the reward order under minimization for this pMDP and region R_1. Recall from Example 2.27, that the minimizing scheduler for this region is σ_1. The depicted order is exhaustive and sufficient.

The following proposition states that if two states are ordered by the reward order, they are also ordered under minimization.

Proposition 4.8 For pMDP $\mathcal{M}$ with $s, t \in S$:

$$s \preceq^{\mathsf{rew}} t \quad \text{implies} \quad s \preceq^{\mathsf{rew}}_{min} t \,.$$

Example 4.8 Reconsider the reward order of Figure 4.2a as discussed in Example 4.7. All states that are sorted in this reward order are also sorted in the reward order under minimization, as depicted in Figure 4.2b

Reward order and monotonicity under minimization | In the remainder of this section, we focus on local monotonicity under minimization and the reward order under minimization. Of course, we can adapt the introduced propositions, lemmas, and corollaries to the more general case without minimization. We assume a (possibly dependent) scheduler σ^{min} is provided.

The following proposition and lemma directly follow from lifting their duals for pMCs, as found in Section 3.3, to local monotonicity under minimization. Both assume that the minimizing scheduler is independent for state s on region R. Recall that we leave out R from the notation for conciseness.

Proposition 4.9 Let $s \in S$, $\mathrm{succ}(s) = \{s_1, s_2\}$ and $\sigma^{min}(s) = \alpha$, with $\mathbf{P}(s, \alpha, s_1) = p$ and $\mathbf{P}(s, \alpha, s_2) = 1{-}p$. Then:

$$s_2 \preceq^{\mathsf{rew}}_{min} s_1 \quad \text{implies} \quad \mathcal{M}{\uparrow}^{\ell, min}_{p,s}.$$

Lemma 4.10 Let $\mathrm{succ}(s, \sigma^{min}(s)) = \{s_1, \ldots, s_n\}$, $\sigma^{min}(s) = \alpha$, $\mathbf{P}(s, \alpha, s_i) = f_i$, and $\forall j > i.\, s_j \preceq^{\mathsf{rew}}_{min} s_i$. Then:

$$\exists i \in [1, \ldots, n].\, \Big(\forall j \leq i.\, f_j(\vec{u}){\uparrow}_p \text{ and } \forall j > i.\, f_j(\vec{u}){\downarrow}_p\Big) \quad \text{implies} \quad \mathcal{M}{\uparrow}^{\ell, min}_{p,s}.$$

Example 4.9 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1 with region $R_1 = [0.8, 0.9]$ and the reward order under minimization as depicted in Figure 4.2b. Recall that $\sigma^{min}(s_0) = \alpha$. As $s_2 \equiv^{\mathsf{rew}} s_3$ and $s_1 \prec^{\mathsf{rew}} s_2$, we obtain from Lemma 4.10 $\mathcal{M}_1{\uparrow}^{\ell,min}_{p,s_0}$.

We lift the above lemma to a dependent scheduler.

Lemma 4.11 Let $\mathrm{succ}(s, \sigma^{min}) = \{s_1, \ldots, s_n\}$, $\sigma_i \in \mathit{Dep}(\sigma^{min})$, $\mathbf{P}(s, \sigma_i(s), s_j) = f_{i,j}$, and $\forall k > j.\, s_k \preceq^{\mathsf{rew}}_{min} s_j$. Then:

$$\text{for all } \sigma_i \in \mathit{Dep}(\sigma^{min})$$

$$\exists j \in [1, \ldots, n].\, \Big(\forall k \leq j.\, f_{i,k}(\vec{u}){\uparrow}_p \text{ and } \forall k > j.\, f_{i,k}(\vec{u}){\downarrow}_p\Big) \tag{4.5}$$

$$\text{implies} \quad \mathcal{M}{\uparrow}^{\ell,min}_{p,s}\,.$$

Proof. The proof follows from Lemma 3.6 and Theorem 4.1. ∎

Example 4.10 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1 with region $R_2 = [0.1, 0.9]$. Recall that $\sigma^{min}(s_2) = \begin{cases} \alpha & \text{if } p \geq \frac{1}{3} \\ \beta & \text{otherwise} \end{cases}$ and $\lightning \preceq^{\mathsf{rew}}_{min} s_3$.
For $\sigma \in \mathit{Dep}(\sigma^{min})$ with $\sigma(s_2) = \alpha$, we observe that for all $\sigma_i \in \mathit{Dep}(\sigma(\vec{u}))$ Equation (4.5) trivially holds, as there is only one successor state.
For $\sigma \in \mathit{Dep}(\sigma^{min})$ with $\sigma(s_2) = \beta$, we observe that Equation (4.5) holds as $\lightning \preceq^{\mathsf{rew}}_{min} s_3$. By Lemma 4.11 we obtain for R_2: $\mathcal{M}{\uparrow}^{\ell,min}_{p,s}$.

Corollary 4.12 Let pMDP $\mathcal{M}$ be such that for each $s \in S$ with $\sigma^{min}(s) = \alpha$, $|\mathrm{succ}(s, \alpha)| \leq 2$ and $\mathbf{P}(s, s')$ is monotonic for each $s' \in \mathrm{succ}(s)$. Then: $\preceq^{\mathsf{rew}}_{min}$ is sufficient for s implies $\mathsf{ER}^{s \to T}(\vec{u})$ is locally monotonic increasing/decreasing under minimization on any region R in all parameters.

4.4 Complexity

In this section, we discuss the complexity of checking whether two states are ordered by the expected total reward order, and the decision problem whether a pMDP is monotonic in the general case (considering all schedulers) and under minimization. We show that these problems are in co-ETR.

4.4.1 General case

Recall the problems as introduced in Section 2.4. As in Section 3.4, we restrict ourselves in the remainder of this section to $\bowtie := < (\bowtie := \geq)$ and $Q := \exists\ (Q := \forall)$ for pMC-F (pMC-V), yielding the following problems:

- **(pMDP-F)** *Given a pMDP $\mathcal{M}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does there exist a $\vec{u} \in R$ such that $\exists \sigma \in Sched(\mathcal{M}).\ \mathsf{ER}^{s \to T}_{\mathcal{M}^\sigma}(\vec{u}) < \lambda$?*
- **(pMDP-V)** *Given a pMDP $\mathcal{M}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does $\mathsf{ER}^{s \to T}_{\mathcal{M}^\sigma}(\vec{u}) \geq \lambda$ hold for all $\vec{u} \in R$ and $\sigma \in Sched(\mathcal{M})$?*

We describe the decision problem whether two states are ordered by the expected total reward (pMDP-RO) and the decision problem whether a pMDP is monotonic (pMDP-Mon) as follows:

- **(pMDP-Mon)** *Given a pMDP $\mathcal{M}$, a parameter $p \in V$, and a graph-preserving region R, does $\mathcal{M}\uparrow^R_p$ hold?*
- **(pMDP-RO)** *Given a pMDP $\mathcal{M}$, a graph-preserving region R, and two states s_1 and s_2, does $s_1 \preceq^{\mathsf{rew}}_{R,T} s_2$ hold, where $\preceq^{\mathsf{rew}}_{R,T}$ denotes the exhaustive reward order?*

Observe from Definitions 4.2 and 4.7 that these two problems consider all $\sigma \in Sched(\mathcal{M})$ (like pMDP-V).

PMDP-V is in co-ETR | [JKPW21] provide an encoding for pMDP-F for reachability properties. Adapting this for expected total reward is easy, as shown in Example 4.11. As pMDP-V is the complement of pMDP-F, we obtain that pMDP-V is in co-ETR.

Example 4.11 Reconsider our running example, with reward function $r(s_i) = i$, target state $\lightning$, and region $R = [0.8, 0.9]$. To encode the pMDP-F problem with $\lambda = 3.2$ as an ETR problem, we use variables $\{x_{s,\sigma} \mid s \in S, \sigma \in Sched(\mathcal{M})\} \cup V$ where $x_{i,\sigma}$ encodes the expected total reward from state s_i under scheduler σ and V are the parameters of the pMDP. We now encode the pMDP by taking the conjunction of the encodings of the induced pMCs for the independent schedulers (see also Example 3.8). So, we encode $\mathcal{M}_1$ by the conjunction of

$$
\begin{aligned}
x_{\lightning,\sigma} &= 0 \\
x_{0,\sigma} &= 0 + (1-p) \cdot x_{1,\sigma} + \frac{1}{2}p \cdot x_{2,\sigma} + \frac{1}{2}p \cdot x_{3,\sigma} \\
x_{1,\sigma} &= 1 + 1 \cdot x_{\lightning,\sigma} \\
x_{2,\sigma} &= 2 + 1 \cdot x_{1,\sigma} \\
x_{3,\sigma} &= 3 + 1 \cdot x_{\lightning,\sigma}
\end{aligned}
$$

for $\sigma \in \mathit{Sched}(\mathcal{M}_1)$. As for pMCs, we encode the region by:

$$0.8 \leq p \leq 0.9.$$

Finally, we encode the bound for the pMC-F problem by the disjunction of

$$x_{0,\sigma} \leq 3.2$$

for $\sigma \in \mathit{Sched}(\mathcal{M}_1)$. If checking for satisfiability yields a solution, we found a solution to the pMC-F. If it yields no solution, we have found the solution to co-ETR problem pMC-V.

PMDP-RO is at least as hard as pMDP-V | As for pMCs, we observe that determining whether two states are ordered by the exhaustive reward order is as hard as pMDP verification.

Theorem 4.13 The pMDP-V decision problem is polynomial-time reducible to pMDP-RO, the decision problem whether two states are ordered by the exhaustive reward order.

Proof. The proof is analogous to the proof of Theorem 3.8, now applied to pMDPs.■

Example 4.12 Reconsider our running example. Consider pMDP-RO to check $s_1 \prec^{\mathsf{rew}} s_3$. We encode the pMDP and the region as in Example 4.11. We encode the check for $s_1 \prec^{\mathsf{rew}} s_3$ by the conjunction for all $\sigma \in \mathit{Sched}(\mathcal{M}_1)$ of

$$x_{1,\sigma} \geq x_{3,\sigma} \,.$$

If checking for satisfiability yields no solution, $s_1 \prec^{\mathsf{rew}} s_3$ follows.

PMDP-Mon is at least as hard as pMDP-RO (and pMDP-V) | As for pMCs, we find for pMDPs that checking for monotonicity is as hard as determining whether two states are ordered by the exhaustive reward order.

Lemma 4.14 The pMDP-RO decision problem is polynomial-time reducible to pMDP-Mon, the decision problem whether a pMDP is monotonic.

Proof. The proof is analogous to the proof of Lemma 3.9, now applied to pMDPs.■

Corollary 4.15 The pMDP-V decision problem is polynomial-time reducible to pMDP-Mon, the decision problem whether a pMC is monotonic.

Corollary 4.15 is an immediate corollary to Theorem 4.13 and Lemma 4.14. Deciding whether a pMDP is monotonic can also be expressed as a co-ETR problem.

Example 4.13 Reconsider our running example. Assume we have the pMDP-Mon problem and want to check $\mathcal{M}_1{\uparrow_p}$. We encode the pMDP and the region as in Example 4.11. We let $d_{i,\sigma}$ encode the derivative of the expected total reward for state s_i given scheduler $\sigma \in \mathit{Sched}(\mathcal{M}_1)$. The encoding is analogous to the encoding of the derivative of a pMC in Example 3.10, e.g., for $\sigma = \sigma_1$ we have:

$$
\begin{aligned}
d_{\lightning,\sigma_1} &= 0 \\
d_{0,\sigma_1} &= (1-p)\cdot d_{1,\sigma_1} - x_{1,\sigma_1} + \frac{1}{2}p \cdot d_{2,\sigma_1} + \frac{1}{2}\cdot x_{2,\sigma_1} + \frac{1}{2}p \cdot d_{3,\sigma_1} + \frac{1}{2}\cdot x_{3,\sigma_1} \\
d_{1,\sigma_1} &= d_{\lightning,\sigma_1} \\
d_{2,\sigma_1} &= d_{1,\sigma_1} \\
d_{3,\sigma_1} &= d_{\lightning,\sigma_1}
\end{aligned}
$$

We encode the monotonicity check as the conjunction for all $\sigma \in \mathit{Sched}(\mathcal{M}_1)$ of

$$d_{0,\sigma} < 0\ .$$

If checking for satisfiability yields no solution, $\mathcal{M}_1{\uparrow_p}$ follows.

4.4.2 Complexity under minimization

We describe the decision problem whether two states are ordered by the expected total reward (pMDP-RO-Min) and the decision problem whether a pMDP is monotonic (pMDP-Mon-Min) as follows:

- **(pMDP-Mon-Min)** *Given a pMDP $\mathcal{M}$, a parameter $p \in V$, and a graph-preserving region R, does $\mathcal{M}{\uparrow_p^{R,min}}$ hold?*
- **(pMDP-RO-Min)** *Given a pMDP $\mathcal{M}$, a graph-preserving region R, and two states s_1 and s_2, does $s_1 \preceq^{\mathsf{rew}}_{min,R,T} s_2$ hold, where $\preceq^{\mathsf{rew}}_{min,R,T}$ denotes the exhaustive reward order under minimization?*

Note that by Definitions 4.3 and 4.8 respectively, these two problems consider the minimizing scheduler σ^{min}.

Analogous to Section 4.4.1, we find the following:

Theorem 4.16 The pMDP-V-Min decision problem is polynomial-time reducible to pMDP-RO-Min, the decision problem whether two states are ordered under minimization by the exhaustive reward order.

Lemma 4.17 The pMDP-RO-Min decision problem is polynomial-time reducible to pMDP-Mon-Min, the decision problem whether a pMDP is monotonic under minimization.

Corollary 4.18 The pMDP-V-Min decision problem is polynomial-time reducible to pMDP-Mon-Min, the decision problem whether a pMC is monotonic under minimization.

4.5 Automatically Proving Monotonicity

As for pMCs (Section 3.5), we aim to construct a sufficient reward order for a pMDP under minimization. First, we show that we could analyze all possible schedulers and obtain monotonicity from this. We then argue why this is not viable and suggest an approach where we compute a minimizing scheduler *on-the-fly*.

For the reward order $\preceq^{\mathsf{rew}}$ for pMDPs under all possible schedulers, we use the same notation as for pMCs (page 49), i.e., we define $\prec^{\mathsf{rew}}$, $\equiv^{\mathsf{rew}}$, $[s]^{\mathsf{rew}}$, reward orders for sets of states, $\mathsf{lb}(X)$, and $\mathsf{ub}(X)$. We introduce equivalent notation for the reward order under minimization and leave out subscripts R and T for conciseness. Let $\prec^{\mathsf{rew}}_{min}$ denote the irreflexive variant of $\preceq^{\mathsf{rew}}_{min}$ and $s \equiv^{\mathsf{rew}}_{min} s'$ denote $\mathsf{ER}^{s\to T}_{min}(\vec{u}) = \mathsf{ER}^{s'\to T}_{min}(\vec{u})$ for all $\vec{u} \in R$, i.e., $s \equiv^{\mathsf{rew}}_{min} s'$ iff $s \preceq^{\mathsf{rew}}_{min,R,T} s'$ and $s' \preceq^{\mathsf{rew}}_{min} s$. We let $s \prec^{\mathsf{rew}}_{min} X$ ($X \prec^{\mathsf{rew}}_{min} s$) for $X \subseteq S$ be shorthand notation for $s \prec^{\mathsf{rew}}_{min} s'$ ($s' \prec^{\mathsf{rew}}_{min} s$) for all $s' \in X$. Equivalently, we define $s \equiv^{\mathsf{rew}}_{min} X$, $X \equiv^{\mathsf{rew}}_{min} s$, $s \preceq^{\mathsf{rew}}_{min} X$, and $X \preceq^{\mathsf{rew}}_{min} s$. Let $[s]^{\mathsf{rew}}_{min}$ denote the set of all $s' \in S$ such that $s \equiv^{\mathsf{rew}}_{min} s'$. For $X \subseteq S$, $\mathsf{ub}^{\min}(X) = \{s \in S \mid X \preceq^{\mathsf{rew}}_{min} s\}$ and $\mathsf{lb}^{\min}(X) = \{s \in S \mid s \preceq^{\mathsf{rew}}_{min} X\}$ denote the upward and downward closure of X w.r.t. reward order under minimization $\preceq^{\mathsf{rew}}_{min}$. Furthermore, let $\min^{\min}(X) = \{x \in X \mid \nexists x' \in X. x' \preceq^{\mathsf{rew}}_{min} x\}$, and $\max^{\min}(X) = \{x \in X \mid \nexists x' \in X. x \preceq^{\mathsf{rew}}_{min} x'\}$.

4.5.1 Checking all schedulers

Recall from Proposition 4.8 that if the reward order orders two states, then they are also ordered by the reward order under minimization.

Lemma 4.19 If for all $\sigma \in \mathit{Sched}(\mathcal{M})$ all orders computed by applying Algorithm 10 (page 184) to $\mathcal{M}^\sigma$ agree on $s \preceq^{\mathsf{rew}} t$, then $s \preceq^{\mathsf{rew}}_{min} t$.

Proof. Lemma 4.19 directly follows from Proposition 4.8. ∎

Example 4.14 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1 with $r(s_i) = i$. Recall that $\mathsf{ER}^{\circlearrowright\to\circlearrowright}(p) = 0$, $\mathsf{ER}^{s_1\to\circlearrowright}(p) = 1$, and $\mathsf{ER}^{s_3\to\circlearrowright}(p) = 3$. For any graph-preserving region R and $\sigma \in \mathit{Sched}(\mathcal{M}_1)$, we have $\mathsf{ER}^{\circlearrowright\to\circlearrowright}(p) \leq \mathsf{ER}^{s_1\to\circlearrowright}(p) \leq \mathsf{ER}^{3\to\circlearrowright}(p) \leq \mathsf{ER}^{s_2\to\circlearrowright}(p)$. So for all schedulers the orders agree on $\circlearrowright \preceq^{\mathsf{rew}} s_1 \preceq^{\mathsf{rew}} s_3 \preceq^{\mathsf{rew}} s_2$. We thus have $\circlearrowright \preceq^{\mathsf{rew}}_{min} s_1 \preceq^{\mathsf{rew}}_{min} s_3 \preceq^{\mathsf{rew}}_{min} s_2$.

The corollary below directly follows from Theorem 3.15, Definition 4.2, and Lemma 4.3. It states that if we obtain monotonicity for all schedulers by applying Algorithm 10 to the induced pMC, we obtain monotonicity for the minimizing scheduler.

Corollary 4.20 If for all $\sigma \in \mathit{Sched}(\mathcal{M})$ all orders computed by applying Algorithm 10 (page 184) on $\mathcal{M}^\sigma$ are a witness for $\mathcal{M}^\sigma{\uparrow_p}$, then $\mathcal{M}{\uparrow_p^{min}}$.

Example 4.15 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1 with $r(s_i) = i$. Recall the six different schedulers σ_i from Example 4.1. First, consider region $R_1 = [0.8, 0.9]$. For all σ_i we obtain $\mathcal{M}_1^{\sigma_i}{\uparrow_p}$ when applying Algorithm 10 to $\mathcal{M}_1^{\sigma_i}$. Thus from Corollary 4.20, we obtain $\mathcal{M}_1{\uparrow_p^{R,min}}$. Now consider region $R_2 = [0.1, 0.9]$. For, e.g., σ_5 we cannot order states s_2 and s_3. Thus Algorithm 10 uses assumptions to order these states. The computed orders for σ_5 do not agree on the monotonicity. Thus we cannot obtain $\mathcal{M}_1^{\sigma_5}{\uparrow_p}$ for R_2. Therefore, we cannot obtain monotonicity for $\mathcal{M}_1$ on R_2.

Although this approach is sound, it is not viable in practice. From Section 3.6, we know that assumptions lead to memory-outs. If we consider all schedulers, thus several induced pMC, we need to analyze each of the pMCs separately, thus the chances of assumptions being required increase, which will in its turn yield to more memory-outs. Even if we do not require assumptions, we still have to analyze all possible schedulers of the pMDP. The number of schedulers is (worst-case) exponential in the number of states. The main challenge of this chapter is thus to determine (a part of) the minimizing scheduler, so we do not have to go over all possible schedulers.

4.5.2 Fixing a scheduler

The main goal is to compute (a part of) the minimizing scheduler on-the-fly. To do so, we use the reward order we obtained so far. For state s, we call action $\alpha \in Act(s)$ minimizing for s if $\sigma^{min}(s) = \alpha$. Furthermore, for $\alpha_1, \alpha_2 \in Act(s)$, we call α_1 *preferred over* α_2 *for* s, if for all $\sigma_i \in Sched(\mathcal{M})$ with $\sigma_i(s) = \alpha_i$ for $i \in \{1,2\}$ we have $\mathsf{ER}^{s\to T}_{\mathcal{M}^{\sigma_1}}(\vec{u}) \leq \mathsf{ER}^{s\to T}_{\mathcal{M}^{\sigma_2}}(\vec{u})$ for all $\vec{u} \in R$. We call α_1 preferred over α_2 for s w.r.t. $\vec{u} \in R$, if for all $\sigma_i \in Sched(\mathcal{M})$ with $\sigma_i(s) = \alpha_i$ for $i \in \{1,2\}$ we have $\mathsf{ER}^{s\to T}_{\mathcal{M}^{\sigma_1}}(\vec{u}) \leq \mathsf{ER}^{s\to T}_{\mathcal{M}^{\sigma_2}}(\vec{u})$.

·minimizing
·preferred

Corollary 4.21 Let $s \in S$ and $\alpha \in Act(s)$ Then:

$$\text{for each } \alpha' \in Act(s) \setminus \{\alpha\} : \ \alpha \text{ is preferred over } \alpha' \text{ for } s$$
$$\text{implies} \quad \alpha \text{ is minimizing for } s\,.$$

Example 4.16 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1 with $r(s_i) = i$. Let R be any graph-preserving region. Consider state s_2, for any scheduler $\sigma_\alpha, \sigma_\beta, \sigma_\gamma \in Sched(\mathcal{M}_1)$ with $\sigma_\alpha(s) = \alpha$, we have $\mathsf{ER}^{s_2\to \ddot{\smile}}_{\mathcal{M}_1^{\sigma_\alpha}}(p) \leq \mathsf{ER}^{s_2\to \ddot{\smile}}_{\mathcal{M}_1^{\sigma_\beta}}(p)$ and $\mathsf{ER}^{s_2\to \ddot{\smile}}_{\mathcal{M}_1^{\sigma_\alpha}}(p) \leq \mathsf{ER}^{s_2\to \ddot{\smile}}_{\mathcal{M}_1^{\sigma_\gamma}}(p)$ for any $p \in R$. We thus obtain that action α is preferred over both β and γ. Thus from Corollary 4.21, we obtain that α is minimizing for s_2.

We call set of actions $Act^{min}_R = \{\alpha \in Act(s) \mid \exists \vec{u} \in R.\, \sigma^{min}(s,\vec{u}) = \alpha\}$ the *minimizing action set*. Observe that we can divide region R into j sub-regions R_j, such that we have a minimizing action for each R_j. The minimizing action set for R is now the union of all minimizing actions for each R_j. Moreover, we can express Act^{min}_R as the set of all actions which are preferred for some $\vec{u} \in R$, i.e., $Act^{min}_R(s) = \{\alpha_1 \in Act(s) \mid \exists \vec{u} \in R. \forall \alpha_2 \in Act(s) \setminus \{\alpha_1\}.\, \alpha_1 \text{ is preferred over } \alpha_2 \text{ for s w.r.t. }', \vec{u}\}$.

·minimizing action set

In the remainder of this section, we first of all focus on finding a minimizing action. If this succeeds, we can fix the scheduler to this action. Secondly, if we cannot find a minimizing action, we focus on finding the minimizing action set. Finally, we introduce action assumptions.

Finding a minimizing action | We observe that we can check if an action α_1 is preferred over α_2 at state s by solving a non-linear program. We let $R(p) = [\mathsf{l}_p, \mathsf{u}_p]$ denote the lower and upper bound of p in region R and let x_i denote the expected total reward from state s_i. Let $\alpha_1, \alpha_2 \in Act(s)$ and $\mathrm{succ}(s) = \{s_1, \ldots, s_n\}$ with $s_i \preceq^{\mathsf{rew}} s_j$ for $i < j$. Now α_1 is preferred over α_2 for s if the following non-linear program (NLP) is not satisfiable:

4

$$0 \leq x_1 < \infty \wedge \ldots \wedge 0 < x_n \leq \infty \tag{4.6}$$

$$x_1 \leq x_2 \wedge x_2 \leq x_3 \wedge \ldots \wedge x_{n-1} \leq x_n \tag{4.7}$$

$$\mathsf{l}_p \leq p \leq \mathsf{u}_p \wedge \mathsf{l}_q \leq q \leq \mathsf{u}_q \wedge \ldots \tag{4.8}$$

$$\sum_{s_i \in \mathrm{succ}(s,\alpha_1)} \mathbf{P}(s,\alpha_1,s_i) \cdot x_i > \sum_{s_i \in \mathrm{succ}(s,\alpha_2)} \mathbf{P}(s,\alpha_2,s_i) \cdot x_i \tag{4.9}$$

Equation (4.6) represents the bounds for the expected reward, which is known to be non-negative and finite. Equation (4.7) describes the ordering of the states obtained from the reward order and Equation (4.8) encodes the bounds for the parameters in R. Finally, Equation (4.9) checks if α_1 is *not* preferred over α_2. If checking for satisfiability yields no solution, it follows α_1 is preferred over α_2. Otherwise, α_1 might still be preferred over α_2, as the obtained counterexample might be spurious since we do not know the exact bounds for the states. If we extend the NLP to all possible actions, we can check if an independent minimizing scheduler (see Corollary 4.21) exists. If this is the case, we can apply the rules from Section 3.5. This reasoning suggests Algorithm 5, where at line 9 we try to find a minimizing action with the help of a NLP. Algorithm 10 is found in Appendix A.1 on page 184.

Example 4.17 Reconsider our running example pMDP $\mathcal{M}_1$ from Figure 4.1 with $r(s_i) = i$ on region $R_1 = [0.1, 0.9]$. Consider state s_2, to check if α is preferred over β, we check if the following is not satisfiable:

$$0 \leq x_{\breve{\cup}} < \infty \wedge 0 \leq x_1 < \infty \wedge 0 < x_3 \leq \infty$$
$$x_{\breve{\cup}} \leq x_1 \wedge x_1 \leq x_3$$
$$0.1 \leq p \leq 0.9$$
$$1 \cdot x_1 > (1-p) \cdot x_{\breve{\cup}} + p \cdot x_3$$

Unfortunately, this is satisfiable, e.g., $x_{\breve{\cup}} = 0$, $x_1 = x_3 = 1$, $p = 0.1$ is a satisfying instantiation. Note that this is a spurious counterexample, see also Example 4.18.

Remark 4.3 Note that we do not consider bounds from parameter lifting. We can extend the NLP to work with bounds. Also, for readability, we assume the states to be totally ordered. If states are not ordered, we can still solve the NLP; however, the chances of getting a spurious counterexample increase.

Algorithm 5 Construction of a reward order under minimization for a pMDP

Input: pMDP $\mathcal{M} = (S, s_I, \{\lightning\}, Act, V, \mathbf{P})$ and bounds L_R, U_R
s.t. $L_R(s) \leq \mathsf{ER}^{s \to \lightning}(\vec{u}) \leq U_R(s)$ for all $\vec{u} \in R$ and $s \in S$
Output: Result = set of pairs of annotated order and scheduler $\left(\preceq^{\mathsf{rew},\mathcal{A}}, \sigma\right)$

```
1: Result ← ∅, Queue ← (S' : S \ {↯}, A : ∅, ≺^rew: {↯} × S', σ(s) : ⊥)
2: while Queue not empty do
3:    S', A, ≤^rew, σ ← Queue.pop()
4:    if S' = ∅ then
5:       Result ← Result ∪ {(≤^{rew,A}, σ)}
6:    else
7:       select s ∈ S' with s topologically last w.r.t. ↯
8:       Algorithm 10 lines 8-10
9:       if best action for s is α then
10:         σ(s) ← α
11:         Algorithm 10 lines 11-18 on succ(s, α)
12:         if ≤^rew is a total order for succ(s, α) then
13:            Algorithm 10 lines 20-26
14:            Queue.push(S' \ {s}, A, ≤^rew, σ)
15:         else if succ(s, α) = {s1, s2} and
                  ≤^rew is not a total order for {s1, s2} and
                  (s1 ≡^rew s or s ≺^rew s1 or (s1 ≺^rew s and r(s) = 0)) then
16:            Algorithm 10 lines 29-37
17:            Queue.push(S' \ {s}, A, ≤^rew, σ)
18:         else
19:            Algorithm 10 lines 40-43
20: return Result
```

Example 4.18 We continue from Example 4.17 and extend the NLP with bounds, so we check for satisfiability of:

$$
\begin{aligned}
& x_{\lightning} = 0 \wedge x_1 = 1 \wedge x_3 = 3 \\
& x_{\lightning} \leq x_1 \wedge x_1 \leq x_3 \\
& 0.1 \leq p \leq 0.9 \\
& 1 \cdot x_1 > (1-p) \cdot x_{\lightning} + p \cdot x_3
\end{aligned}
$$

As no satisfying instantiation is found, we obtain that action α is preferred over β. This agrees with the result from Example 4.16 that state α is minimizing for state s_2.

Using a minimizing action set | If a minimizing scheduler does not exist, we try to find a minimizing action set for each state to reduce the number of possible actions. Let $\mathrm{succ}(s) = \{s_1, \ldots, s_n\}$ with $s_i \preceq^{\mathsf{rew}} s_j$ for $i < j$. We check if $\alpha_1 \in Act_R^{min}(s)$ by checking the following NLP with $Act(s) = \{\alpha_1, \ldots, \alpha_n\}$:

$$
\begin{aligned}
&0 \leq x_1 < \infty \wedge \ldots \wedge 0 \leq x_n < \infty \\
&x_1 \leq x_2 \wedge x_2 \leq x_3 \wedge \ldots \wedge x_{n-1} \leq x_n \\
&\mathsf{l}_p \leq p \leq \mathsf{u}_p \wedge \mathsf{l}_q \leq q \leq \mathsf{u}_q \wedge \ldots \\
&\quad \sum_{s_i \in \mathrm{succ}(s,\alpha_2)} \mathbf{P}(s,\alpha_2,s_i) \cdot x_i > \sum_{s_i \in \mathrm{succ}(s,\alpha_1)} \mathbf{P}(s,\alpha_1,s_i) \cdot x_i \vee \ldots \\
&\quad \vee \sum_{s_i \in \mathrm{succ}(s,\alpha_n)} \mathbf{P}(s,\alpha_n,s_i) \cdot x_i > \sum_{s_i \in \mathrm{succ}(s,\alpha_1)} \mathbf{P}(s,\alpha_1,s_i) \cdot x_i
\end{aligned} \tag{4.10}
$$

The structure is akin to the structure for finding a minimizing action. However, Equation (4.9) is replaced by the disjunction of Equation (4.10). If the NLP is unsatisfiable, this means that for all $\vec{u} \in R$ there exists an $\alpha \in Act(s) \setminus \{\alpha_1\}$ such that α is preferred over α_1. Thus, α_1 is not in $Act_R^{min}(s)$. As before, if solving the NLP provides us with a counterexample this might be spurious, thus $Act'(s)$ is a superset of $Act_R^{min}(s)$.

The following proposition states that if for all actions in the set computed by the NLP described above we have local monotonicity at s, then we have local monotonicity under minimization at s.

Proposition 4.22 Let $Act'(s) \supseteq Act_R^{min}(s)$. Then:

$$\mathcal{M}^{\sigma}\uparrow_{p,s}^{\ell,min} \text{ for all } \sigma \in Sched(\mathcal{M}) \text{ with } \sigma(s) \in Act'(s) \quad \text{implies} \quad \mathcal{M}\uparrow_{p,s}^{\ell,min}.$$

Proof. As $Act'(s) \supseteq Act_R^{min}(s)$, the proof directly follows from Definition 4.6 and Corollary 4.5. ∎

Using graph rules given a minimizing action set | As in Lemma 3.11, we sort s w.r.t. its successor states in a backwards manner.

Lemma 4.23 For $s \in S \setminus T$, let $Act'(s) \supseteq Act_R^{min}(s)$, $S_l = \mathrm{lb}^{\min}(\mathrm{succ}(s, Act'(s)))$ and $S_u = \mathrm{ub}^{\min}(\mathrm{succ}(s, Act'(s)))$. Then:

1. if $r(s) > 0$, then $S_l \prec_{min}^{\mathsf{rew}} s$,
2. if $r(s) = 0$ and $S_u \neq \emptyset$, *then*
   either $\mathrm{succ}(s) \subseteq [s]_{min}^{\mathsf{rew}}$ or $S_l \prec_{min}^{\mathsf{rew}} s \prec_{min}^{\mathsf{rew}} S_u$.

Proof. We show the proof for Lemma 4.23.1. Assume $s_1 \in S_l$. By definition of the downward closure under minimization, we have for all $s_2 \in \mathrm{succ}(s, Act'(s)) \setminus \{s_1\}$: $s_1 \preceq^{\mathsf{rew}}_{min} s_2$. Let $r(s) > 0$. As $Act'(s) \supseteq Act^{min}_R(s)$ we have:

$$\begin{aligned}
\mathsf{ER}^{s\to T}_{min}(\vec{u}) &= r(s) + \min_{\alpha \in Act'(s)} \left\{ \left(\sum_{s' \in \mathrm{succ}(s,\alpha)} \mathbf{P}(s,\alpha,s') \cdot \mathsf{ER}^{s'\to T}_{min} \right)(\vec{u}) \right\} \\
&\geq r(s) + \min_{\alpha \in Act'(s)} \left\{ \left(\sum_{s' \in \mathrm{succ}(s,\alpha)} \mathbf{P}(s,\alpha,s') \cdot \mathsf{ER}^{s_1\to T}_{min} \right)(\vec{u}) \right\} \\
&= r(s) + \mathsf{ER}^{s_1\to T}_{min}(\vec{u}) \\
&> \mathsf{ER}^{s_1\to T}_{min}(\vec{u})
\end{aligned}$$

From this, Lemma 4.23.1 directly follows. Lemma 4.23.2 is obtained by a similar reasoning. ∎

Lemma 4.23 states that if the reward obtained for our current state s is strictly greater than 0, s is ordered above the successor state with the lowest expected total reward. If the reward obtained at s is 0, and the upward closure of the successor states of s is non-empty, then either s lies in the order at the successor states or between the downward and upward closure of the successors. Note that the successor state can be any of the successors reachable via the actions in $Act'(s)$.

Example 4.19 Reconsider our running example on region $R_2 = [0.1, 0.9]$. Let $Act'(s_2) = \{\alpha, \beta\}$ and assume $\lightning \prec^{\mathsf{rew}}_{min} s_1 \prec^{\mathsf{rew}}_{min} s_2$. From Example 2.27, we know

$$\sigma^{min}(s_2) = \begin{cases} \alpha & \text{if } p \geq \frac{1}{3} \\ \beta & \text{otherwise}, \end{cases}$$

thus $Act'(s_2) \supseteq Act^{min}_{R_2}(s_2)$. From Lemma 4.23, we now obtain $\lightning \prec^{\mathsf{rew}}_{min} s_2$.

Following Lemma 3.12, we obtain for a cyclic pMDP:

Lemma 4.24 For any state $s \in S \setminus T$ with $Act'(s) \supseteq Act^{min}_R(s)$ and for all $\alpha \in Act'(s)$ with $\mathrm{succ}(s,\alpha) = \{s_1, s_2\}$, the following holds:

1. if $s_1 \equiv^{\mathsf{rew}}_{min} s$ and $r(s) = 0$, then $s_2 \equiv^{\mathsf{rew}}_{min} s$,
2. if $s_1 \equiv^{\mathsf{rew}}_{min} s$ and $r(s) > 0$, then $s_2 \prec^{\mathsf{rew}}_{min} s$,

3. if $s \prec^{\mathsf{rew}}_{min} s_1$, then $s_2 \prec^{\mathsf{rew}}_{min} s$, and
4. if $s_1 \prec^{\mathsf{rew}}_{min} s$ and $r(s) = 0$, then $s \prec^{\mathsf{rew}}_{min} s_2$.

Proof. We show the proof for Lemma 4.24.1. Let $Act'(s) \supseteq Act^{min}_R(s)$ and for all $\alpha \in Act'(s)$ with $\mathrm{succ}(s,\alpha) = \{s_1, s_2\}$. Assume $s_1 \equiv^{\mathsf{rew}}_{min} s$ and $r(s) = 0$. First of all, observe that for all $\vec{u} \in R$ we have:

$$\begin{aligned}\mathsf{ER}^{s\to T}_{min}(\vec{u}) &= r(s) + \min_{\alpha\in Act'(s)} \left\{ \left(\mathbf{P}(s,\alpha,s_1) \cdot \mathsf{ER}^{s_1\to T}_{min} + \mathbf{P}(s,\alpha,s_2) \cdot \mathsf{ER}^{s_2\to T}_{min} \right)(\vec{u}) \right\} \\ &= \min_{\alpha\in Act'(s)} \left\{ \left(\mathbf{P}(s,\alpha,s_1) \cdot \mathsf{ER}^{s\to T}_{min} + (1-\mathbf{P}(s,\alpha,s_1)) \cdot \mathsf{ER}^{s_2\to T}_{min} \right)(\vec{u}) \right\}\end{aligned}$$

From this, we obtain:

$$\begin{aligned}0 &= \min_{\alpha\in Act'(s)} \left\{ \left(\mathbf{P}(s,\alpha,s_1) \cdot \mathsf{ER}^{s\to T}_{min} + (1-\mathbf{P}(s,\alpha,s_1)) \cdot \mathsf{ER}^{s_2\to T}_{min} \right)(\vec{u}) \right\} - \mathsf{ER}^{s\to T}_{min}(\vec{u}) \\ &= \min_{\alpha\in Act'(s)} \left\{ \left(\mathbf{P}(s,\alpha,s_1) \cdot \mathsf{ER}^{s\to T}_{min} + (1-\mathbf{P}(s,\alpha,s_1)) \cdot \mathsf{ER}^{s_2\to T}_{min} - \mathsf{ER}^{s\to T}_{min} \right)(\vec{u}) \right\} \\ &= \min_{\alpha\in Act'(s)} \left\{ \left((1-\mathbf{P}(s,\alpha,s_1)) \cdot \mathsf{ER}^{s_2\to T}_{min} - (1-\mathbf{P}(s,\alpha,s_1)) \cdot \mathsf{ER}^{s\to T}_{min} \right)(\vec{u}) \right\} \\ &= \min_{\alpha\in Act'(s)} \left\{ \left(\mathsf{ER}^{s_2\to T}_{min} - \mathsf{ER}^{s\to T}_{min} \right)(\vec{u}) \right\} \\ &= \mathsf{ER}^{s_2\to T}_{min}(\vec{u}) - \mathsf{ER}^{s\to T}_{min}(\vec{u})\end{aligned}$$

From this, Lemma 4.24.1 directly follows. Lemma 4.24.2, Lemma 4.24.3, and Lemma 4.24.4 are obtained by a similar reasoning. ∎

To deal with minimizing action sets, we extend Algorithm 5 by Algorithm 6, which implements Lemmas 4.23 and 4.24.

Making action assumptions under minimization | Using a superset on the minimizing action set seems viable if the ordering of the successors is known and the successors for the different actions overlap, i.e., $\mathrm{succ}(s,\alpha) \cap \mathrm{succ}(s,\beta) \neq \emptyset$. However, this might not always be the case. The following example motivates the use of action assumptions to obtain a sufficient reward order.

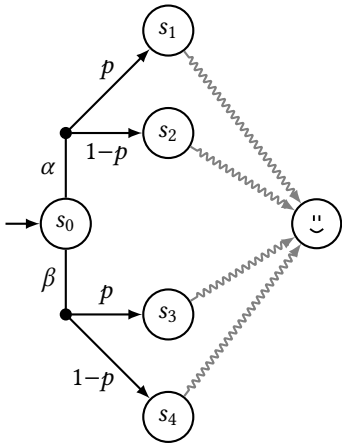
Example 4.20 Consider the pMDP in Figure 4.3a, with $r(s_i) = 0$. Assume we somehow know $s_1 \prec^{\mathsf{rew}} s_2$ and $s_4 \prec^{\mathsf{rew}} s_3$. As we do not know how s_1 and s_2 are ordered w.r.t. s_3 and s_4, we cannot find a minimizing action and therefore cannot

Algorithm 6 Action set extension for $Act'(s)$ (put after line 19 in Algorithm 5)

1: **else if** $r(s) > 0$ and $\mathsf{lb}(\mathsf{succ}(s, Act'(s))) = S_I$ **then**
2: extend $\preceq^{\mathsf{rew}}$ with: $S_I \preceq^{\mathsf{rew}} s$ *Lemma 4.23.1*
3: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}}, \sigma)$
4: **else if** $r(s) = 0$ and $\max(\mathsf{lb}(\mathsf{succ}(s, Act'(s)))) = s_l$ and $\min(\mathsf{ub}(\mathsf{succ}(s, Act'(s)))) = s_u$ **then**
5: **if** $s_l \equiv^{\mathsf{rew}} s_u$ **then**
6: extend $\preceq^{\mathsf{rew}}$ with: $s \equiv^{\mathsf{rew}} s_u$ *Lemma 4.23.2*
7: **else**
8: extend $\preceq^{\mathsf{rew}}$ with: $s_l \prec^{\mathsf{rew}} s$ and $s \prec^{\mathsf{rew}} s_u$ *Lemma 4.23.2*
9: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}}, \sigma)$
10: **else if** $\mathsf{succ}(s, \alpha) = \{s_1, s_2\}$ for all $\alpha \in Act'(s)$ and $\preceq^{\mathsf{rew}}$ is not a total order for $\{s_1, s_2\}$ and $\big(s_1 \equiv^{\mathsf{rew}} s$ or $s \prec^{\mathsf{rew}} s_1$ or $(s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0)\big)$ **then**
11: **if** $s_1 \equiv^{\mathsf{rew}} s$ **then**
12: **if** $r(s) = 0$ **then**
13: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \equiv^{\mathsf{rew}} s$ *Lemma 4.24.1*
14: **else**
15: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$ *Lemma 4.24.2*
16: **else if** $s \prec^{\mathsf{rew}} s_1$ **then**
17: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$ *Lemma 4.24.3*
18: **else if** $s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0$ **then**
19: extend $\preceq^{\mathsf{rew}}$ with: $s \prec^{\mathsf{rew}} s$ *Lemma 4.24.4*
20: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}}, \sigma)$

order s_0. However, any region R can be partitioned into three (potentially empty) sub-regions: A region with α as minimizing action, a region with β as minimizing action, and a region with α and β both being minimizing actions. Note that for the latter case, we pick, w.l.o.g., α. We create two copies of the order, each containing one of the first two action assumptions. Figure 4.3b shows the two copies assuming α and β minimizing, respectively.

We introduce a new type of assumptions, the *action assumptions*, denoted by $\mathcal{A}^{Act}$. Let $\mathcal{A}^{Act} \subseteq S \times Act$ be a set of action assumptions, such that for all $s \in S$ we have at most one action selected, i.e., $|(s, \cdot) \in \mathcal{A}^{Act}| \leq 1$, and for all $(s, \alpha) \in \mathcal{A}^{Act}$ we have $\alpha \in Act(s)$.



(a) Example pMDP with $s_1 \prec^{\mathsf{rew}} s_2$ and $s_4 \prec^{\mathsf{rew}} s_3$

Assume α is minimizing:

$s_2 \longrightarrow s_0 \longrightarrow s_1$, $s_3 \longrightarrow s_4$, :)

Assume β is minimizing:

$s_2 \longrightarrow s_1$, $s_3 \longrightarrow s_0 \longrightarrow s_4$, :)

(b) Reward orders with assumptions

Figure 4.3: Illustrating the use of action assumptions

·action assumption

Definition 4.9 Let $Sched(\mathcal{M})^{\mathcal{A}^{Act}}$ be the set of schedulers with *action assumptions* $\mathcal{A}^{Act}$, i.e., $Sched(\mathcal{M})^{\mathcal{A}^{Act}} = \{\sigma \in Sched(\mathcal{M}) \mid \forall (s,\alpha) \in \mathcal{A}^{Act}.\, \sigma(s) = \alpha\}$. The set of dependent schedulers with assumptions, denoted $Sched(\mathcal{M}, \vec{u})^{\mathcal{A}^{Act}}$, is defined analogously.

·valid under minimization

Lemma 4.25 Given set of action assumptions $\mathcal{A}^{Act}$. If for all $\vec{u} \in \mathbb{R}$

$$\min\left\{\mathsf{ER}^{s\to T}_{\mathcal{M}^\sigma}(u) \mid \sigma \in Sched(\mathcal{M})\right\} = \min\left\{\mathsf{ER}^{s\to T}_{\mathcal{M}^\sigma}(u) \mid \sigma \in Sched(\mathcal{M})^{\mathcal{A}^{Act}}\right\}$$

then we call $\mathcal{A}^{Act}$ *(globally) valid under minimization.*

We now adapt Algorithm 5 extended by Algorithm 6 to keep track of a scheduler and deal with scheduler assumptions. The result is given in Algorithm 7. As in Algorithm 3, we annotate the resulting order with the state assumptions (Algorithm 7 line 5). Furthermore, we annotate the scheduler with the action assumptions. Note that at Algorithm 7 lines 11-12 we make action assumptions. This part of the algorithm is only reached if no minimizing action is found, and for the minimizing action set the states are not totally ordered. The number of orders introduced by the action assumptions is worst-case exponential in the number of states. As in Section 3.5.3, we want to discharge assumptions. The following theorem is analogous to Theorem 3.15 and states

Algorithm 7 Construction of a Reward Order for a pMDP

Input: pMDP $\mathcal{M} = (S, s_I, \{\cup\}, Act, V, \mathbf{P})$ and bounds L_R, U_R
s.t. $L_R(s) \leq \mathsf{ER}^{s \to \cup}(\vec{u}) \leq U_R(s)$ for all $\vec{u} \in R$ and $s \in S$
Output: Result = set of pairs of annotated order and scheduler $\left(\preceq^{\mathsf{rew},\mathcal{A}}, \sigma^{\mathcal{A}^{Act}}\right)$

1: Result $\leftarrow \emptyset$, Queue $\leftarrow \left(S' : S \setminus \{\cup\}, \mathcal{A} : \emptyset, \mathcal{A}^{Act} : \emptyset, \prec^{\mathsf{rew}}: \{\cup\} \times S', \sigma(s) : \bot\right)$
2: **while** Queue not empty **do**
3: $S', \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma \leftarrow$ Queue.pop()
4: **if** $S' = \emptyset$ **then**
5: Result $\leftarrow$ Result $\cup \{(\preceq^{\mathsf{rew},\mathcal{A}}, \sigma^{\mathcal{A}^{Act}})\}$
6: **else**
7: determine $Act'(s)$ based on NLP and σ
8: Algorithm 5 lines 7-19 with Queue.push($S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma$)
9: Algorithm 6 lines 1-20 with Queue.push($S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma$)
10: **else**
11: **for all** $\alpha \in Act'(s)$ **do**
12: Queue.push($S', \mathcal{A}, \mathcal{A}^{Act} \cup \{\alpha\}, \preceq^{\mathsf{rew}}, \sigma(s) \leftarrow \alpha$)
13: **return** Result

that assumptions can sometimes be ignored.

Theorem 4.26 If all orders computed by Algorithm 7 are witnesses for a parameter to be monotonic increasing (decreasing), then the parameter is indeed monotonic increasing (decreasing).

Unfortunately, this theorem only allows us to discharge orders after computations.

As the minimizing scheduler might be dependent, it is hard to discharge all actions, however, we can limit the number of action assumptions by only considering the actions found when searching for $Act'(s) \supseteq Act_R^{min}$. The final result is found in Algorithm 11 in Appendix A.2 (page 182).

4.6 Empirical Evaluation

In this section, we evaluate our monotonicity checking approach for pMDPs as described in this chapter. We implemented the automatic inference of monotonicity for pMDPs in the probabilistic model checker Storm [HJKQ+22]. To check for bounds for monotonicity we use parameter lifting (see also Chapter 5) as already implemented in Storm. We use the setup as described in Appendix B.1 and use the graph-preserving region $[0.1, 0.9]$. We took the benchmarks sets with pMDPs and non-trivial reachability

properties and expected reward properties as described in Appendix B.2.2. For all benchmarks, we tested for non-monotonicity by uniformly taking 100 samples on the parameter space for the provided property. All benchmarks have possibly monotonic parameters. The model characteristics of the pMDPs are found in Table B.3 (page 193), Table B.4 (page 194), and Table B.5 (page 195) in Appendix B.2.2.

To evaluate our approach, we answer the following question:

Q4.1 How does automatically inferring monotonicity with help of the order perform on different properties?

Q4.2 How do (action and state) assumptions influence the performance of automatically inferring monotonicity with help of the order?

4.6.1 Results

Tables 4.1 and 4.2 show the results for monotonicity checking for the considered benchmarks for reachability and expected total reward properties, respectively. It lists the benchmark and their instances. Furthermore, it shows the timings for obtaining bounds for the model (**Bounds check**) and for inferring monotonicity from the built model (**Mon. check**). Additionally, we report the result for the number of parameters ($|V|$), i.e., the number of monotonic parameters found (**#mon.**), the number of constant parameters (**#const.**), and the number of parameters for which monotonicity is unknown (**#unk.**). Note that the reported number of constant parameters also includes parameters eliminated by model simplification. Finally, we report whether or not assumptions (**asm. used**) were needed to sort states in the order.

In Table 4.1, we leave out the results for AODV, as all benchmarks time-out. Furthermore, for brp (pPTA), we only show the results for the S2 instances; results for S1 are similar. For the honest instances of repudiation, we only consider the non-strict (ns) instances; the results for the strict (s) instances are equivalent.

> **Remark 4.4** Contrary to the results for pMCs we do not compare to the **sol. func.** as this would require to calculate the solution function for all schedulers.

4.6.2 Discussion

In general, we observe that checking for bounds for the states is done in less than a second for all benchmarks. So this does not yield much overhead.

Reachability vs. expected total reward properties | Recall from Section 3.6 that we see better results for reachability properties when comparing the reachability properties to expected total reward properties for pMCs. This is caused by only having

Table 4.1: Automatically inferring monotonicity for reachability properties in pMDPs

benchmark	instance	time in seconds: bounds check	time in seconds: mon. check	$\|V\|$: #mon.	$\|V\|$: #const.	$\|V\|$: #unk.	asm. used
consensus	(2,1,min)	<1	TO	-	-	-	yes
	(2,2,min)	<1	MO	-	-	-	
csma	(2,2,min)	<1	<1	0	0	2	yes
	(2,6,min)	<1	<1	0	0	2	
	(3,2,min)	<1	TO	-	-	-	
	(2,2,min2)	<1	<1	0	0	2	
	(2,6,min2)	<1	<1	0	0	2	
	(3,2,min2)	<1	207	0	0	3	
	(3,4,min2)	<1	TO	-	-	-	
	(2,2,max)	<1	<1	0	0	2	
	(2,6,max)	<1	<1	0	0	2	
	(3,2,max)	<1	TO	-	-	-	
zeroconf	(20,2,true,min)	<1	2	2	0	0	yes
	(20,8,true,min)	<1	3239	0	0	2	
	(20,2,false,min)	<1	MO	-	-	-	
	(20,2,true,max)	<1	<1	2	0	0	
	(20,8,true,max)	<1	42	2	0	0	
	(20,2,false,max)	<1	MO	-	-	-	
brp (pPTA)	(S2,br,D)	<1	MO	-	-	-	yes
	(S2,br,1)	<1	TO	-	-	-	
	(S2,br,2)	<1	TO	-	-	-	
	(S2,br,3)	<1	TO	-	-	-	
	(S2,br,4)	<1	<1	1	1	0	no
	(S2,dc,D)	<1	MO	-	-	-	yes
	(S2,dc,1)	<1	<1	2	0	0	no
	(S2,dc,2)	<1	<1	0	0	2	
	(S2,dc,3)	<1	<1	0	0	2	
	(S2,dc,4)	<1	<1	1	1	0	
firewire (pPTA)	(br,400)	<1	<1	1	0	0	no
	(dc,400)	<1	<1	1	0	0	
	(dc,600)	<1	TO	-	-	-	yes
nrp (pPTA)	(honest,ns,br,100)	<1	<1	1	0	0	no
	(honest,ns,dc,20)	<1	TO	-	-	-	yes
	(malicious,ns,br,5)	<1	155	1	0	2	
	(malicious,ns,br,10)	<1	TO	-	-	-	
	(malicious,ns,br,event)	<1	TO	-	-	-	
	(malicious,ns,dc,5)	<1	9	1	0	2	
	(malicious,ns,dc,10)	<1	TO	-	-	-	
	(malicious,ns,dc,event)	<1	1370	0	0	3	
zeroconf (pPTA)	(br,100)	<1	<1	2	0	0	no
	(br,150)	<1	<1	2	0	0	yes
	(br,200)	<1	TO	-	-	-	
	(br,wrong)	<1	<1	2	0	0	no
	(dc,100)	<1	<1	2	0	0	
	(dc,150)	<1	TO	-	-	-	yes
	(dc,200)	<1	TO	-	-	-	
	(dc,wrong)	<1	1	2	0	0	no

one state with a strictly positive reward. For pMDPs, there is not much of a difference between the properties. This can be explained as follows: for pMDPs, we have an additional source of assumptions, the action assumptions. This makes the advantage of having only one state with a strictly positive reward insignificant.

Table 4.2: Automatically inferring monotonicity for expected total reward properties in pMDPs

benchmark	instance	time in seconds bounds check	time in seconds mon. check	$\lvert V \rvert$ #mon.	$\lvert V \rvert$ #const.	$\lvert V \rvert$ #unk.	asm. used
consensus	(2,1,min)	<1	TO	-	-	-	yes
	(2,1,max)	<1	MO	-	-	-	
csma	(2,2,min)	<1	<1	0	0	2	yes
	(2,4,min)	<1	<1	0	0	2	
	(2,6,min)	<1	<1	0	0	2	
	(3,2,min)	<1	TO	-	-	-	
	(2,2,max)	<1	<1	0	0	2	
	(2,4,max)	<1	<1	0	0	2	
	(2,6,max)	<1	<1	0	0	2	
	(3,2,max)	<1	TO	-	-	-	
eajs	(2,30,max)	<1	<1	0	5	0	no
	(2,35,max)	<1	323	1	0	4	yes
	(2,40,max)	<1	TO	-	-	-	
	(3,30,max)	<1	1	2	5	0	
	(3,35,max)	<1	TO	-	-	-	
	(4,30,max)	<1	<1	3	6	0	
	(4,35,max)	<1	TO	-	-	-	
	(5,30,max)	<1	<1	4	7	0	
	(5,35,max)	<1	TO	-	-	-	
	(6,30,max)	<1	16	5	8	0	
	(6,35,max)	<1	TO	-	-	-	
firewire	(3,min)	<1	2	1	0	0	yes
	(10,min)	<1	63	1	0	0	
	(15,min)	<1	1808	1	0	0	
	(20,min)	<1	TO	1	0	0	
	(3,max)	<1	69	0	0	1	
	(10,max)	<1	TO	-	-	-	

Answer to Q4.1 (How does automatically inferring monotonicity with help of the order perform on different properties?)

There is no significant difference between reachability and expected total reward properties for the automatic inference of monotonicity in pMDPs.

Assumptions | We observe that for the benchmarks for which we do not need assumptions, checking for monotonicity is done within at most one second. Also, for these benchmarks, we can often find monotonic parameters. Only for brp (pPTA), there are instances for which we cannot obtain monotonicity, although we have a sufficient ordering of the states. This is caused by our method not being complete, i.e., local monotonicity implies global monotonicity but not vice versa. For this model, there are states for which the parameters are locally monotonic increasing, as well as states where the parameters are locally monotonic decreasing.

Now, consider the benchmarks for which we do need assumptions. First of all, many of these benchmarks have a memory-out or time-out. Recall that this also occurs for

pMCs (which only have state assumptions). Therefore, it is no surprise that this also arises for pMDPs, which additionally have action assumptions. Secondly, we see that for, e.g., csma (2,2,min), we need assumptions and obtain (several) sufficient orders for the states. Note that these orders are not necessarily valid for the entire region. Therefore, we cannot find monotonicity as the orders do not agree on monotonicity, i.e., Theorem 4.26 cannot be applied. For nrp (pPTA), zeroconf (MDP), and zeroconf (pPTA), however, we observe that for some instances, the computed orders are all witnesses for monotonicity, so Theorem 4.26 is applied, and we can find monotonicity for a subset of the parameter.

Answer to Q4.2 (How do (action and state) assumptions influence the performance of automatically inferring monotonicity with help of the order?)

Assumptions may help finding monotonicity.

4.7 Conclusion

This chapter shows a method to automatically infer monotonicity of pMDPs for both reachability and expected total reward properties. It extends the approach for pMCs (Chapter 3) by computing the minimizing scheduler on-the-fly and introducing action assumptions. If we do not need assumptions to build the order, we often find monotonicity in our experiments. However, when assumptions are needed, finding monotonicity gets more challenging as it often leads to memory and time-outs. As using the solution function is not viable for pMDPs (cf. Remark 4.4), our approach still offers a way to find monotonicity. Future work includes improvement of building the reward order, by, e.g., eliminating states which cannot be order, finding monotonicity on the input modeling language, and investigating if a reward order that is almost always valid (as in [CRSL22]) can help the analysis of pMDPs.

5 Tackling the ε-Optimal Synthesis Problem

Summary | In this chapter, we focus on the ε-optimal synthesis problem described in Section 2.4. First, we discuss parameter lifting [QDJJ+16] and the divide-and-conquer approach, which already exists in the model checker Storm. We refer to this existing divide-and-conquer approach as the vanilla approach. As we have seen in Section 3.5, monotonicity checking can benefit from having bounds for the expected total reward for states. Parameter lifting is a method to obtain these bounds. In Section 5.4, we discuss how parameter lifting can profit from monotonicity checking. Next, we suggest an integrated divide-and-conquer approach where both the monotonicity checker and the parameter lifter benefit from each other. Finally, we study the use of the integrated approach through an empirical evaluation.

Origins | This chapter is an extended version of the results presented in [3; 5]. Parameter lifting (Section 5.2) and the vanilla divide-and-conquer approach (Section 5.3) are based on [QDJJ+16]. Finally, for the heuristics (Section 5.4) we use [JJWQ+18].

Background | We consider pMCs as introduced in Section 2.2 and we refer to monotonicity checking as defined in Chapter 3. Furthermore, we consider the ε-optimal synthesis problem for pMC $\mathcal{D}$ (Section 2.4): Given tolerance $\varepsilon \geq 0$, find an instantiation $\vec{u}^*$ such that

$$\max_{\vec{u} \in R} \mathsf{ER}^{s_I \to T}(\vec{u}) \cdot (1-\varepsilon) \;\leq\; \mathsf{ER}^{s_I \to T}_{\mathcal{D}}(\vec{u}^*) \;\leq\; \max_{\vec{u} \in R} \mathsf{ER}^{s_I \to T}(\vec{u}) \,.$$

Assumption 5.1 We assume parameter $p \in V$, pMC $\mathcal{D}$, and graph-preserving region R to be given. For conciseness, we omit super/subscript R from the notation whenever R is clear from the context. We let $R(p) = [\mathsf{l}_p, \mathsf{u}_p]$ denote the lower and upper bound of p in region R.

Assumption 5.2 We assume all transition functions to be monotonic. Furthermore, to keep presentation simple, we assume transition functions to contain at most one parameter, i.e., for all $s \in S$. $|\mathsf{occur}(s)| \leq 1$. More parameters at a state can be added at the expense of a slightly more involved theory.

5.1 Related Work

Monotonicity in Markov chains | The work related to monotonicity in pMCs and the automatic inference of monotonicity is found in Section 3.1, which is mainly based on [1; 5]

Problems related to the analysis of parametric Markov chains | Many problems that are closely related to pMCs exist, e.g., Chen et al. [CFRS14] analyze (non-controllable) perturbations in MCs from a robustness perspective. Also, interval MCs [JL91; KU02] are closely related to pMCs; they can be seen as pMCs without parameter dependencies, i.e., no parameters occur at multiple states. Model repair [BGKR+11], an extension of the feasibility problem — finding a satisfying instantiation —, has been formulated as a search problem before: Chen et al. [CHHK+13] consider three different search methods: particle swarm optimization (PSO), Markov Chain Monte Carlo, and Cross-Entropy. In this context, PSO is the most successful. Further research on this is done by Cubuktepe et al., resulting in the methods quadratically-constrained quadratic programming (QCQP) [CJJK+18] and sequential convex programming (SCP) [CJJK+22]. For QCQP and SCP, Cubuktepe et al. [CJJK+17; CJJK+18; CJJK+22] take an iterative approach in which the encoding is simplified around a point to guide the search.

Analyzing parametric Markov chains | The initial approaches to solving problems for pMCs focus on computing a closed form for the solution function, which maps parameter values to expected total rewards [Daw04; LMT07; HHZ09; FGT11; JCVW+14; JÁHJ+19; BHHJ+20; FCGA21]. This approach is practically infeasible, as the solution function can be exponentially large in the number of parameters [BHHJ+20]. PSO [CHHK+13], QCQP [CJJK+18], and SCP [CJJK+22] are not suitable for finding an ε-optimal solution, as they are developed for the feasibility problem. Parameter lifting [QDJJ+16] is a method to obtain upper and lower bounds on the solution function at a given region. Parameter lifting can be used to, e.g., find (almost) optimal parameter instantiations. As a preprocessing step in the analysis of pMCs, fast sampling of the parameter space of a pMC and evaluating the corresponding MC is often used [HHZ11a; JÁHJ+19]. Storm offers optimized routines, and for large numbers of samples, just-in-time compilation is a feasible alternative [GHS18].

5.2 Parameter Lifting

Parameter lifting [QDJJ+16] is a technique to obtain a lower and upper bound for the expected total reward on a given region. For a state $s \in S$, parameter lifting provides us with bounds $L(s)$ and $U(s)$ satisfying:

$$L(s) \;\leq\; \mathsf{ER}^{s \to T}(\vec{u}) \;\leq\; U(s) \quad \text{for all } \vec{u} \in R.$$

Recall that a parameter in a pMC is dependent if it occurs at multiple states, i.e., there exists $s, s' \in S$ such that $s \neq s'$ and $\mathsf{occur}(s) \cap \mathsf{occur}(s') \neq \emptyset$ (see also page 16). The key idea of parameter lifting is to drop all parameter dependencies by introducing fresh parameters. The outcome is an *interval* MC [JL91; KU02], which can be considered a special case of pMCs in which all parameters are independent.

Definition 5.1 A pMC is an *interval MC (iMC)*, if for all $s, s' \in S$: interval MC

$$s \neq s' \;\text{ implies }\; \mathsf{occur}(s) \cap \mathsf{occur}(s') = \emptyset\,.$$

Example 5.1 Consider Figure 5.1a, the pMC $\mathcal{D}_7$ is not an iMC, as parameter p occurs at states s_0 and s_1. All parametric states of the pMC $\mathcal{D}_8$ in Figure 5.1b have different parameters, thus all parameters are independent and $\mathcal{D}_8$ is an iMC. We often depict an iMC by replacing the transition functions by intervals. For region $R = [0.1, 0.9]^2$, the iMC $\mathcal{D}_8$ is depicted in Figure 5.1c.

Recall from Theorem 3.1 that local monotonicity at all states implies global monotonicity. As all parameters in an iMC are independent, global monotonicity implies local monotonicity.

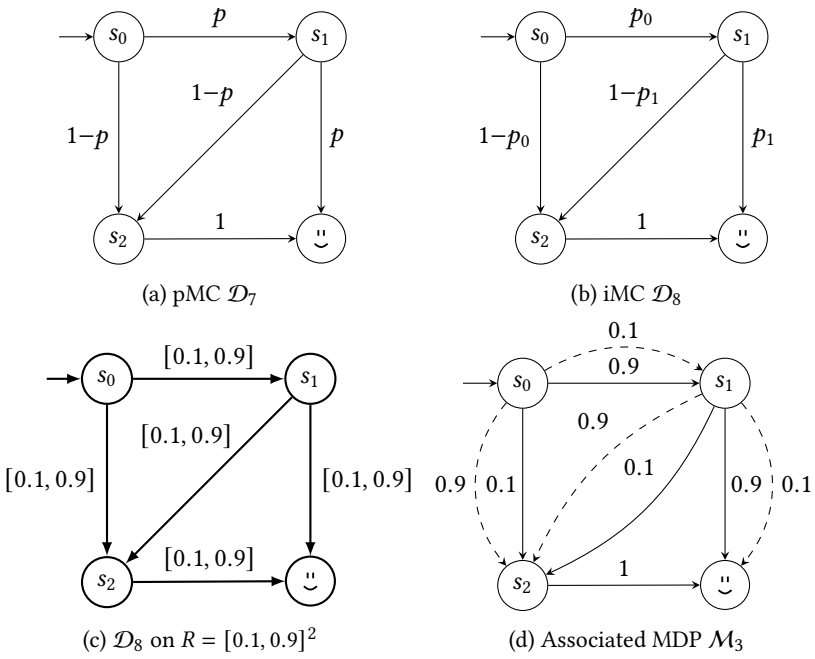
Theorem 5.1 Given iMC $\mathcal{D}$ with $\Pr^{s_I \to s} > 0$. For $s \in S$ with $p \in \mathsf{occur}(s)$, we have:

$$\mathcal{D}{\uparrow}^{\ell}_{p,s} \;\text{ iff }\; \mathcal{D}{\uparrow}_{p}\,.$$

Proof. Let $\mathcal{D}$ be an iMC and $s \in S$ such that $p \in \mathsf{occur}(s)$.
For all $s' \in S \setminus \{s\}$, we have $p \notin \mathsf{occur}(s')$, so trivially $\mathcal{D}{\uparrow}^{\ell}_{p,s'}$ and $\mathcal{D}{\downarrow}^{\ell}_{p,s'}$ hold.
$\mathcal{D}{\uparrow}^{\ell}_{p,s}$ implies $\mathcal{D}{\uparrow}_{p}$ directly follows from the above and Theorem 3.1.
Now consider the other direction. We have to show

$$\mathcal{D}{\uparrow}_{p} \text{ implies } \mathcal{D}{\uparrow}^{\ell}_{p,s}\,.$$

We proceed by contra-position. Let $\mathcal{D}{\uparrow}_{p}$ and assume $\mathcal{D}{\uparrow}^{\ell}_{p,s}$ does not hold. So



(a) pMC $\mathcal{D}_7$ (b) iMC $\mathcal{D}_8$

(c) $\mathcal{D}_8$ on $R = [0.1, 0.9]^2$ (d) Associated MDP $\mathcal{M}_3$

Figure 5.1: An example pMC, iMC and MDP with $r(s_i) = i$

there exists $\vec{v} \in R$ such that:

$$\left(\sum_{s' \in \mathrm{succ}(s)} \left(\frac{\partial}{\partial p} \mathbf{P}(s, s') \right) \cdot \mathsf{ER}^{s' \to T} \right) (\vec{v}) \; < \; 0 \; . \tag{5.1}$$

Recall from Section 2.3.4, that $\Pr^{s' \to s}(\vec{u})$ denotes the probability of eventually reaching s from s'. As s is the only state where p occurs, and we may assume T being absorbing, we have for any $s' \in S \setminus \{s\}$:

$$\frac{\partial}{\partial p} \mathsf{ER}^{s' \to T}(\vec{u}) = \left(\Pr^{s' \to s} \cdot \frac{\partial}{\partial p} \mathsf{ER}^{s \to T} \right) (\vec{u}) \; . \tag{5.2}$$

From this we obtain the following:

$$
\begin{aligned}
&\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u}) \\
&= \frac{\partial}{\partial p}\left(r(s) + \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathsf{ER}^{s'\to T}\right)(\vec{u}) \\
&= \left(\sum_{s'\in\mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T} + \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot\left(\frac{\partial}{\partial p}\mathsf{ER}^{s'\to T}\right)\right)(\vec{u}) \\
&\overset{\text{Eq. (5.2)}}{=} \left(\sum_{s'\in\mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T}\right. \\
&\qquad \left. + \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot\left(\mathrm{Pr}^{s'\to s}\cdot\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}\right)\right)(\vec{u})
\end{aligned}
$$

Let $f(\vec{u}) = \frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})$, we leave out $\vec{u}$ and observe that:

$$
\begin{aligned}
& f = \sum_{s'\in\mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T} + \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot\left(\mathrm{Pr}^{s'\to s}\cdot f\right) \\
\iff & f = \sum_{s'\in\mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T} + \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathrm{Pr}^{s'\to s}\cdot f \\
\iff & 1 - \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathrm{Pr}^{s'\to s}\cdot f = \sum_{s'\in\mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T} \\
\iff & f = \frac{\sum_{s'\in\mathsf{succ}(s)} \left(\frac{\partial}{\partial p}\mathbf{P}(s,s')\right)\cdot \mathsf{ER}^{s'\to T}}{1 - \sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathrm{Pr}^{s'\to s}}
\end{aligned}
$$

Note that $\left(\sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathrm{Pr}^{s'\to s}\right)(\vec{u})$ denotes the return probability of s. Recall from Assumption 2.6 that all states eventually reach T so for all $\vec{u}\in R$:

$$
\left(\sum_{s'\in\mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathrm{Pr}^{s'\to s}\right)(\vec{u}) < 1\,.
$$

We thus obtain:

$$\left(1 - \sum_{s' \in \mathsf{succ}(s)} \mathbf{P}(s,s') \cdot \Pr^{s' \to s}\right)(\vec{u}) > 0 \,.$$

From Equation (5.1), we now obtain that there exists a $\vec{v} \in R$ such that:

$$\frac{\partial}{\partial p} \mathsf{ER}^{s \to T}(\vec{v}) < 0 \,. \tag{5.3}$$

From Equation (5.2), we obtain for s_I and for all $\vec{u} \in R$:

$$\frac{\partial}{\partial p} \mathsf{ER}^{s_I \to T}(\vec{u}) = \left(\Pr^{s_I \to s} \cdot \frac{\partial}{\partial p} \mathsf{ER}^{s \to T}\right)(\vec{u}) \,.$$

As $\Pr^{s_I \to s}(\vec{u}) > 0$ for all $\vec{u} \in R$, we obtain from Equation (5.3) for $\vec{v}$:

$$\frac{\partial}{\partial p} \mathsf{ER}^{s_I \to T}(\vec{v}) < 0 \,.$$

This contradicts $\mathcal{D}{\uparrow_p}$, so our assumption (Equation (5.1)) is invalid, thus we obtain:

$$\mathcal{D}{\uparrow_p} \text{ implies } \mathcal{D}{\uparrow^{\ell}_{p,s}} \,. \qquad \blacksquare$$

Definition 5.2 The *relaxation* of pMC $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ is the iMC $\mathsf{relax}(\mathcal{D}) = (S, s_I, T, V', \mathbf{P}')$ with independent variables $V' = \{p_s \mid s \in S \text{ and } \mathsf{occur}(s) \neq \emptyset\}$, and $\mathbf{P}'(s,s') = \mathbf{P}(s,s')[\mathsf{occur}(s) \leftarrow p_s]$.

For state s with $\mathsf{occur}(s) = p$, let $\mathsf{relax}(R)(p_s) = R(p)$. Likewise, an instantiation in $\vec{u} \in R$ is mapped to $\mathsf{relax}(\vec{u})$ by $\mathsf{relax}(\vec{u})(p_s) = \vec{u}(p)$ for $\mathsf{occur}(s) = p$.

Example 5.2 For pMC $\mathcal{D}_7$ in Figure 5.1a, the relaxed pMC $\mathsf{relax}(\mathcal{D}_7)$ is depicted by $\mathcal{D}_8$ in Figure 5.1b.

As we assume all transition functions to be monotonic, the extremal values of the expected total reward on an iMC are reached at the extremal values of a region. So the maximal expected total reward in $\mathcal{D}$ is bounded by the maximal expected total reward in $\mathsf{relax}(\mathcal{D})$. Formally [QDJJ+16], for each state s and region R in pMC $\mathcal{D}$:

$$\max_{\vec{u} \in R} \mathsf{ER}^{s \to T}_{\mathcal{D}}(\vec{u}) \;\leq\; \max_{\vec{u} \in \mathsf{relax}(R)} \mathsf{ER}^{s \to T}_{\mathsf{relax}(\mathcal{D})}(\vec{u}). \tag{5.4}$$

The right-hand side of (5.4) denotes the upper bound on R ($U_R(s)$). Analogously, we

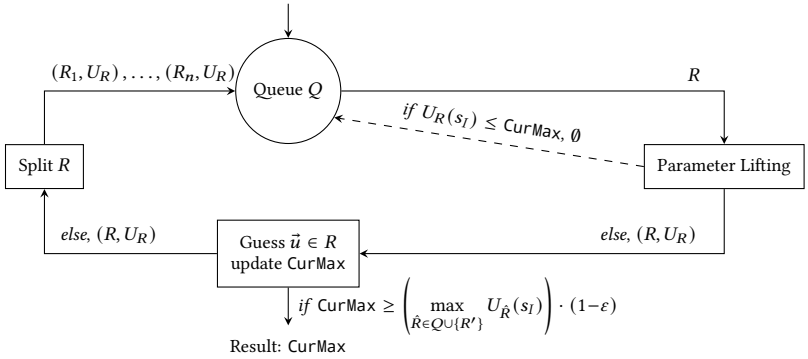


Figure 5.2: Divide-and-conquer with pure parameter lifting (taken from [3])

define the lower bound $L_R(s)$. We obtain the extremal values for the expected total reward in an iMC by interpreting the iMC as an MDP. We introduce a non-deterministic choice on the lower and upper bound for each parameter and apply off-the-shelf MDP model checking.

Example 5.3 Reconsider pMC $\mathcal{D}_7$ on region $R_2 = [0.1, 0.9]$ and its relaxation $\mathcal{D}_8 = \mathsf{relax}(\mathcal{D}_7)$ as depicted in Figure 5.1. The associated MDP is depicted in Figure 5.1d. The dashed line denotes the lower bound for p, whereas the solid line denotes the upper bound. For regions R_2 and $\mathsf{relax}(R_2) = [0.1, 0.9]^2$, we obtain:

$$\max_{\vec{u} \in R_2} \mathsf{ER}^{s_0 \to T}_{\mathcal{D}_7}(\vec{u}) = \frac{17}{8} \leq \frac{68}{25} = \max_{\vec{u} \in \mathsf{relax}(R_2)} \mathsf{ER}^{s_0 \to T}_{\mathcal{D}_8}(\vec{u}).$$

5.3 Divide-and-Conquer

Recall that we consider the ε-optimal synthesis problem, so we want to find an ε-optimal expected total reward without knowing the "real" optimal value since the optimal synthesis problem is ETR-hard [JKPW21]. Figure 5.2 shows how the extremal value of the expected total reward for region R_i, pMC $\mathcal{D}$, and precision ε can be computed using parameter lifting. The main idea is to analyze regions and split them if the result is inconclusive. The approach uses a queue of regions (Q) that need to be checked and the largest value CurMax currently found. Furthermore, we know the (possibly trivial) upper bound for the regions in the queue, i.e., $U^{\max} = \max_{\hat{R} \in Q} U_{\hat{R}}(s_I)$. We iteratively check regions and improve both bounds until an ε-optimal solution is

found, i.e., $\mathsf{CurMax} \geq U^{\max} \cdot (1-\varepsilon)$.

Initially, the queue only contains R_t with the trivial upper bound $U_{R_t}(s_I) = \infty$. We initialize the maximum expected total reward CurMax so far as 0, as the rewards are non-negative; this is a valid lower bound on the maximal expected total reward. For a selected R from the queue, we compute an upper bound U_R with parameter lifting. If U_R at the initial state is below the current optimum (CurMax), we discard R, as there does not exist a $\vec{u} \in R$ such that $\mathsf{ER}^{s_I \to T}(\vec{u}) > \mathsf{CurMax}$ (cf. Equation (5.4)). Otherwise, we try to improve CurMax by guessing $\vec{u} \in R$ and computing $\mathsf{ER}^{s_I \to T}(\vec{u})$. To compute $\mathsf{ER}^{s_I \to T}(\vec{u})$, we use an instantiation checker that reuses the model-checking results from the last guess. If $\mathsf{ER}^{s_I \to T}(\vec{u})$ exceeds CurMax, we update CurMax. Now, we check whether we can terminate: Recall that the maximum so far is bounded by $U^{\max}$, and we are interested in an ε-optimal solution. So our ε-upper bound is given by: $U^{\varepsilon} = U^{\max} \cdot (1-\varepsilon)$. If U^{ε} is smaller than CurMax, we are done and return CurMax together with the associated $\vec{u}$. Otherwise, we split R into smaller regions $(R_1, \ldots, R_n)$. We add $R_1, \ldots, R_n$ to the queue with upper bound U_R. We continue by picking a new region from the queue and repeating the above. By default, we split R along all dimensions and pick $\vec{u}$ at the center point of the region. This algorithm converges in the limit [QDJJ+16].

Example 5.4 Reconsider pMC $\mathcal{D}_7$ of Figure 5.1a and assume we want to find p such that $\mathsf{ER}^{s_0 \to \lightning}(p)$ is ε-optimal for $\varepsilon = 0.1$.

iter.	Q	CurMax	$U_R(s_I)$	p	$\mathsf{ER}^{s_I \to \lightning}(p)$
1	$\underline{([\mathbf{0.1, 0.9}], \infty)}$	0	2.72	0.5	2
2	$\underline{([\mathbf{0.1, 0.5}], \mathbf{2.72})}$, $([0.5, 0.9], 2.72)$	2	2.4	0.3	2.12
3	$\underline{([\mathbf{0.5, 0.9}], \mathbf{2.72})}$, $([0.1, 0.3], 2.4)$, $([0.3, 0.5], 2.4)$	2.12	2	-	-
4	$\underline{([\mathbf{0.1, 0.3}], \mathbf{2.4})}$, $([0.3, 0.5], 2.4)$	2.12	2.24	0.2	2.12
5	$\underline{([\mathbf{0.3, 0.5}], \mathbf{2.4})}$, $([0.1, 0.2], 2.24)$, $([0.2, 0.3], 2.24)$	2.12	2.2	0.4	2.08

The table above shows how the divide-and-conquer approach evolves for initial region $R = [0.1, 0.9]$. The queue contains the regions and their upper bounds. Furthermore, we keep track of CurMax. The chosen region is underlined and bold. We pick p at the region's center. In the first two iterations, we update CurMax, as $\mathsf{ER}^{s_I \to T}(0.5)$ is larger than the optimal expected total reward found so far. Furthermore, we split the region, as CurMax is not ε-close to the upper bound. In the third iteration, we consider region $R = [0.5, 0.9]$; as the computed bound for this region is below CurMax, we can safely discard the region and do not calculate $\mathsf{ER}^{s_I \to \lightning}$. The fourth iteration proceeds as the first two, although we do not update CurMax, as $\mathsf{ER}^{s_I \to T}(0.2)$ is not strictly larger than CurMax. In the fifth iteration,

we consider region $[0.3, 0.5]$ for which we have bound 2.4; we update this to 2.2. Now the upper bound on the regions in the queue and the current region is 2.24, as $2.12 > 2.24 \cdot 0.9$, we return $p = 0.3$ with extremal value 2.12.

In the remainder, we refer to the divide-and-conquer approach using only parameter lifting as the vanilla approach.

Remark 5.1 We described the vanilla approach w.r.t. relative precision ε; however, for reachability properties, one could also consider an absolute precision, as the upper bound of reachability probabilities is trivially 1. To deal with absolute precision, we replace $\texttt{CurMax} \geq \max_{\hat{R} \in Q \cup \{R'\}} U_{\hat{R}}(s_I) \cdot (1-\varepsilon)$ with $\texttt{CurMax} \geq \max_{\hat{R} \in Q \cup \{R'\}} U_{\hat{R}}(s_I) - \varepsilon$.

5.4 Improving Parameter Lifting

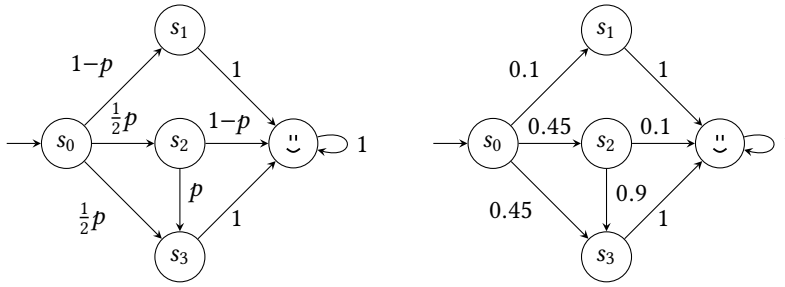
In Section 3.5, we have seen how monotonicity checking can benefit from parameter lifting. In this section, we show how parameter lifting can benefit from monotonicity checking. First of all, consider global monotonicity. If a parameter is globally monotonic, we can directly replace the parameter by its extremal value, making relaxation for this parameter superfluous.

Lemma 5.2 Given pMC $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ and region R. Let $\mathcal{D}' = (S, s_I, T, V', \mathbf{P}')$ with $V' = V \setminus \{p\}$ and $\mathbf{P}'(s, s') = \mathbf{P}(s, s')[p \leftarrow \mathsf{u}_p]$ and region $R' = \bigtimes_{q \in V'} R(q)$. Then:

$$\mathcal{D}{\uparrow_p} \quad \text{implies} \quad \max_{\vec{u} \in R} \mathsf{ER}^{s \xrightarrow{} T}_{\mathcal{D}}(\vec{u}) = \max_{\vec{u} \in R'} \mathsf{ER}^{s \xrightarrow{} T}_{\mathcal{D}'}(\vec{u}) \text{ for all } \vec{u} \in R\,.$$

Example 5.5 Reconsider our running example pMC $\mathcal{D}_1$ from Chapter 3, also depicted in Figure 5.3a on region $R = [0.1, 0.9]$. From monotonicity checking as described in Chapter 3, we obtain $\mathsf{ER}^{s_0 \xrightarrow{} \{\!\{\}\!\}}(p){\uparrow_p}$. Instead of relaxing $\mathcal{D}_1$ and analyzing the associated MDP, we directly replace parameter p by 0.9, yielding the parameter-free MC in Figure 5.3b. Analyzing this MC yields $\mathsf{ER}^{s_0}(\Diamond\, \{\!\{\}\!\}) = 3.565$.

As we have seen above, global monotonicity simplifies parameter lifting. Next, we show how we can shrink the region and pMC given global monotonicity. Let $X^R_{\uparrow}$ denote the globally monotonic increasing parameters on R, and similarly, let $X^R_{\downarrow}$ denote globally monotonic decreasing parameters on R. For brevity, we omit the superscript



(a) pMC $\mathcal{D}_1$ with $r(s_i) = i$ (b) $\mathcal{D}_1$ for $p = 0.9$

Figure 5.3: Running example (parametric) Markov chain

R in the following. We define region $\mathsf{Shrink}^{max}_{X_\uparrow, X_\downarrow}(R)$ as follows:

$$\mathsf{Shrink}^{max}_{X_\uparrow, X_\downarrow}(R)(p) = \begin{cases} \mathsf{l}_p & \text{if } p \in X_\downarrow \\ \mathsf{u}_p & \text{if } p \in X_\uparrow \\ R(p) & \text{otherwise} . \end{cases}$$

Furthermore, for pMC $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ with $X_\uparrow$ and $X_\downarrow$, let pMC $\mathsf{Shrink}^{max}_{X_\uparrow, X_\downarrow}(\mathcal{D}) = (S, s_I, T, V', \mathbf{P}')$ with $V' = V \setminus (X_\uparrow \cup X_\downarrow)$ and

$$\mathbf{P}'(s, s') = \begin{cases} \mathbf{P}(s, s')[p \leftarrow \mathsf{l}_p] & \text{if } p \in X_\downarrow \\ \mathbf{P}(s, s')[p \leftarrow \mathsf{u}_p] & \text{if } p \in X_\uparrow \\ \mathbf{P}(s, s') & \text{otherwise} , \end{cases}$$

denote $\mathcal{D}$ modified to obtain the maximal expected total reward. Analogously, we define $\mathsf{Shrink}^{min}_{X_\uparrow, X_\downarrow}(R)$ and $\mathsf{Shrink}^{min}_{X_\uparrow, X_\downarrow}(\mathcal{D})$, with $X_\downarrow$ and $X_\uparrow$ swapped, to obtain the minimizing region and the minimal expected total reward, respectively. We now lift Lemma 5.2 to the case of several monotonic parameters yields Proposition 5.3.

Proposition 5.3 For pMC $\mathcal{D}$ with $X_\uparrow$ and $X_\downarrow$, the following holds:

$$\max_{\vec{u} \in R} \mathsf{ER}^{s \to T}_{\mathcal{D}}(\vec{u}) = \max_{\vec{u} \in \mathsf{Shrink}_{X_\uparrow, X_\downarrow}(R)} \mathsf{ER}^{s \to T}_{\mathsf{Shrink}^{max}_{X_\uparrow, X_\downarrow}(\mathcal{D})}(\vec{u}) .$$

The following lemma suggests how we can use local monotonicity to improve the analysis of the MDP obtained via parameter lifting.

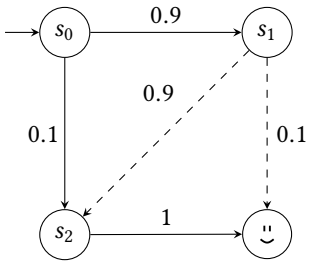


Figure 5.4: MC $\mathcal{D}_9$, obtained by applying parameter lifting with monotonicity checking to $\mathcal{D}_7$

Lemma 5.4 Given pMC $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ and region R.
Let $\mathcal{D}' = (S, s_I, T, V, \mathbf{P}')$ such that

$$\mathbf{P}'(s,s') = \begin{cases} \mathbf{P}(s,s')[p \leftarrow \mathsf{u}_p] & \text{if } \mathcal{D}{\uparrow}^{\ell}_{p,s} \\ \mathbf{P}(s,s')[p \leftarrow \mathsf{l}_p] & \text{if } \mathcal{D}{\downarrow}^{\ell}_{p,s} \\ \mathbf{P}(s,s') & \text{otherwise} \end{cases} .$$

Now the following holds:

$$\max_{\vec{u} \in R} \mathsf{ER}^{s \to T}_{\mathcal{D}}(\vec{u}) = \max_{\vec{u} \in R'} \mathsf{ER}^{s \to T}_{\mathcal{D}'}(\vec{u}) .$$

Example 5.6 Consider pMC $\mathcal{D}_7$ of Figure 5.1b on region $R = [0.1, 0.9]$. Following the monotonicity checking approach of Chapter 3, we obtain $\mathcal{D}_7{\uparrow}^{\ell}_{p,s_0}$ and $\mathcal{D}_7{\downarrow}^{\ell}_{p,s_1}$. When analyzing the MDP $\mathcal{M}_3$ (Figure 5.1d) obtained through applying parameter lifting to $\mathcal{D}_7$, we fix the scheduler for s_0 and s_1 to the upper bound and lower bound, respectively. We thus analyze MC $\mathcal{D}_9$, see Figure 5.4.

Remark 5.2 If for all parametric states we have local monotonicity, the resulting model (after parameter lifting with monotonicity information) will be a parameter-free MC, otherwise, we are still left to model check an MDP. However, the number of non-deterministic states is smaller compared to this number in the MDP obtained when using parameter lifting without monotonicity information.

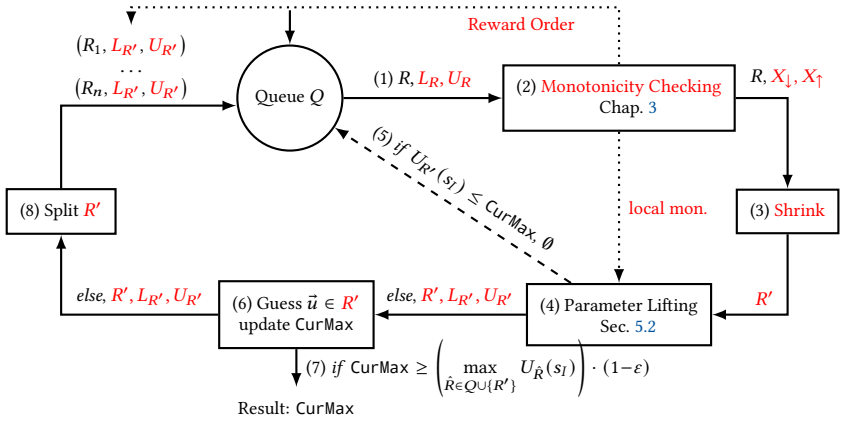


Figure 5.5: Divide-and-conquer with parameter lifting and monotonicity checking (taken from [5]). In red the new elements compared to the vanilla approach in Fig. 5.2

5.5 Improving Divide-and-Conquer

In this section, we improve the divide-and-conquer approach using parameter lifting by using monotonicity checking. Figure 5.5 shows this integrated approach. In red we indicate the new parts compared to Figure 5.2.

In this section, we let R' denote $\mathsf{Shrink}^{max}_{X_\uparrow, X_\downarrow}(R)$, $\mathcal{D}^U = \mathsf{Shrink}^{max}_{X_\uparrow, X_\downarrow}(\mathcal{D})$, and $\mathcal{D}^L = \mathsf{Shrink}^{min}_{X_\uparrow, X_\downarrow}(\mathcal{D})$. Furthermore, we let $U^{\max} = \max_{\hat{R} \in Q} U_{\hat{R}}(s_I)$ and $U^\varepsilon = U^{\max} \cdot (1-\varepsilon)$ denote the upper bound and ε-optimal upper bound on the queue, respectively.

5.5.1 Overall algorithm

Instead of only computing the upper bound for a region, each region is now also equipped with a lower bound, i.e., for all regions R, we have $L_R(s)$ and $U_R(s)$ such that $L_R(s) \leq \mathsf{ER}^{s \to T}(\vec{u}) \leq U_R(s)$ for every state $s \in S$ and instantiation $\vec{u} \in R$. Furthermore, for each region R in the queue, we keep track of monotonicity information about the monotonic increasing (and decreasing) parameters on R. To this end, we also store the reward order since the monotonicity information is based on this.

Now consider the integrated approach, as depicted in Figure 5.5. Initially, the input region R_ι is extended with $L_{R_\iota}(s) = 0, U_{R_\iota}(s) = \infty$ for every $s \in S$ and empty monotonicity information. As for the vanilla approach, we iteratively pick regions from the queue and analyze them. However, we now additionally analyze the regions

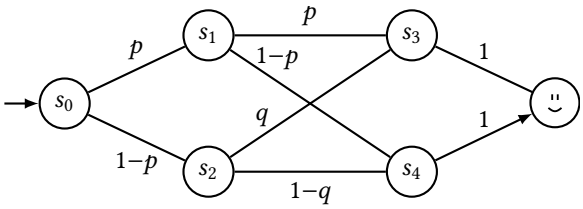


Figure 5.6: An example pMC $\mathcal{D}_{10}$ with $r(s_3) = 1$ and for $s \in S \setminus \{s_3\}$ $r(s) = 0$

w.r.t. monotonicity. First, (1) a region R and the associated reward order are taken from the queue Q, and (2) its monotonicity is computed while using the bounds L_R and U_R. Moreover, we compute the globally monotonic increasing ($X_\uparrow$) and decreasing parameters ($X_\downarrow$). Note that for conciseness, we omit superscript R. Next, we (3) shrink a region based on global monotonicity. Recall from Proposition 5.3 that we obtain $\max_{\vec{u} \in R} \mathsf{ER}^{s \to T}_{\mathcal{D}}(\vec{u}) = \max_{\vec{u} \in R'} \mathsf{ER}^{s \to T}_{\mathcal{D}^U}(\vec{u})$ for all $s \in S$. So, we can safely discard the instantiations in $R \setminus R'$. Then, we (4) apply parameter lifting, as described in Section 5.2, enhanced with the local monotonicity information from (2). I.e., We analyze region R' to get bounds $L_{R'}, U_{R'}$. We, first of all, observe that $L_R(s) \leq L_{R'}(s)$ and $U_{R'}(s) \leq U_R(s)$ for every $s \in S$, so the bounds for R' are at least as tight as the bounds for R. Furthermore, we observe that if all parameters are globally monotonic, the region R' is a singleton and straightforward to analyze.

> **Remark 5.3** The following does not hold in general: consider pMC $\mathcal{D}$ with $U_{R'}(s) = U_R(s)$. Let $p \in X_\uparrow$, $s, s' \in S$ such that $s \neq s'$, $\mathcal{D}{\uparrow}^{\ell}_{p,s}$, and $\mathcal{D}{\downarrow}^{\ell}_{p,s'}$. When analyzing $\mathcal{D}$, parameter lifting sets p to u_p at s and l_p at s', Whereas for $\mathcal{D}^U$ at both s and s', u_p is chosen. If $\mathsf{u}_p \neq \mathsf{l}_p$, then we have for initial state s_I: $U_{R'}(s_I) < U_R(s_I)$. Analogously, $L_R(s) = L_{R'}(s)$ does not hold in general.

We now check if we can discard R' by comparing the obtained upper bound to the current extremal value, i.e., if $U_{R'}(s_I) \leq \mathtt{CurMax}$, then we discard R' and go to (1). Otherwise, we (6) guess a $\vec{u} \in R'$ and set $\mathtt{CurMax}$ to $\max(\mathtt{CurMax}, \mathsf{ER}^{s_I \to T}(\vec{u}))$. If (7) $\mathtt{CurMax} \geq U^\epsilon$, we have found a solution to our problem statement and return $\mathtt{CurMax}$. Otherwise, we have not yet found a solution, so we (8) *split* the region R' into smaller (rectangular) regions $R_1, \ldots, R_n$. As for the vanilla approach, the sub-regions inherit the bounds of their parent region R'. The bounds are refined once the region is picked from the queue and analyzed. Termination in the limit (i.e., convergence of the lower and upper bound to the limit) follows from the termination of monotonicity checking and the termination of the loop in Figure 5.2.

Example 5.7 Consider pMC $\mathcal{D}_{10}$ in Figure 5.6 with $r(s_3) = 1$ and $r(s) = 0$ for $s \in S \setminus \{s_3\}$. Let $R(p) = [0.1, 0.9]$, $R(q) = [0.1, 0.5]$ and $\varepsilon = 0.1$.

iter.	Q	CurMax	Rew. Order	R'	$L_{R'}, U_{R'}$	$\vec{u}$	$\mathsf{ER}^{s_0 \to \lightning}$
1	$\underline{(\mathbf{R}, \mathbf{0}, \infty, \bot)}$	0	$s_4 \preceq^{\mathsf{rew}} s_1$ $s_4 \preceq^{\mathsf{rew}} s_2$ $s_1 \preceq^{\mathsf{rew}} s_3$ $s_2 \preceq^{\mathsf{rew}} s_3$	$[0.1, 0.9]$	0.1, 0.86	(0.5, 0.5)	0.5
2	$\underline{([\mathbf{0.1}, \mathbf{0.5}], \mathbf{0.14}, \mathbf{0.86}, \preceq_1^{\mathsf{rew}})}$ $([0.5, 0.9], 0.14, 0.86, \preceq_1^{\mathsf{rew}})$	0.5	$\preceq_1^{\mathsf{rew}}$	$[0.1, 0.5]$	0.1, 0.5	-	-
3	$\underline{([\mathbf{0.5}, \mathbf{0.9}], \mathbf{0.14}, \mathbf{0.86}, \preceq_1^{\mathsf{rew}})}$	0.5	$s_4 \preceq^{\mathsf{rew}} s_2$ $s_2 \preceq^{\mathsf{rew}} s_1$ $s_1 \preceq^{\mathsf{rew}} s_3$	-	0.3, 0.86	(0.5,0.9)	0.86

The table above shows the steps of the integrated divide-and-conquer approach. The chosen region is underlined and bold. We let $\bot$ denote the empty reward order, and $\preceq_i^{\mathsf{rew}}$ denote the reward order as obtained in iteration i. In the first iteration, we obtain global monotonicity for q. We cannot obtain monotonicity for p since the resulting order is insufficient for state s_0 as s_1 and s_2 cannot be ordered. Recall from the definitions $\mathcal{D}^L$ and $\mathcal{D}^U$ that for $L(s)$, we set q to 0.1, and for $U(s)$, we take $q = 0.5$. For $\vec{u}$, we take the center point for p and the upper bound u_q for q. As CurMax is not ε-close to $U(s_I)$, we split region R'. In the second iteration, we observe that $U(s_I) \leq$ CurMax; therefore, we discard the region. Finally, in the third iteration, we find $s_2 \preceq^{\mathsf{rew}} s_1$. We obtain global monotonicity for p as well, so our region becomes a singleton. We now pick $\vec{u}$ for both p and q at their upper bound, yielding CurMax $= 0.86$. This CurMax is not only ε-close to the optimum, but it is also the exact (so far unknown) optimum.

iter.	Q	CurMax	$U_R(s_I)$	$\vec{u}$	$\mathsf{ER}^{s_0 \to \lightning}$
1	$\underline{([\mathbf{0.1}, \mathbf{0.9}] \times [\mathbf{0.1}, \mathbf{0.5}], \infty)}$	0	0.86	(0.5, 0.3)	0.4
2	$\underline{([\mathbf{0.1}, \mathbf{0.5}] \times [\mathbf{0.1}, \mathbf{0.3}], \mathbf{0.86})}$ $([0.1, 0.5] \times [0.3, 0.5], 0.86)$ $([0.5, 0.9] \times [0.1, 0.3], 0.86)$ $([0.5, 0.9] \times [0.3, 0.5], 0.86)$	0.4	0.4	-	-

The table above shows the first two steps of applying the vanilla approach to the same model. If we compare this to the integrated approach, we see that after the first iteration, we have four regions in the queue instead of two; also CurMax of the vanilla approach is smaller than the CurMax in the integrated approach.

5.5.2 Incrementality

The key aspect of our approach is its incrementality. First of all, parameter lifting is incremental, as it efficiently reuses the associated MDP structure. Secondly, we observe that the way we check for monotonicity (i.e., by using the reward order) is also incremental For any region $R' \subseteq R$, we have that the monotonicity result of R transfers to R'. Moreover, if for region R, we have $s \preceq^{\mathsf{rew}} s'$, then this transfers to R'. Furthermore, we only use assumptions for the monotonicity checking if exactly one assumption can be proven valid on the considered region. If this is not the case, our monotonicity checker terminates in the iteration. We then store the state at which we stopped, such that in a subsequent iteration, we can quickly check where we gave up and whether the refined bounds allow us to continue building the reward order. Finally, even though we have to duplicate the order for each region, we only do this as long as the computed reward order is insufficient. Also, we only duplicate an order when modifications are necessary. Until then, we only track which regions point to this specific order.

5.5.3 Heuristics

Although our approach is incremental, there are still several choices to make in the implementation which significantly influence the performance. Note that our choices do not influence our approach's correctness. Below, we discuss the most important choices and how we resolve those in the implementation.

Initializing `CurMax` | The vanilla approach, as implemented in Storm, is only capable of dealing with a handful of parameters. It initializes `CurMax` by sampling all vertices of the region, which is exponential in the number of parameters. To scale to more parameters, we discard this sampling and instead sample each parameter independently to find out which parameters are possibly globally monotonic. We skip the parameters that are already known to be globally monotonic. We proceed as follows: we distribute 50 points evenly along the interval of the considered parameter. We fix all other parameter values in the following manner: If parameter p is monotonic increasing, we set it to its upper bound u_p. If it is monotonic decreasing, we set it to its lower bound l_p. If we do not know the monotonicity of p, we set it to its center point, i.e., $\frac{1}{2}\left(\mathsf{u}_p + \mathsf{l}_p\right)$. We now check whether the sample points suggest monotonicity or not. Finally, we initialize `CurMax` at the value of $\mathsf{ER}^{s_I \to T}(\vec{u})$, where for $\vec{u}$ all (possibly) monotonic parameters are set to their bounds, and all non-monotonic parameters are set to their center point.

Example 5.8 We continue from Example 5.7 on pMC $\mathcal{D}_{10}$ with $R(p) = [0.1, 0.9]$, $R(q) = [0.1, 0.5]$ and $\varepsilon = 0.1$. We already know that $\mathcal{D}_{10}{\uparrow_q}$, so we set $\vec{u}(q) = 0.5$ and sample over p. If we consider three sample points, we obtain the following sampling results:

p	$\mathsf{ER}^{s_0 \rightarrow \breve{\cup}}$
0.1	0.46
0.5	0.5
0.9	0.86

From this sampling, we obtain that p is possibly monotonic increasing, so when initializing `CurMax`, we pick $\vec{u}(p) = 0.9$. This gives us `CurMax` $= 0.86$, so the integrated approach with improved initialization of `CurMax` returns after the first iteration.
Note that if we consider more sample points (e.g., 5), we observe that the function is not monotonic in p, we then initially have `CurMax` $= \mathsf{ER}^{s_0 \rightarrow \breve{\cup}}(0.5, 0.5) = 0.5$.

Updating `CurMax` | To terminate quickly, it is essential to find a large value for `CurMax` as fast as possible. In the vanilla approach, `CurMax` is updated by comparing its current value to the value obtained by taking $\vec{u}$ at the region's center. We observe that taking $L(s_I)$ or the expected total reward at the region's center point is often too pessimistic. However, sampling at too many places within the region yields significant overhead. Therefore, we reuse the results of initializing `CurMax`, updated with the so far obtained monotonicity information. We distinguish three cases.

- If a parameter is globally monotonic, we fix it according to its monotonicity information.
- If a parameter is not known to be globally monotonic but was possibly monotonic at initialization, we assume it is also possibly monotonic on the considered region.
- If a parameter is not known to be globally monotonic and was not possibly monotonic at initialization, it might be that it is monotonic now, as we split the initial region one or more times. However, we do not make up for this, as resampling also takes time. So we take this parameter's center point (in the current region).

Splitting the region | When splitting the region, there are two choices to be made. First of all, we need to determine in which parameters we split the region. Secondly, we need to decide where to split for each parameter.

Let us consider the latter choice. Suppose a parameter is not monotonic on the initial region. We stick to splitting at the region's center (as the vanilla approach does),

as we have no information on what splitting point gives us better bounds. For possibly monotonic parameters, we considered three trivial splitting points: splitting at the center, splitting close to the lower bound (i.e., $l_p + \frac{1}{4}(u_p - l_p)$), and splitting close to the upper bound (i.e., $l_p + \frac{3}{4}(u_p - l_p)$). Unfortunately, we had little success in experimenting with these trivial approaches. Therefore, we stick to the center point for the possibly monotonic parameters.

Next, we want to determine in which parameters to split the region. We observe that splitting in monotonic parameters does not bring anything. Splitting in all parameters at once, as the vanilla approach does, is also infeasible for our approach, as the following example shows. Assume we have $n = 10$ parameters that are not known to be globally monotonic, splitting the region in all directions then yields $2^n = 1024$ regions we need to analyze. If we have to split half of them again, we get more than 500,000 regions. Which, in turn, might have to be split again, so we have to find a different approach. Therefore, we decide to split in one parameter at a time and use the region estimates, which are provided by the parameter lifter, as introduced by Junges et al. in [JJWQ+18], to determine which parameter we split in. These estimates indicate how well the scheduler for the MDP agrees with the parameter dependencies. The value is computed as the sum of the differences between the local lower and upper bound on the expected total reward over all states with this parameter. The more the scheduler agrees with the parameter values, the lower the value. Moreover, for parameters that are locally monotonic increasing at all states (thus globally monotonic increasing), the estimate will be 0. As long as the region estimate is larger than 10^{-4}, we split the region in the parameter with the largest estimate. Otherwise, we split the region in the parameter with the largest interval.

Priorities in the region queue | In [QDJJ+16], the goal is to partition the region into safe and unsafe sub-regions such that a given percentage of the region is covered. Therefore, the standard technique splits the region based on its size, so it is a breadth-first search on the regions. We, however, want to find the extremal value within the entire region. Therefore, we prioritize sub-regions based on their upper bound, i.e., for sub-regions $R_1, R_2 \subseteq R$, we prioritize R_1 over R_2 if $U_{R_1}(s_I) \geq U_{R_2}(s_I)$. We use the age of a region to break ties.

Dealing with the monotonicity checker | First of all, we additionally need to compute lower bounds for the monotonicity checker (in contrast to the vanilla approach, where we only need to compute upper bounds). The overhead of this is discussed in Section 5.6. Secondly, if the number of regions grows, the number of copies of the reward order also grows. Therefore, we only continue extending the reward order (thus checking for monotonicity) as long as the reward order is sufficient for less than 25% of the states. Next to a reduction in the number of copies, this also reduces the

Table 5.1: Comparing vanilla parameter lifting to the integrated approach for reachability properties

			ε: 0.1					ε: 0.05									
			integrated heuristic			vanilla		heuristic			integrated no h-CurMax			no h-order			vanilla
dir	name	instance	# i	# i_b	t	# i	t	# i	# i_b	t	# i	# i_b	t	# i	# i_b	t	# i	t
Minimize	evade	(1,2,0,1)	0	2	<1	-	MO	0	2	<1	-	-	TO	0	2	<1	-	MO
		(1,2,1,1)	0	2	<1	-	MO	0	2	<1	-	-	MO	0	2	<1	-	MO
		(1,2,3,1)	0	2	<1	-	MO	0	2	1	-	-	MO	0	2	1	-	MO
	maze	(25)	0	2	<1	1	<1	0	2	<1	0	2	<1	0	2	<1	-	MO
		(1000)	0	2	<1	1	<1	0	2	<1	0	2	<1	0	2	<1	-	MO
		(10000)	0	2	115	1	2	0	2	166	0	2	239	0	2	152	-	MO
	nrp	(5,1)	0	2	<1	-	MO	5	2	<1	-	-	MO	5	4	<1	-	MO
		(10,1)	0	2	<1	1	<1	3	2	<1	-	-	MO	3	3	<1	-	MO
		(15,1)	0	2	<1	1	<1	0	2	<1	-	-	MO	0	2	<1	-	MO
Maximize	evade	(1,2,0,1)	0	2	<1	-	MO	0	2	<1	-	-	MO	0	2	< 1	-	MO
		(1,2,1,1)	0	2	<1	-	MO	0	2	<1	-	-	MO	0	2	<1	-	MO
		(1,2,3,1)	0	2	<1	-	MO	0	2	1	-	-	MO	0	2	1	-	MO
	maze	(25)	0	2	<1	1	<1	0	2	<1	0	2	<1	0	2	<1	-	MO
		(1000)	0	2	<1	1	<1	0	2	<1	0	2	<1	0	2	<1	-	MO
		(10000)	0	2	118	1	2	0	2	160	0	2	263	0	2	111	-	MO
	nrp	(5,1)	417	2	<1	-	MO	4833	2	<1	-	-	MO	4833	2418	<1	-	MO
		(10,1)	12891	2	2	-	MO	3298331	2	750	-	-	MO	3298331	1649167	764	-	MO
		(15,1)	-	-	MO	-	MO	-	-	MO	-	-	MO	-	-	MO	-	MO

overhead for computing lower bounds, as we can stop as soon as we stop extending the reward order.

> **Remark 5.4** In the empirical evaluation, we refer to the initialization and update of `CurMax` as `h-CurMax`. Furthermore, we let `h-order` denote the heuristic, which only extends the reward order as long as the reward order is sufficient for less than 25% of the states.

5.6 Empirical Evaluation

In this section, we compare the integrated approach, as described in this chapter, to the already existing vanilla approach (as already present in Storm [HJKQ+22]). We implemented the integrated approach in the probabilistic model checker Storm [HJKQ+22]. We use the general setup as described in Appendix B.1. We took the pMCs with more than two parameters and non-trivial reachability and expected total reward properties as described in Appendix B.2.1. For the reachability properties, we consider an absolute precision (see Remark 5.1), this is also the way it was originally implemented in Storm. We modified the implementation to allow for relative precision, so that we can handle the ε-optimal synthesis problem for expected total reward properties. We decided not to use absolute precision for expected total reward properties as we do not know the optimal value in advance, so finding a suitable absolute precision is cumbersome. For reachability properties, however, we know that the result is a probability, thus

Table 5.2: Comparing vanilla parameter lifting to the integrated approach for expected total reward properties

			ε: 0.1					ε: 0.05								
			integrated heuristic			vanilla		heuristic			integrated no h-CurMax			no h-order		
dir	name	instance	# i	# i_b	t	# i	t	# i	# i_b	t	# i	# i_b	t	# i	# i_b	t
Minimize	egl	(5,2)	0	2	<1	-	MO	0	2	<1	15	2	< 1	0	2	<1
		(6,2)	0	2	<1	-	MO	0	2	<1	15	2	<1	0	2	<1
		(7,2)	0	2	<1	-	MO	0	2	<1	15	2	<1	0	2	<1
	4x4-grid	(2)	7	5	<1	-	MO	19	11	<1	-	-	MO	19	11	<1
		(4)	377385	188694	872	-	MO	-	-	TO	-	-	MO	-	-	TO
		(8)	-	-	TO	-	MO	-	-	TO	-	-	TO	-	-	TO
	maze	(2)	83	43	<1	-	MO	1955	979	3	-	-	MO	1955	979	3
		(4)	-	-	MO	-	MO	-	-	MO	-	-	MO	-	-	MO
	network_dp	(2,1,1)	0	2	<1	-	MO	47	2	<1	-	-	MO	47	25	<1
		(2,3,1)	-	-	MO	-	TO	-	-	MO	-	-	MO	-	-	MO
		(4,1,1)	0	2	<1	-	TO	405	2	<1	-	-	MO	0	2	<1
		(4,3,1)	-	-	MO	-	TO	-	-	MO	-	-	MO	-	-	MO
	network_ps	(2,1,1)	0	2	<1	-	MO	0	2	<1	0	2	<1	0	2	<1
		(2,5,1)	43351	2	227	-	TO	-	-	MO	-	-	MO	-	-	MO
		(4,1,1)	0	2	<1	-	TO	0	2	<1	-	-	MO	0	2	<1
		(4,5,1)	-	-	TO	-	TO	-	-	TO	-	-	MO	-	-	TO
	samplerocks	(1,1)	2476289	2	3623	-	MO	-	-	MO	-	-	MO	-	-	MO
		(2,1)	-	-	MO	-	MO	-	-	MO	-	-	MO	-	-	MO
Maximize	egl	(5,2)	0	2	<1	-	MO	0	2	<1	0	2	<1	0	2	<1
		(6,2)	0	2	<1	-	MO	0	2	<1	0	2	<1	0	2	<1
		(7,2)	0	2	<1	-	MO	0	2	<1	0	2	<1	0	2	<1
	4x4-grid	(2)	57	30	<1	-	MO	571	291	<1	-	-	MO	571	291	< 1
		(4)	-	-	TO	-	MO	-	-	TO	-	-	MO	-	-	TO
	maze	(2)	0	2	<1	-	MO	0	2	<1	-	-	TO	0	2	<1
		(4)	0	2	1	-	MO	0	2	1	-	-	MO	0	2	1
		(64)	0	2	795	-	MO	0	2	914	-	-	MO	0	2	823
	network_dp	(2,1,1)	9	2	<1	-	MO	165	2	<1	-	-	MO	165	84	<1
		(2,3,1)	-	-	MO	-	TO	-	-	MO	-	-	MO	-	-	MO
		(4,1,1)	0	2	<1	-	TO	7	2	<1	-	-	MO	7	5	<1
		(4,3,1)	-	-	MO	-	TO	-	-	MO	-	-	MO	-	-	MO
	network_ps	(2,1,1)	0	2	<1	-	MO	0	2	<1	0	2	<1	0	2	<1
		(2,5,1)	-	-	MO	-	TO	-	-	MO	-	-	MO	-	-	MO
		(4,1,1)	0	2	<1	-	TO	0	2	<1	-	-	MO	0	2	< 1
		(4,5,1)	-	-	TO	-	TO	-	-	TO	-	-	MO	-	-	TO
	samplerocks	(1,1)	13	2	<1	-	MO	921	2	2	-	-	MO	923	477	2
		(2,1)	-	-	MO	-	MO	-	-	MO	-	-	MO	-	-	MO

between 0 and 1, so using absolute precision is suitable. We consider region $[0.1, 0.9]$ for the reachability properties, whereas for the expected total reward properties, we take region $[0.1, 0.4]$. We decided to take this smaller region for expected total reward properties as for these properties, the solution function is often possibly monotonic on this smaller region but not on $[0.1, 0.9]$. Picking this smaller region allows us to discuss the influence of heuristic `h-CurMax`, the heuristic for initializing and updating `CurMax` for expected total reward (see also Remark 5.4). We use a splitting threshold of 1, meaning we split in one direction at a time. We modified the vanilla approach to split in one direction at a time to overcome the explosion in the number of regions. To determine `CurMax` in the vanilla approach, we initially pick the region's center point, as analyzing all vertices yields time-outs or even memory-outs if we first compute all vertices and then try to analyze them. The model characteristics of the pMCs are

Table 5.3: Monotonicity checking results

	benchmark	instance	sufficient	$\lvert V\rvert\uparrow^R$	$\lvert V\rvert\uparrow^{R}?$
Reachability	evade	(1,2,0,1)	9%	2	38
		(1,2,1,1)	7%	4	76
		(1,2,3,1)	6%	8	151
	maze	(25)	100%	24	0
		(1000)	100%	999	0
		(10000)	100%	9999	0
	nrp	(5,1)	100%	1	4
		(10,1)	100%	1	9
		(15,1)	100%	1	14
Expected Total Reward	egl	(5,2)	100%	10	0
		(6,2)	100%	12	0
		(7,2)	100%	14	0
	4x4-grid	(1)	6%	0	3
		(2)	4%	0	7
		(4)	4%	0	25
		(8)	5%	0	53
	maze	(1)	63%	4	11
		(2)	21%	4	27
		(4)	11%	6	121
		(64)	3%	50	3510
	network_dp	(2,1,1)	52%	0	9
		(2,3,1)	48%	18	65
		(4,1,1)	31%	12	24
		(4,3,1)	27%	23	191
	network_ps	(2,1,1)	100%	14	0
		(2,5,1)	32%	20	121
		(4,1,1)	36%	9	32
		(4,5,1)	24%	16	284
	samplerocks	(1,1)	39%	1	36
		(2,1)	20%	2	184

found in Table B.1 (page 190) and Table B.2 (page 191) in Appendix B.2.1.

To evaluate our approach, we answer the following question:

Q5.1 How do the heuristics influence the performance of the integrated approach?

Q5.2 How does the integrated approach perform compared to the vanilla approach?

Q5.3 How do the characteristics of the integrated approach influence its performance?

Q5.4 How good are the obtained optima?

5.6.1 Results

Tables 5.1 and 5.2 show the timing results for the considered benchmarks for reachability and expected total reward properties, respectively. The upper half of each table shows the results when minimizing the extremal value, whereas the lower half displays the maximization. For both the vanilla and integrated approach, we consider two values for ε: $\varepsilon = 0.1$ and $\varepsilon = 0.05$. Note that we leave out the results for expected total reward properties on the vanilla approach for $\varepsilon = 0.05$, as for $\varepsilon = 0.1$, all instances already have time and memory-outs. For the integrated approach with $\varepsilon = 0.05$, we

Table 5.4: Computed optima for vanilla and integrated parameter lifting

			ε: 0.1				ε: 0.05					
			integrated		vanilla		heuristic		integrated no h-CurMax		no h-order	
	name	**instance**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**
Reachability	evade	(1,2,0,1)	0.007	0.991			0.007	0.991			0.007	0.991
		(1,2,1,1)	0.007	0.991	-		0.007	0.991	-		0.007	0.991
		(1,2,3,1)	0.007	0.991			0.007	0.991			0.007	0.991
	maze	(25)	0.009	0.145	0.077	0.077	0.009	0.145	0.009	0.145	0.009	0.145
		(1000)	0.009	0.145	0.077	0.077	0.009	0.145	0.009	0.145	0.009	0.145
		(10000)	0.009	0.145	0.077	0.077	0.009	0.145	0.009	0.145	0.009	0.145
	nrp	(5,1)	0.082	0.200	-		0.082	0.200			0.082	0.200
		(10,1)	0.065	0.100	0.100	-	0.065	0.100	-		0.065	0.100
		(15,1)	0.053	-	0.067	-	0.053	-			-	
Expected Total Reward	egl	(5,2)	6.667e-1	1.546e0			6.667e-1	1.546e0	6.667e-1	1.546e0	6.667e-1	1.546e0
		(6,2)	6.667e-1	1.523e0	-		6.667e-1	1.523e0	6.667e-1	1.523e0	6.667e-1	1.523e0
		(7,2)	6.667e-1	1.511e0			6.667e-1	1.511e0	6.667e-1	1.511e0	6.667e-1	1.511e0
	4x4-grid	(2)	4.026e1	1.069e4	-		4.026e1	1.069e4	-		4.026e1	1.069e4
		(4)	4.419e1	-			-				-	
	maze	(2)	7.059e1	1.545e6			7.059e1	1.545e6			7.059e1	1.545e6
		(4)	-	1.861e6	-		-	1.861e6	-		-	1.861e6
		(64)	-	2.171e6			-	2.171e6			-	2.171e6
	network_dp	(2,1,1)	2.789e0	4.769e0	-		2.789e0	4.769e0	-		2.789e0	4.769e0
		(4,1,1)	1.276e0	3.800e0			1.276e0	3.800e0			1.276e0	3.800e0
	network_ps	(2,1,1)	2.610e-1	4.569e-1			2.610e-1	4.569e-1	2.610e-1	4.569e-1	2.610e-1	4.569e-1
		(2,5,1)	3.939e0	-	-		-				-	
		(4,1,1)	1.766e0	2.029e1			1.766e0	2.029e1	-		1.766e0	2.029e1
	samplerocks	(1,1)	7.755e0	3.904e1	-		-	3.911e1	-		-	3.911e1

additionally show results without heuristic `h-CurMax` and without heuristic `h-order`. Recall from Remark 5.4 that `h-CurMax` refers to the improved picking of `CurMax`, whereas `h-order` refers to stopping the exploration of the order if it is sufficient for at least 25% of the states. For each ε, we consider the time **t** required for the divide-and-conquer approach and the number of parameter lifting invocations (**# i**) that the loop in the integrated and vanilla approach required. Furthermore, for the integrated approach, we provided the number (**# $\mathbf{i_b}$**) of extra parameter lifting invocations needed to assist the monotonicity checker. Time-outs and memory-outs are denoted by TO and MO, respectively. If the number of parameter lifting invocations is not available, we denote this by -.

Table 5.3 shows the results of the initial monotonicity checking in the integrated approach. It lists the benchmarks and instances and shows, for each instance, the percentage of states for which the initial order, i.e., before splitting the regions, is sufficient (**sufficient**). Furthermore, Table 5.3 lists the number of parameters for which monotonicity was found by the integrated approach ($|V|\uparrow^R$) and the number of possibly monotonic parameters ($|V|\uparrow^R$?); the remaining parameters are not monotonic on the initial region. Compared to the other tables in this section, we leave out the optimization direction (**dir**) and precision (ε), as these do not influence the initial monotonicity checking. Also, we leave out the results using the integrated approach with less heuristics, as the initial monotonicity checking in all approaches is the same.

Table 5.4 shows the optimal values (**min**, **max**) obtained by the vanilla and integ-

rated divide-and-conquer approach. Furthermore, Table 5.4 shows the optimal values obtained using only one of the two heuristics. We leave out the optimal values for the vanilla approach with a precision of $\varepsilon = 0.05$, as all benchmarks either had a time-out or memory-out. If the extremal value is not available, we denote this by -.

5.6.2 Discussion

We first elaborate on some inconsistencies in the results caused by the implementation and running the benchmarks in parallel on the cluster. Secondly, we discuss the results of the vanilla and integrated approach as displayed in Tables 5.1 and 5.2 and elaborate on the results using Tables 5.3 and 5.4. We then discuss the characteristics of the integrated approach. Next, we discuss the obtained optima for the vanilla and integrated approach and examine their similarities and differences. Finally, we discuss the influence of the heuristics on the run-time, number of parameter lifting invocations, and obtained optima.

Influence of the setup | Before answering the questions (Q5.1 through Q5.4), we discuss several results caused by the setup, i.e., the implementation in Storm and running the benchmarks in parallel on the high-performance cluster. First of all, we observe that for several benchmarks, we have 0 parameter lifting invocations for the loop (# $\mathbf{i}$) and 2 invocations (# $\mathbf{i_b}$) for the bounds for the monotonicity checking. As the parameter lifter initially only has trivial bounds, we start with 2 invocations: one for U and one for L. We reuse the result of U for the initial region to check against `CurMax`. Therefore, # $\mathbf{i}$ is 0, whereas # $\mathbf{i_b}$ is 2. After this initial region, we reuse parameter lifting results; thus, in each loop, # $\mathbf{i}$ and # $\mathbf{i_b}$ increase with at most 1. Secondly, we observe that when maximizing samplerocks (1,1), we have 921 parameter lifting invocations when we use all heuristics and 923 invocations when we do not use `h-order`. This can be explained by the order of the regions in the queue, as this is not deterministic. They are sorted on their bounds, and if they have the same bound, they are sorted on their age. However, if two regions have the same age, they can have any order in the queue. Thus, we might need to do parameter lifting for some more regions. Finally, we see timing differences, e.g., at the reachability properties for maze on $\varepsilon = 0.05$ timing differences. This is due to the parallelization on the cluster and cannot be avoided.

Influence of the heuristics | First of all, we observe that for all instances of maze for the reachability property and instance (2,1,1) of network_ps, the heuristics have no influence, as all parameters are globally monotonic (cf. Table 5.3 and Tables B.1 and B.2 in Appendix B.2.1).

Next, we consider the case where we do not use heuristic `h-CurMax`. Recall from Section 3.6 that nrp is monotonic in all parameters. However, our approach can only

find one monotonic parameter. The influence of `h-CurMax` is, in particular, evident at this benchmark. Without `h-CurMax`, we cannot solve the nrp instances, whereas we find a solution for almost all instances, with `h-CurMax`. Furthermore, we observe for egl that we need more parameter lifting invocations to find a suitable `CurMax`. In general, we get time and memory-outs for most benchmarks, although we do find more extremal values compared to the vanilla approach. For those where we can find extremal values, they are similar to the ones with both heuristics. This can be explained by the fact that we have several globally monotonic parameters which are set to their bounds.

Finally, we consider the case where we do not use heuristic `h-order` but do use `h-CurMax`. In order to discuss the influence of `h-order`, we look at the benchmarks where the initial order is not sufficient for all states but is sufficient for at least 25% of the states. Furthermore, we restrict ourselves to the benchmarks described above, where we additionally need several region splits; as for those benchmarks, we compute new reward orders (and monotonicity results). So we only consider expected total reward properties of maze, network_dp, and samplerocks. We see for all of the benchmarks that the timing results are more or less equivalent. Those benchmarks with a memory or time-out while using the heuristics also have a memory or time-out while not using `h-order`. When analyzing the results for those benchmarks where we did not get an error, we see that the number of additional parameter lifting invocations ($\#\,\mathbf{i_b}$) is higher than this number while using both heuristics. This is as expected, as this is the reason why we introduced `h-order`.

Furthermore, we observe from Table 5.4 that if this approach finds an extremal value, this value is similar to the value using both heuristics. So even though we continue to extend the order and thus have more parameters for which we know monotonicity, this does not influence `CurMax` as the initial guess of `CurMax` was already close to optimal.

Answer to Q5.1 (How do the heuristics influence the performance of the integrated approach?)

- For models with only globally monotonic parameters, the heuristics have no influence.
- Heuristic `h-CurMax` allows us to solve more benchmark instances.
- Heuristic `h-order` reduces the number of additional parameter invocations.

Vanilla vs. integrated | When comparing the vanilla and integrated approach, we observe that the integrated approach often outperforms the vanilla approach. For an increased precision (i.e., $\varepsilon = 0.05$), the vanilla approach even has a memory or

time-out for all benchmarks. In contrast, the integrated approach can solve many of the given benchmarks. The only benchmark where the vanilla approach outperforms the integrated approach is the maze benchmark with a reachability property for precision $\varepsilon = 0.1$. This is due to the overhead caused by the monotonicity checking. When we consider the obtained optima (Table 5.4), we observe that the results for the integrated approach are closer to the real optima than the results for the vanilla approach. Moreover, when we consider Table 5.3, we see that the optima for, e.g., the reachability property for the maze benchmark are the provably optimal values as all parameters are monotonic (cf. Table B.1 in Appendix B.2.1). We can conclude that even though the integrated approach performs slower, the resulting optima are more precise. Next, we discuss the time-outs and memory-outs in Tables 5.1 and 5.2. The splitting of the regions causes the memory-outs. Although the integrated approach 1) yields a better `CurMax`, 2) splits based on the region split estimates, 3) only splits in parameters that are not known to be globally monotonic, and 4) splits in one direction at a time, it may still happen that we need to split the region several times. The time-outs are caused by the time it takes to solve the associated MDP. We observe several time-outs for expected total reward properties, whereas, for reachability properties, we only have memory-outs. This can be explained by the MDP solver needing more time for expected total reward properties than reachability properties.

Answer to Q5.2 (How does the integrated approach perform compared to the vanilla approach?)

- The integrated approach outperforms the vanilla approach on (most) benchmarks.
- The results for the integrated approach are more precise than the results for the vanilla approach.

Characteristics of the integrated approach | Now we discuss the influence of the several characteristics of the integrated approach. First of all, we observe that the percentage of states for which an order is sufficient (see Table 5.3) does not directly influence the number of monotonic parameters. The explanation for this is twofold:

1. Even though an order can be sufficient for parameter p, this does not imply global monotonicity. If, for one state, p is locally monotonic increasing and for another, p is locally monotonic decreasing, then global monotonicity cannot be obtained. Consider nrp, for all instances, we find a sufficient order. However, we only obtain monotonicity for one parameter.

2. If a parameter occurs at only a few states, the chances that the order is sufficient

for that parameter are higher. Also, if a parameter only occurs at a few states, the chances that the parameter is only locally monotonic increasing (or only decreasing) are higher. This is easy to see for a parameter occurring at exactly one state. If the order is sufficient for the state (and the transition function is monotonic), chances are high that we obtain local (and thus global) monotonicity. Consider, e.g., network_dp, we observe that a parameter occurs on average at 10.9 and 6.6 states for instances (2,1,1) and (2,3,1), respectively. Note that this is the number of states divided by the number of parameters. The order for instance (2,1,1) has a higher sufficiency than the order for instance (2,3,1), i.e., 52% and 48%, respectively. However, for instance (2,3,1), global monotonicity for 18 parameters is found, whereas no global monotonicity is found for instance (2,1,1).

Secondly, we consider the influence of the global monotonic parameters. From Table 5.3, we observe that for the reachability property of the instances of maze, the expected total reward property of all instances of egl, and the (2,1,1) instance of network_ps, we have global monotonicity in all parameters. As already discussed, this yields provably optimal results. Next, we consider the influence of several parameters being possibly monotonic. E.g., for both $\varepsilon = 0.1$ and $\varepsilon = 0.05$, we see that guessing `CurMax` such that it is an (almost) optimal solution reduces the number of splits we need for a region. Consider, e.g., nrp, we observe that as we guess `CurMax` close to the real optimum, we can find a solution, whereas we often obtain a memory-out for the vanilla approach (where we pick `CurMax` at the center of the region). Also, for most benchmarks, we do not update `CurMax` once it is initialized, as our initial guess is already (almost) optimal. We further discuss this when discussing the obtained optima.

Answer to Q5.3 (How do the characteristics of the integrated approach influence its performance?)

- The percentage of states for which an order is sufficient does not directly influence the number of monotonic parameters.
- Global and possible monotonic parameters improve the initial value of `CurMax`.

Obtained optima | Now, we consider the obtained optima for the vanilla approach and the integrated approach with heuristics as depicted in Table 5.4. Recall that the results for the vanilla approach for $\varepsilon = 0.05$ are omitted, as all benchmarks had time and memory-outs. Thus no results are available. We see for the reachability properties for maze and nrp that the vanilla approach can find a solution for $\varepsilon = 0.1$. However, as we use absolute precision, the results are far from those of the integrated approach.

Moreover, for both minimization and maximization, the same value is found. Our modification of the initial `CurMax` of the vanilla approach can explain this. As sampling over all vertices is infeasible for more parameters, we pick the center point, which happens to be ε close to the real optimal value using absolute precision. If we compare the results for the integrated approach, we observe that the values found do not differ for almost all benchmarks between $\varepsilon = 0.1$ and $\varepsilon = 0.05$. We can explain this by the initial guess of `CurMax` being (almost) optimal. Thus we do not update `CurMax` (see also Figure 5.5 step (6)). We see that the obtained maximum only differs for samplerocks at instance (1,1). This instance has 189 parameters, of which 2 are globally monotonic and 184 are possibly monotonic, leaving 3 parameters being definitely not monotonic. The fact that we update `CurMax` confirms that at least three parameters are not monotonic.

Answer to Q5.4 (How good are the obtained optima?)

- For globally monotonic parameters, the result is provably optimal.
- Using a higher precision for the integrated approach does not always improve the result. Thus the result is close to optimal.

5.7 Conclusion

This chapter showed how parameter lifting and the vanilla divide-and-conquer approach can benefit from monotonicity checking. We have shown that with the help of monotonicity, we can tackle the ε-optimal synthesis problem for both reachability and expected total reward properties. The experiments showed encouraging results: where we could only solve a few instances for reachability properties with the vanilla approach, the integration with monotonicity checking allows us to check models with hundreds of parameters. Furthermore, the discussed heuristics have shown their worth. Future work includes improving the monotonicity checking, as discussed in Section 3.7, and extending the integrated approach to pMDPs.

6 Gradient Descent

Summary | In this chapter, we describe how we can use the optimization method gradient descent (GD) to solve the feasibility problem, as described in Section 2.4. The main idea of GD is to update the value of a parameter based on its gradient and a given learning rate. First of all, in Section 6.2, we introduce the basic idea of GD and explain how we can use GD to solve the feasibility problem. Secondly, in Sections 6.3 and 6.4, we discuss several classical (plain GD, momentum GD, Nesterov-accelerated GD) and adaptive (RMSProp, Adam, RAdam) GD methods. Furthermore, we consider the classical GD methods, where we only respect the sign of the gradient. In Section 6.5, we discuss several region restriction methods to ensure that the obtained parameter values are within a given region. Section 6.6 introduces an equation-based approach to evaluate the gradient at a given instantiation efficiently. Furthermore, it shows an alternative approach to obtain the derivative of the solution function for the expected total reward by first taking the derivative of a pMC (yielding a weighted automaton) and then solving the standard equation system. Finally, we study the use of GD to solve the feasibility problem. The benchmarks are pMCs obtained from partially observable Markov decision processes (POMDPs), which are transformed into pMCs as described in Appendix C. First, we evaluate the different region restriction methods and different GD methods. Then we compare the best combination to the baseline methods particle swarm optimization (PSO) [CHHK+13], quadratically-constrained quadratic programming (QCQP) [CJJK+18], and sequential convex programming (SCP) [CJJK+22]. Moreover, we answer the following questions:

Q1 Which region restriction method works best?

Q2 Which GD method works best?

Q3 How does GD compare to the baseline methods PSO, QCQP, and SCP?

Origins | This chapter is an extended version of the results presented in [4]. All GD update methods and region restriction methods originate from the literature. The structure of [4] is based on the recent overview given by Ruder [Rud16]. The GD updates methods we describe are momentum GD [Rum89], Nesterov-accelerated GD [Nes83; SMDH13], RMSProp [TH12], Adam [KB15], and RAdam [LJHC+20]). Furthermore, we compare GD to PSO, QCQP, and SCP as introduced in [CHHK+13], [CJJK+18], and [CJJK+22], respectively.

Background | We consider pMCs as introduced in Section 2.2. Furthermore, we consider the feasibility problem as introduced in Section 2.4:

Given a pMC $\mathcal{D}$, a comparison operator $\bowtie \in \{<, \leq, \geq, >\}$, a threshold $\lambda \in [0, \infty)$, and a graph-preserving region R, does there exist a $\vec{u} \in R$ such that $\mathsf{ER}_{\mathcal{D}}^{s \to T}(\vec{u}) \bowtie \lambda$?

Preliminary knowledge on gradient descent in given in Section 6.2.

6.1 Related Work

Solving the feasibility problem for Markov chains | As discussed in Section 5.1 several methods exist to tackle the feasibility problem for pMCs. In this chapter, we consider particle swarm optimization (PSO) [CHHK+13], quadratically-constrained quadratic programming (QCQP) [CJJK+18], and sequential convex programming (SCP) [CJJK+22]. Where PSO is a search method, QCQP and SCP take an iterative approach in which the encoding is simplified around a point to guide the search.

Using GD on Markov models | The use of GD methods to find finite state controllers for partially observable systems was explored, among others, by Meuleau et al. [MPKK99]. Their approach has further developed into deep learning for POMDPs, which is, e.g., used to learn Atari games [MKSG+13]. Some of the developed approaches allow the extraction of finite-state controllers [CJT20]. These methods learn the controllers from demonstrations or provided traces and are thus model-free. Closest to our approach, where we analyze pMCs (obtained from POMDPs) with the help of GD, is the work by Aberdeen [Abe03] in using a model-based approach to find memoryless strategies in POMDPs via GD. Also Meuleau et al. [MKKC99], try to find memoryless or small-meory controllers in POMDPs using GD. However, the approach back then could and did not compare to the current state-of-the-art methods. Other approaches using GD and MCs are, e.g., [SSY18], where the random samples for GD are taken on the trajectory of an MC), and [DDB20], which use MC tools to improve GD analysis.

On GD, there exists lots of research. The GD methods can be divided into classical and adaptive methods. The classical methods have a fixed learning rate, whereas, for adaptive methods, this learning rate is changed over time and depends on the parameter. The GD updates methods we describe are momentum GD [Rum89], Nesterov-accelerated GD [Nes83; SMDH13], RMSProp [TH12], Adam [KB15], and RAdam [LJHC+20]). Other GD methods include, e.g., Adagrad [DHS11] and its extension Adadelta [Zei12] and AdaMax [KB15], and the methods QHM [MY18] and Demon [CKWL19].

Analyzing partially observable Markov decision processes | Partially observable MDPs (POMDPs) [KLC98] are the standard model for controller synthesis under partial observability. Unfortunately, synthesizing a controller for a quantitative

Algorithm 8 Basic Idea of GD

1: pick $\vec{u}$
2: **while** $\mathsf{ER}^{s_I \to T}(\vec{u}) < \lambda$ **do**
3: **if** $\vec{u}$ is a local optimum **then**
4: pick new $\vec{u}$
5: update $\vec{u}$ with GD method
6: **return** $\vec{u}$

objective is undecidable [MHC03]. Several techniques exist for finding optimal values in partially observable MDPs (POMDPs), including belief-MDPs [Lov91; WS17; HBC18; BJKQ20; BKQ22], dynamic programming [Bon98], approximate value iteration [Hau00], optimization and search techniques [MKKC99; BB04; ABZ10], machine learning [DPR08; CJWS+19; CJT20], Monte Carlo simulation [SV10], and game-based abstraction [WJWJ+17]. If the memory structure and size is known, the main approaches are to use satisfiability checking and SMT solving [CCD16; WCK18] or to transform the POMDP to a pMC and use parameter synthesis [JJWQ+18; Jun20] (see also Appendix C).

Breakthroughs in point-based solvers [PGT03; SV05] and Monte-Carlo methods for finding solutions [SV10] heavily influence the use of belief-MDPs. The most recent support in the probabilistic model checkers PRISM [NPZ17] and Storm [BJKQ20] is based on an abstraction of the belief-MDP and abstraction refinement. The use of game-based abstraction [WJWJ+21] leads to non-randomized controllers. Winterer et al. [WWJB20] support a finite set of uniform randomizations, whereas the POMDP to pMC approach considers an infinite combination of possibilities. Andriushchenko et al. [ACJK21] consider syntax-guided synthesis for partial information controllers with a finite set of options.

6.2 Background

gradient descent

Gradient descent (GD) is an optimization method that uses the first derivative to optimize an objective function $f(\vec{u})$. It does so by updating the parameters in the direction of their gradients, i.e., parameter p gets update in the direction of $\frac{\partial}{\partial p} f(\vec{u})$. A GD method determines how the parameters are updated based on this gradient. Algorithm 8 shows the algorithm to find a feasible solution for comparison operator $\sim = \geq$, a given λ, and any GD method. The main advantage of GD over calculating the solution function for the expected total reward (see also Section 2.4) is that we do not need to analyze the function itself (which is intractable, see Section 3.4). Instead, we pick an instantiation $\vec{u}$, at which we calculate $\mathsf{ER}^{s_I \to T}(\vec{u})$ (line 1). If $\mathsf{ER}^{s_I \to T}(\vec{u})$ is strictly smaller than λ, we

are done. Otherwise, we update $\vec{u}$ by analyzing $\frac{\partial}{\partial p}\mathsf{ER}^{s_I \to T}(\vec{u})$ (line 5) for a given GD method. As the function $\mathsf{ER}^{s_I \to T}(\vec{u})$ might not be convex, we have to deal with local optima. If the updated $\vec{u}$ is a local optimum, we pick a new $\vec{u}$ (line 4).

6.2.1 GD Methods

Over the years, several GD update methods have been developed. All GD methods have a learning rate that determines how we update the parameter. In general, we can split the GD methods into two categories. The first category consists of the classical GD methods. These methods have a fixed learning rate, i.e., the learning rate does not change over time. In Section 6.3, we consider the following classical GD methods: plain GD, momentum GD [Rum89], and Nesterov-accelerated GD [Nes83; SMDH13]. For these classical methods, we also consider the variant in which we only respect the sign of the gradient and not its magnitude. The second category of GD methods consists of adaptive GD methods. The learning rate changes over time for these methods, and each parameter has its own learning rate. In Section 6.4, we describe the following adaptive GD methods: RMSProp [TH12], Adam [KB15], and RAdam [LJHC+20]). The GD methods cannot ensure that the parameter instantiation stays within a given region after an update. Therefore, the GD method is often extended by a region restriction method. In Section 6.5, we describe the following region restriction methods: projection, penalty function [SCBF+97], barrier function [Van98], and logistic function [HM95].

6.2.2 Number of parameters to update

Next to determining the GD method, we also have to determine the number of parameters we update at each iteration. There are three common variants to update the parameters: batch GD, stochastic GD, and mini-batch GD. *Batch GD* computes the gradient of $f(\vec{u})$ w.r.t. all parameters, thus for each parameter p of $f(\vec{u})$, we analyze $\frac{\partial}{\partial p} f(\vec{u})$ and update p in $\vec{u}$ according to the GD method. In contrast, *stochastic GD* updates the parameters in a round-robin fashion, i.e., at iteration t we update the instantiation $\vec{u}$ at parameter $p_{i(t)}$ with $i(t) = t \mod |V|$, while $\vec{u}$ at the other parameters remain the same. *Mini-batch GD* performs an update for a subset of parameters.

batch GD

stochastic GD

mini-batch GD

> **Remark 6.1** In the following, we describe the GD update methods w.r.t. stochastic GD. Note that stochastic GD can be extended to mini-batch GD and batch GD, by updating more/all parameters at the same time.

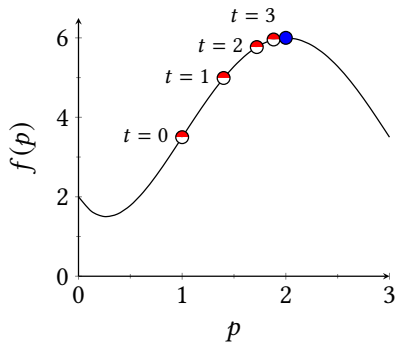


Figure 6.1: Plain GD on f for $R = [0, 3]$

Remark 6.2 We describe the GD methods and region restriction methods in Sections 6.3 through 6.5 on any differentiable rational function $f(\vec{u})$. Clearly, this can be applied to $\mathsf{ER}^{s_I \to T}(\vec{u})$ for a graph-preserving region R, as $\mathsf{ER}^{s_I \to T}(\vec{u})$ is continuously differentiable on R [QDJJ+16].

6.3 Classical GD Methods

In this section, we discuss *classical GD* methods, which have a fixed learning rate at which the parameters are updated. Furthermore, we discuss the Sign-version of the methods, in which we do not take the magnitude of the gradients into account but only their signs.

·classical

6.3.1 Plain GD

Plain GD is the simplest type of GD. To determine the step size to reach a (local) optimum, a *learning rate* η is used. This learning rate is the same for every parameter and does not change over time, therefore, we refer to it as *fixed learning rate*. For iteration t, we let $\vec{u}_i^t = \vec{u}^t(p_i)$ denote the value of p_i at instantiation $\vec{u}^t$. As we consider stochastic GD, we update parameter p_i, with $i = t \mod |V|$, in $\vec{u}$ based on $\frac{\partial}{\partial p_i} f(\vec{u})$ as follows:

plain GD
learning rate
·fixed

$$\vec{u}_i^{t+1} = \vec{u}_i^t + \eta \cdot \frac{\partial}{\partial p_i} f(\vec{u}^t) \,. \tag{6.1}$$

Example 6.1 Consider $f(p) = \frac{1}{2}p^4 - 4p^3 + 9p^2 - 4p + 2$ on region $R = [0, 3]$ with $\lambda = 5.9$. We apply plain GD with a learning rate η of 0.1.

t	p	$f(p)$	$\frac{\partial}{\partial p} f(p)$
0	1	3.5	4
1	1.4	4.985	3.168
2	1.717	5.726	1.654
3	1.882	5.958	-

The table above shows how the value of p changes at each iteration. Initially, at $t = 0$, we pick $p = 1$. As $\frac{\partial}{\partial p} f(p) = 2p^3 - 12p^2 + 18p - 4$, we obtain for $p = 1$ a gradient of 4, and update p to 1.4. For $t = 1$, the gradient is 3.168, increasing p again. This scheme is repeated until at $t = 3$, we have $f(p) = 5.958$. As this value exceeds λ, a feasible instantiation ($p = 1.882$) is found. Figure 6.1 visualizes the above. The red half-dots indicate the computed value at each iteration. The blue dot indicates the optimum.

6.3.2 Momentum GD

momentum GD
average decay

Where plain GD only considers the current derivative, *momentum GD* [Rum89] also considers previous updates (Equation (6.2)). They are weighted by *average decay factor* $\beta_1 \in [0, 1)$, which is typically set to at least 0.9. To keep track of the previous updates, an additional update vector $\vec{v}$ is introduced. Momentum GD adjusts the parameter value according to the equations below:

$$\vec{v}_i^{t+1} = \beta_1 \cdot \vec{v}_i^t + \eta \cdot \frac{\partial}{\partial p_i} f(\vec{u}^t) \tag{6.2}$$

$$\vec{u}_i^{t+1} = \vec{u}_i^t + \vec{v}_i^{t+1} \tag{6.3}$$

Note that, if $\beta_1 = 0$, momentum GD is equal to plain GD.

Example 6.2 Reconsider the feasibility problem from Example 6.1. We apply momentum GD with average decay factor β_1 of 0.9. As in Example 6.1, we have a learning rate η of 0.1 and we start at $t = 0$ with $p = 1$.

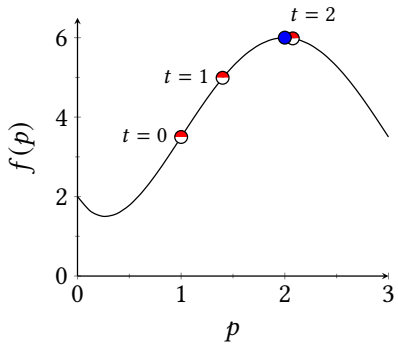


Figure 6.2: Momentum GD on f for $R = [0, 3]$

t	$\vec{v}(p)$	p	$f(p)$	$\frac{\partial}{\partial p} f(p)$
0	0	1	3.5	4
1	0.4	1.4	4.985	3.168
2	0.677	2.077	5.982	-

The table above shows how p is updated and provides the values $\vec{v}(p)$. Figure 6.2 visualizes the changes of p. At $t = 0$, the gradient is 4 and $\vec{v}^0 = 0$, as in Example 6.1 we update p to 1.4. Furthermore, we set $\vec{v}^1$ to 0.4. For $t = 1$, the gradient is 3.168, as $\vec{v}^1 = 0.4$, we set $\vec{v}(p)$ to $0.9 \cdot 0.4 + 0.1 \cdot 3.168$ and increase p by this. At $t = 2$ we have $f(\vec{u}^t) = 5.982$. As this value exceeds λ, a feasible instantiation ($p = 2.077$) is found. Although we overstepped the optimum (the red half-dot is right of the blue dot), we need one step less than plain GD.

6.3.3 Nesterov-Accelerated GD

The main flaw of momentum GD is that it easily moves to the other side of the optimum. To prevent this, *Nesterov-accelerated GD* [Nes83; SMDH13] extends momentum GD by attempting to predict the future by guessing the next instantiation of $\vec{u}$ (denoted $\vec{u'}$, see Equation (6.4)). We then take the derivative at $\vec{u'}$. As for momentum GD, the instantiation is updated according to Equation (6.3), whereas the update vector is obtained as in Equation (6.5):

Nesterov-accelerated GD

$$\vec{u'}_j^{\,t} = \begin{cases} \vec{u}_j^t + \beta_1 \cdot \vec{v}_j^t & \text{if } j = i \\ \vec{u}_j^t & \text{otherwise} \end{cases} \tag{6.4}$$

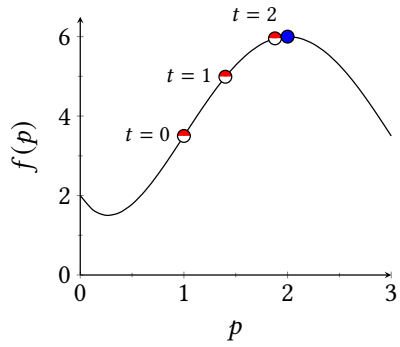


Figure 6.3: Nesterov-accelerated GD on f for $R = [0, 3]$

$$\vec{v}_i^{t+1} = \beta_1 \cdot \vec{v}_i^t + \eta \cdot \frac{\partial}{\partial p_i} f(\vec{u'}^t) \tag{6.5}$$

Example 6.3 We apply GD as in Example 6.2, however, we now use Nesterov-accelerated GD.

t	$\vec{v}(p)$	p	$f(p)$	p'	$\frac{\partial}{\partial p} f(p')$
0	0	1	3.5	1	4
1	0.4	1.4	4.985	1.76	1.412
2	0.501	1.901	5.971	-	-

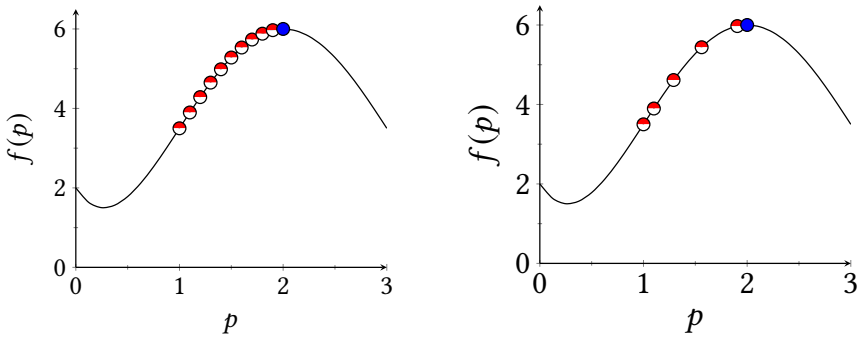
The table above shows p changes at each iteration and provides the values $\vec{v}(p)$. Figure 6.3 visualizes how p changes over time. As for momentum GD, we need one step less compared to plain GD. However, where at momentum GD we overstepped the optimum, this does not occur for Nesterov-accelerated GD.

6.3.4 Sign methods

sign GD

Instead of considering the magnitude of the gradient, the *sign* version of the GD methods described above only respects the signs of the gradients. For plain-Sign GD, we replace Equation (6.1) by:

$$\vec{u}_i^{t+1} = \vec{u}_i^t + \eta \cdot \text{sgn}(\frac{\partial}{\partial p_i} f[\vec{u}^t]) \ .$$



(a) plain-Sign GD (b) Momentum-Sign GD

Figure 6.4: Sign GD on f for $R = [0, 3]$

For momentum-Sign GD, we replace Equation (6.2) by:

$$\vec{v}_i^{t+1} = \beta_1 \cdot \vec{v}_i^t + \eta \cdot \text{sgn}(\frac{\partial}{\partial p_i} f(\vec{u}^t)) \ .$$

For Nesterov-accelerated-Sign GD, we replace Equation (6.5) by:

$$\vec{v}_i^{t+1} = \beta_1 \cdot \vec{v}_i^t + \eta \cdot \text{sgn}(\frac{\partial}{\partial p_i} f(\vec{u'}^t)) \ .$$

Example 6.4 Reconsider Examples 6.1 through 6.3; Figure 6.4a shows how the value of p changes over time using plain-Sign GD. At each iteration, p is updated with 0.1, as the sign of the gradient is positive and we have a learning rate of 0.1. Figure 6.4b shows how the value of p changes over time using momentum-Sign GD. As we only consider the sign of the gradient and do not step over the optimum in momentum-Sign GD, the plot for Nesterov-accelerated-Sign GD is similar to the plot for momentum-Sign GD (see Figure 6.4b).

6.4 Adaptive Methods

In this section, we discuss several *adaptive GD* methods. They have been developed for machine learning purposes [LJHC+20]. The learning rate changes over time, and each parameter has its own learning rate. We refer to this as an *adaptive learning rate*. For all adaptive GD methods, the parameters with larger gradients have lower learning rates

·adaptive

·adaptive

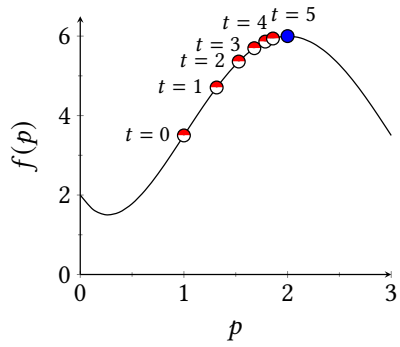


Figure 6.5: RMSProp on f for $R = [0, 3]$

than the ones with smaller gradients. As the learning rates of the adaptive methods depend on the magnitude of the gradients, we do *not* consider adaptive sign methods.

6.4.1 RMSProp

RMSProp

As for momentum GD, *RMSProp* (Root Mean Square Propagation) [TH12] considers the previous updates. However, it considers the past squared gradient, also called the exponential decaying average of past squared gradients. Furthermore, RMSProp adapts the learning rate based on this previous squared gradient (Equation (6.7)). We let $\vec{s}$ denote the update vector for the exponential decaying average of past squared gradients.

$$\vec{s}_i^{\,t+1} = \beta_2 \cdot \vec{s}_i^{\,t} + (1 - \beta_2) \cdot \left(\frac{\partial}{\partial p_i} f(\vec{u}) \right)^2 \tag{6.6}$$

$$\vec{u}_i^{\,t+1} = \vec{u}_i^{\,t} + \frac{\mu}{\sqrt{\vec{s}_i^{\,t+1}} + \epsilon} \cdot \frac{\partial}{\partial p_i} f(\vec{u}) \tag{6.7}$$

Equation (6.6) recursively defines the update vector as the sum of $\beta_2 \in [0, 1)$ multiplied by the past squared gradient and $1 - \beta_2$ times the current squared gradient. β_2 is called the *squared average decay factor*. In Equation (6.7), a small amount $\epsilon > 0$ is added to the update vector for p_i to avoid division by zero.

squared average decay

Example 6.5 Reconsider the feasibility problem from the previous examples in this chapter. We now use RMSProp. As before, let $\eta = 0.1$. Furthermore, let $\beta_2 = 0.9$ and $\epsilon = 10^{-6}$.

t	$\vec{s}(p)$	p	$f(p)$	$\frac{\partial}{\partial p} f(p)$
0	0	1	3.5	4
1	1.6	1.316	4.707	3.463
2	2.639	1.529	5.360	2.615
3	3.059	1.679	5.696	1.860
4	3.100	1.785	5.862	1.272
5	2.952	1.859	5.940	-

The table above shows how p is updated at each iteration and provides the values $\vec{s}(p)$. Figure 6.5 visualizes the updates of p. As before, the red half-dots indicate the computed values, and the blue dot indicates the optimum.
Where we updated p after one iteration to 1.4 for the classical methods, we now get an update to 1.316. This is due to the adapted learning rate, e.g., for $t = 1$, we get $p = 1 + {}^{0.1}/_{\sqrt{1.6}+10^{-6}} \cdot 4 \approx 1.316$.

6.4.2 Adam

Adam (Adaptive Moment Estimation) [KB15] takes both the past squared gradients (like RMSProp) and past gradients (like momentum GD) into account. The equation for $\vec{s}_i^{\,t+1}$ is as in Equation (6.6). For $\vec{v}_i^{\,t+1}$, we have: Adam

$$\vec{v}_i^{\,t+1} = \beta_1 \cdot \vec{v}_i^{\,t} + (1 - \beta_1) \cdot \left(\frac{\partial}{\partial p_i} f(\vec{u}) \right) \tag{6.8}$$

The main disadvantage of this approach is that the method gets biased at the initial time steps if β_1 and β_2 are close to one. This is because initially, $\vec{v}$ and $\vec{s}$ are 0 vectors. Therefore, for Adam, we additionally compute $\hat{v}_i$ and $\hat{s}_i$, where we let β_1^t and β_2^t denote β_1 and β_2 to the power t.

$$\hat{\vec{v}}_i^{\,t} = \frac{\vec{v}_i^{\,t}}{1 - \beta_1^t}, \qquad \hat{\vec{s}}_i^{\,t} = \frac{\vec{s}_i^{\,t}}{1 - \beta_2^t} \tag{6.9}$$

$$\vec{u}_i^{\,t+1} = \vec{u}_i^{\,t} + \frac{\mu}{\sqrt{\hat{\vec{s}}_i^{\,t+1}} + \epsilon} \cdot \hat{\vec{v}}_i^{\,t+1} \tag{6.10}$$

Example 6.6 We apply GD as in Example 6.5, however, we now use Adam.

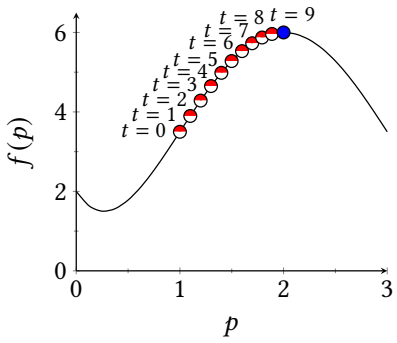


Figure 6.6: Adam on f for $R = [0, 3]$

t	$\vec{v}(p)$	$\vec{s}(p)$	$\hat{\vec{v}}(p)$	$\hat{\vec{s}}(p)$	p	$f(p)$	$\frac{\partial}{\partial p} f(p)$
0	0	0	0	0	1	3.5	4.0
1	0.4	1.6	4	16	1.100	3.898	3.942
2	0.754	2.994	3.969	15.758	1.200	4.285	3.776
3	1.056	4.120	3.898	15.204	1.300	4.650	3.514
4	1.302	4.943	3.786	14.374	1.400	4.984	3.169
5	1.489	5.453	3.636	13.316	1.499	5.280	2.752
6	1.615	5.665	3.447	12.091	1.599	5.530	2.279
7	1.682	5.618	3.223	10.769	1.697	5.728	1.763
8	1.690	5.367	2.967	9.424	1.793	5.873	1.222
9	1.643	4.98	2.682	8.129	1.888	5.962	0.672

The table above shows how p gets updated using Adam and provides the values for the other variables. Figure 6.6 visualizes this result. If we compare Adam to RMSProp, we observe that the steps for Adam are almost constant, whereas for RMSProp p increases a lot at the beginning.

6.4.3 Rectified Adam

As we can see from Example 6.6, we need many iterations using Adam before finding feasible solution. This is due to the variance of the learning rate being too large in the initial steps of the algorithm. *RAdam* (Rectified Adam) [LJHC+20] solves this issue. Liu et al. only use the variance if it is tractable, i.e., the approximated *simple movement*

RAdam

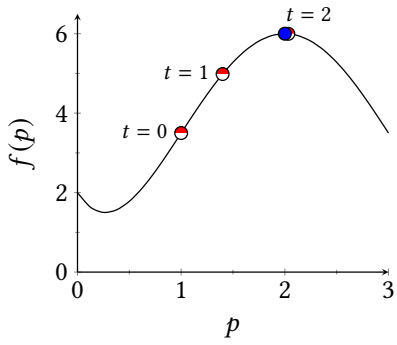


Figure 6.7: RAdam on f for $R = [0, 3]$

average (SMA) is larger than 4. The approximated SMA is calculated as follows:

simple movement average

$$\rho^\infty = \frac{2}{1-\beta_2} - 1, \qquad \rho^t = \rho^\infty - \frac{2t \cdot \beta_2^t}{1-\beta_2^t}$$

$\hat{\vec{s}}_i^t$, $\vec{s}_i^t$, $\hat{\vec{v}}_i^t$, and $\vec{v}_i^t$ are calculated as before. We now update $\vec{u}$ as follows:

$$r^t = \sqrt{\frac{(\rho^t - 4) \cdot (\rho^t - 2) \cdot \rho^\infty}{(\rho^\infty - 4) \cdot (\rho^\infty - 2) \cdot \rho^t}} \tag{6.11}$$

$$\vec{u}_i^{t+1} = \begin{cases} \vec{u}_i^t + \dfrac{\mu}{\sqrt{\hat{\vec{s}}_i^{t+1}} + \epsilon} \cdot r^{t+1} \cdot \hat{\vec{v}}_i^{t+1} & \text{if } \rho^{t+1} > 4 \\ \vec{u}_i + \mu \cdot \hat{\vec{v}}^{t+1} & \text{otherwise} \end{cases} \tag{6.12}$$

Example 6.7 We apply GD as in Example 6.6, however, we now use RAdam.

t	$\vec{v}(p)$	$\vec{s}(p)$	$\hat{\vec{v}}(p)$	$\hat{\vec{s}}(p)$	ρ	r	p	$f(p)$	$\frac{\partial}{\partial p} f(p)$
0	0.0	0.0	0.0	0.0	19.0	1.0	1.0	3.5	4.0
1	0.4	1.6	4.0	16.0	1.0	0.473	1.4	4.985	3.168
2	0.677	2.444	3.562	12.861	1.947	0.064	1.756	5.823	1.434
3	0.752	2.405	2.777	8.874	2.86	0.0	2.034	5.997	−0.203

The table above shows how p changes over time. Furthermore, it shows the values for the other variables. Note that for all iterations, we have $\rho^\infty = 19$. Figure 6.7

visualizes the updates of p. The main difference to Adam (Example 6.6) is that for $f(p)$, we do not use the adapted learning rate, as the approximated SMA is smaller than 4. Thus it is closer to the classical momentum approach. The main difference to momentum GD (Example 6.2) is that we consider the previous gradients and make up for the bias (as explained in Section 6.4.2).

6.5 Dealing with Parameter Regions

In Sections 6.3 and 6.4, we discussed several GD methods. In Examples 6.1 through 6.7, all computed values for p stayed within the region. In general, GD does not necessarily yield parameter values in R. Therefore, we need to deal with those values getting out of R. We discuss suitability for our approach of the following *region restriction methods*: projection, penalty function [SCBF+97], barrier function [Van98] and logistic function [HM95]. We let $R(p_i) = R_i = [\mathsf{l}_i, \mathsf{u}_i]$ denote the bound for parameter p_i in region R.

region restriction method

6.5.1 Projection

projection

The *projection* method is computationally the simplest of all methods. As its name suggests, it projects the instantiation of a parameter p_i back on the bounds of its region, i.e.,

$$\vec{u'}_i^{\,t} = min(max(\vec{u}_i^t, \mathsf{l}_i), \mathsf{u}_i) \ .$$

Example 6.8 Reconsider our running example on $f(p) = \frac{1}{2}p^4 - 4p^3 + 9p^2 - 4p + 2$ and assume we use plain GD with $\eta = 0.1$. We now consider the sub-region $R' = [0.5, 1.5] \subset R$. For $t = 0$, the gradient is 4, and p is updated to 1.4. For $t = 1$, the gradient is 3.17, yielding p to be updated to 1.72. As this is out of the region R', p is projected to 1.5.

Remark 6.3 In the example above for $\lambda = 5.9$, no feasible solution exists; this is no problem as we only want to show how the different region restriction methods work.

6.5.2 Penalty function

penalty function

The *penalty function* method introduces a red warning zone *outside* of the region. If $\vec{u}$ gets out of the region R, a penalty is added to $f(\vec{u})$. This penalty depends on how bad the violation is, e.g., the difference between $\vec{u}_i$ and the bounds of R_i. If $\vec{u}$ is only ε

out of R for parameter p, the instantiation might not be graph-preserving or even not well-defined for a pMC. Therefore, the penalty function approach is unsuitable for GD on pMCs.

6.5.3 Barrier function

barrier function

log-barrier function

The *barrier function* (also called *indicator function*) introduces a red warning zone *inside* of the region. It discourages $\vec{u}_i$ from getting close to its bounds in R_i. We consider the *log-barrier function* for maximizing f as this yields a differentiable function (see Equations (6.13) through (6.17)). The barrier function is weighted by $\mu \in [0, 1]$ and provides us with the equations below.

$$h(\vec{u}^t) = f(\vec{u}^t) + \mu \cdot bar(\vec{u}^t) \tag{6.13}$$

$$\frac{\partial}{\partial p_i} h(\vec{u}^t) = \frac{\partial}{\partial p_i} f(\vec{u}^t) - \mu \cdot \frac{\partial}{\partial p_i} bar(\vec{u}^t) \tag{6.14}$$

where the barrier functions are given by:

$$bar(\vec{u}^t) = \sum_i bar_i(\vec{u}^t) \tag{6.15}$$

$$bar_i(\vec{u}) = \begin{cases} \log(\vec{u}_i - \mathsf{l}_i) & \text{if } \mathsf{l}_i + \frac{\mathsf{u}_i - \mathsf{l}_i}{2} > \vec{u}_i \text{ and } \vec{u}_i \in R_i \\ \log(\mathsf{u}_i - \vec{u}_i) & \text{if } \mathsf{l}_i + \frac{\mathsf{u}_i - \mathsf{l}_i}{2} \leq \vec{u}_i \text{ and } \vec{u}_i \in R_i \\ -\infty & \text{otherwise} \end{cases} \tag{6.16}$$

$$\frac{\partial}{\partial p_i} bar_i(\vec{u}) = \begin{cases} \frac{1}{\vec{u}_i - \mathsf{l}_i} & \text{if } \mathsf{l}_i + \frac{\mathsf{u}_i - \mathsf{l}_i}{2} > \vec{u}_i \text{ and } \vec{u}_i \in R_i \\ \frac{1}{\mathsf{u}_i - \vec{u}_i} & \text{if } \mathsf{l}_i + \frac{\mathsf{u}_i - \mathsf{l}_i}{2} \leq \vec{u}_i \text{ and } \vec{u}_i \in R_i \\ \infty & \text{otherwise} \end{cases} \tag{6.17}$$

Example 6.9 As in Example 6.8, let $f(p) = \frac{1}{2}p^4 - 4p^3 + 9p^2 - 4p + 2$, weight $\mu = 0.1$, and assume we use plain GD. First of all, consider the learning rate $\eta = 0.1$. For this learning rate and the barrier function, as described in this section, we get the following:

t	p	bar	$\frac{\partial}{\partial p} bar$	$h'(p)$	$\frac{\partial}{\partial p} h'(p)$
0	1	−0.301	2	3.470	3.8
1	1.38	−0.921	8.333	4.829	2.410
2	1.621	$-\infty$	∞	-	-

The table above shows how p changes over time using plain GD with a log-barrier function. Observe that for iterations 0 and 1, we have $p \in [0.5, 1.5]$, so the barrier

function is given by $bar = \log(1.5 - p)$. At $t = 0$, the gradient is $4 - \mu \cdot \frac{1}{1.5-p} = 3.8$, so p is updated to 1.38. For $t = 1$, the gradient is 2.410. So p is updated to 1.621, which is outside region R'. Therefore, at $t = 2$ we get $-\infty$ and ∞ for bar and $\frac{\partial}{\partial p} bar$, respectively. We observe that the barrier function is not strong enough for our feasibility problem and learning rate.
Secondly, consider the smaller learning rate $\eta = 0.01$. For this learning rate, plain GD with a log-barrier function converges around $t = 30$ with $p \approx 1.46 \in R'$. Although we never find a feasible solution, we stay within the provided region.

As the example above shows, the barrier function might not be strong enough to prevent $\vec{u}_i$ from getting out of R. Especially if the learning rate is high, this may occur. In Section 6.7, we will see that GD for pMCs profit from a larger learning rate. Therefore, we conclude that the barrier function is less suitable for pMCs.

6.5.4 Logistic function

logistic function

The *logistic function*, maps each parameter p_i to unrestricted parameter q_i by using a sigmoid function [HM95] tailored to R_i. We denote the vector for unrestricted parameters q_i by $\vec{q}$. For $\vec{q}_i = 0$, the sigmoid function maps p_i to $\frac{\mathsf{u}_i + \mathsf{l}_i}{2}$. If the value for q_i goes to infinity (minus infinity), the value of p_i goes to u_i (l_i). Formally, it is defined as follows:

$$\vec{u}_i = \frac{\mathsf{u}_i - \mathsf{l}_i}{1 + e^{-\vec{q}_i}} + \mathsf{l}_i \tag{6.18}$$

It depends on the range of the region R ($\mathsf{u}_i - \mathsf{l}_i$) and the lower bound of the region. The gradient for $\vec{q}$ is computed as follows:

$$\vec{g}_i(\vec{q}) = \frac{e^{\vec{q}_i} \cdot \frac{\partial}{\partial p_i} f(\vec{u})}{(1 + e^{\vec{q}_i})^2}. \tag{6.19}$$

We update $\vec{q}$ according to the chosen GD method.

Example 6.10 As in Example 6.9, let $f(p) = \frac{1}{2}p^4 - 4p^3 + 9p^2 - 4p + 2$ and assume we use plain GD with $\eta = 0.1$. We now consider the logistic function, giving us the following results:

t	q	p	$f(p)$	$\frac{\partial}{\partial p} f(p)$	g
0	0.0	1.0	3.5	4.0	1.0
1	0.1	1.025	3.6	3.996	0.997
2	0.2	1.05	3.699	3.985	0.986
5	0.49	1.12	3.977	3.917	0.923
10	0.92	1.215	4.342	3.742	0.762
20	1.551	1.325	4.737	3.435	0.496
50	2.519	1.425	5.064	3.068	0.212
100	3.254	1.463	5.176	2.913	0.104
200	3.973	1.482	5.23	2.832	0.051
500	4.908	1.493	5.261	2.783	0.02
1000	5.609	1.496	5.271	2.766	0.01

The table above shows how p and q evolve over time. Following Equation (6.18), we have $p = \frac{1.5-0.5}{1+e^{-q}} + 0.5$. As in the previous examples, we start at $p = 1$, so we set q to 0, over time p gets closer to 1.5.

As the logistic function ensures that the resulting parameter instantiation is within the given region, this method is suitable for pMCs.

6.6 Equation-Based Characterization

In Section 3.4, we saw that computing $\mathsf{ER}^{s\rightarrow \{\!\{\}\!\}}(\vec{u})$ by deriving this function symbolically is intractable as the function may be exponentially large in the number of parameters. For the derivative of $\mathsf{ER}^{s\rightarrow \{\!\{\}\!\}}(\vec{u})$, the same holds. In this section, we show that we can efficiently evaluate the gradient of $\mathsf{ER}^{s\rightarrow \{\!\{\}\!\}}(\vec{u})$ w.r.t. a parameter p at a given instantiation $\vec{u}$ by taking the derivative of the equation system that characterizes the expected total reward [BK08, Ch. 10]. This approach is introduced for POMDPs in [Abe03] by Aberdeen. Furthermore, we observe that we can obtain the gradient at $\vec{u}$ by analyzing the equation system for the expected total rewards of a "derived" pMC.

Assumption 6.1 In this section, we assume pMC $\mathcal{D}$ is given.

6.6.1 Equation system

As we have seen in Section 3.4, we can encode pMC-F as an ETR problem, which is an equation system. We extend this for the partial derivative of our expected total reward

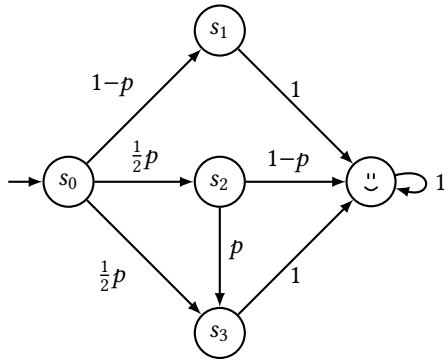


Figure 6.8: An example parametric Markov chain $\mathcal{D}_1$ with $r(s_i) = i$

solution function. In the following, we let x_s denote the value for $\mathsf{ER}^{s\to T}(\vec{u})$, whereas $\partial_p x_s$ refers to the derivative $\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})$.

Definition 6.1 The system of equations for $\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})$ with $p \in V$ is given by:

$$\begin{aligned} x_s &= 0,\ \partial_p x_s = 0 && \text{if } s \in T \\ x_s &= r(s) + \sum_{s' \in S} \mathcal{P}(s,s') \cdot x_{s'} && \text{for } s \in S \setminus T \\ \partial_p x_s &= \sum_{s' \in S} \left(\frac{\partial}{\partial p}\mathcal{P}(s,s') \cdot x_{s'} + \mathcal{P}(s,s') \cdot \partial_p x_{s'}\right) && \text{for } s \in S \setminus T. \end{aligned}$$

Note that in the definition above we obtain the derivative for x_s, i.e., $\partial_p x_s$, by applying the sum rule and the product rule to x_s. This equation system is equivalent to an equation system for POMDPs in [Abe03, p. 47-48]. Note that as the reward at each state is constant, its derivative is 0 (see also Remark 3.1).

Example 6.11 Reconsider our running example pMC $\mathcal{D}_1$, also depicted in Figure 6.8. We obtain the following (simplified) equations for our states:

$$\begin{aligned} x_0 &= (1-p)\cdot x_1 + \frac{1}{2}p \cdot x_2 + \frac{1}{2}p \cdot x_3 & x_1 &= 1 + x_{\text{☹}}, \\ x_2 &= 2 + p \cdot x_3 + (1-p)\cdot x_{\text{☹}}, & x_3 &= 3 + x_{\text{☹}}, \text{ and} \\ x_{\text{☹}} &= 0\,. \end{aligned}$$

Furthermore, we obtain the following equations for the derivatives:

$$\begin{aligned}
\partial_p x_0 &= -x_1 + (1-p)\cdot\partial_p x_1 + \frac{1}{2}\cdot x_2 + \frac{1}{2}p\cdot\partial_p x_2 + \frac{1}{2}\cdot x_3 + \frac{1}{2}p\cdot\partial_p x_3\,,\\
\partial_p x_1 &= \partial_p x_{\circlearrowleft}\,,\\
\partial_p x_2 &= \partial_p x_3 + p\cdot\partial_p x_3 - x_{\circlearrowleft} + (1-p)\cdot\partial_p x_{\circlearrowleft}\,,\\
\partial_p x_3 &= \partial_p x_{\circlearrowleft}\,, \text{ and}\\
\partial_p x_{\circlearrowleft} &= 0\,.
\end{aligned}$$

We observe that we can solve the equations for the states without considering the equations for the derivatives. Solving these state equations yields $x_0 = \frac{3}{2}p^2 + \frac{3}{2}p + 1$, which is the expected total reward function $\mathsf{ER}^{s_0\to\circlearrowleft}(p)$ as also obtained in Example 2.26. Now that we solved the state equations, we can solve the equations for the derivatives. We obtain $\partial_p x_0 = 3p + \frac{3}{2}$, which is the derivative of the expected total reward function, i.e., $\frac{\partial}{\partial p}\mathsf{ER}^{s_0\to\circlearrowleft}(p)$.

As the example above demonstrates, we can first solve the equations for x_s and then solve the equations for $\partial_p x_s$, as the equations for x_s are independent from those for $\partial_p x_s$. For a non-parametric MC, the equations for x_s have a unique solution that coincides with $\mathsf{ER}^s(\Diamond T)$ [BK08, Ch. 10]; this carries over to pMCs [Jun20, Ch. 8] for the solution function $\mathsf{ER}^{s\to T}(\vec{u})$. The following theorem states that the equation system for $\partial_p x_s$ has a unique solution as well and yields the derivative $\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})$.

Theorem 6.1 The equation system of Definition 6.1 has exactly one solution: x_s equals $\mathsf{ER}^{s\to T}(\vec{u})$ and $\partial_p x_s$ equals $\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})$ for each $s \in S$.

Proof. In order to prove Theorem 6.1, we first provide an alternative notation for the equations of Definition 6.1. Secondly, we show that the solution for x_s equals $\mathsf{ER}^{s\to T}(\vec{u})$ (and that this solution is unique). Finally, we show that the solution for $\partial_p x_s$ is unique and equals $\frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u})$.

Alternative notation | Let A be the transition probability matrix based on $\mathbf{P}$ and restricted to $S \setminus \{\circlearrowleft\}$. An alternative way to write the equations in Definition 6.1

for $S \setminus \{\text{␣}\} = \{s_0, \ldots, s_{n-1}\}$ is:

$$\left(1 - \left(\begin{array}{c|c} A & 0 \\ \hline \partial_p A & A \end{array}\right)\right) \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \\ \partial_p x_0 \\ \vdots \\ \partial_p x_{n-1} \end{pmatrix} = \begin{pmatrix} r(s_0) \\ \vdots \\ r(s_{n-1}) \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{6.20}$$

Solution for x_s is unique | We, first of all, observe that by definition of the expected total reward, x_s equals $\mathsf{ER}^{s \to T}(\vec{u})$. Secondly, we obtain its uniqueness as follows: For variables $x_0, \ldots, x_{n-1}$, we have:

$$\begin{aligned} (1 - A) \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} &= \begin{pmatrix} r(s_0) \\ \vdots \\ r(s_{n-1}) \end{pmatrix} \\ \Leftrightarrow \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} &= (1 - A)^{-1} \begin{pmatrix} r(s_0) \\ \vdots \\ r(s_{n-1}) \end{pmatrix} = \begin{pmatrix} \mathsf{ER}^{s_0 \to T}(\vec{u}) \\ \vdots \\ \mathsf{ER}^{s_{n-1} \to T}(\vec{u}) \end{pmatrix} \end{aligned}$$

$(1 - A)^{-1}$ exists because $1 - A$ is invertible (proof in [BK08, p. 821]). Thus the solution is unique.

Solution for $\partial_p x_s$ is unique | We observe that in order to obtain the derivative for x_s, i.e., $\partial_p x_s$, we apply the sum rule and the product rule to x_s. For the variables $\partial_p x_0, \ldots, \partial_p x_{n-1}$, we thus have:

$$\begin{aligned} \begin{pmatrix} \partial_p x_0 \\ \vdots \\ \partial_p x_{n-1} \end{pmatrix} &= (\partial_p A) \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} + A \begin{pmatrix} \partial_p x_0 \\ \vdots \\ \partial_p x_{n-1} \end{pmatrix} \\ \Longleftrightarrow (1 - A) \begin{pmatrix} \partial_p x_0 \\ \vdots \\ \partial_p x_{n-1} \end{pmatrix} &= (\partial_p A) \begin{pmatrix} \mathsf{ER}^{s_0 \to T}(\vec{u}) \\ \vdots \\ \mathsf{ER}^{s_{n-1} \to T}(\vec{u}) \end{pmatrix} \end{aligned}$$

$$\iff \begin{pmatrix} \partial_p x_0 \\ \vdots \\ \partial_p x_{n-1} \end{pmatrix} = (1-A)^{-1} \left((\partial_p A) \begin{pmatrix} \mathsf{ER}^{s_0 \to T}(\vec{u}) \\ \vdots \\ \mathsf{ER}^{s_{n-1} \to T}(\vec{u}) \end{pmatrix} \right)$$

As $(1-A)^{-1}$ exists, there exists a unique solution of the system of equations. We thus obtain that x_s equals $\mathsf{ER}^{s \to T}(\vec{u})$, $\partial_p x_s$ equals $\frac{\partial}{\partial p}\mathsf{ER}^{s \to T}(\vec{u})$, and Equation (6.20) has a unique solution. ■

Complexity | Recall from Section 6.2 that we are interested in computing the gradient at a given $\vec{u}$. We can efficiently solve the equation system by substituting $\mathcal{P}(s,s')$ with $\mathcal{P}(s,s')(\vec{u})$ and solving the equation system for $\frac{\partial}{\partial p}\mathsf{ER}^{s \to T}(\vec{u})$. As the x_s variables can be solved independently, we first solve the x_s-equation system with $|S|$ variables and equations. Yielding the expected total reward $\mathsf{ER}^{s \to T}(\vec{u})$. For every $p \in V$, we construct one equation system. Each of these equation systems has $|S|$ variables and equations as we directly substitute the x_s variables with the expected total reward $\mathsf{ER}^{s \to T}(\vec{u})$ computed before. In total, this means that if we want to apply batch GD, we evaluate $(|V|+1)$ equation systems, with $|S|$ equations and variables each, at each iteration.

6.6.2 Derived parametric Markov chain

We now show that an alternative way to obtain $\frac{\partial}{\partial p}\mathsf{ER}^{s \to T}(\vec{u})$ is by first taking the "derivative" of a pMC $\mathcal{D}$ (called the derived pMC, denoted $\partial_p \mathcal{D}$) and then applying the standard equation system for $\mathsf{ER}^{s \to}_{\partial_p \mathcal{D}}(\vec{u})$. As the derived pMC is not an MC, as the transitions may contain negative values, we generalize pMCs to (parametric) weighted automata [DKV09] and show that we can describe "taking the derivative" as an operation on these weighted automata. The transition functions for weighted automata have *quasi-distributions* as the requirement that $0 \leq \mu(x)[u] \leq 1$ is dropped. However, $\sum_{x \in X} \mu(x)(\vec{u}) = 1$ still holds. The set of quasi-distributions over X is denoted by $\mathsf{QDistr}_{\mathbb{Q}(V)}(X)$.

quasi-distribution

Definition 6.2 A *weighted finite automaton (WFA)* is a tuple $\mathcal{W} = (S, s_I, T, V, \mathbf{Q})$ where S, s_I, T, V are as in Definition 2.3 and $\mathbf{Q} \colon S \to \mathsf{QDistr}_{\mathbb{Q}(V)}(S)$.

weighted finite automaton

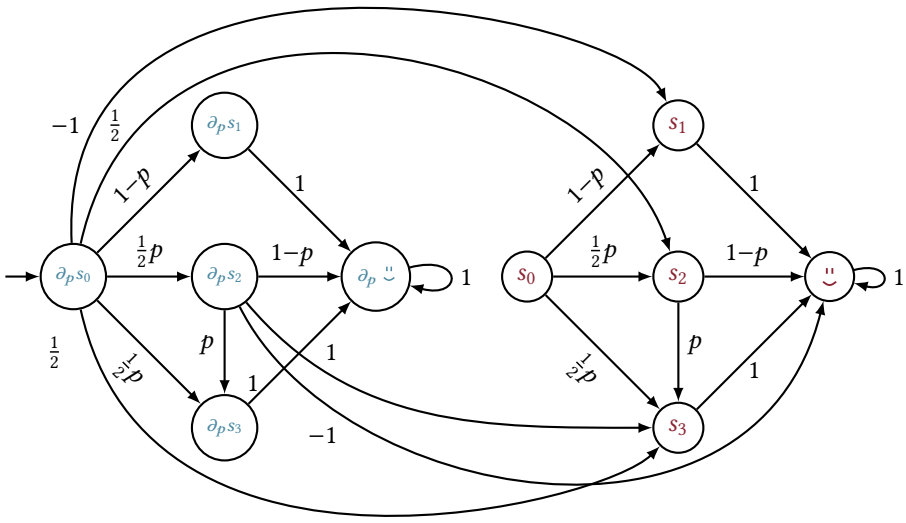


Figure 6.9: Derived pMC $\partial_p \mathcal{D}_1$ of Figure 6.8

Example 6.12 Figure 6.9 depicts WFA $\partial_p \mathcal{D}_1$ with a single parameter p. Note that some of the transitions are labeled with p and $1-p$ (as in Figure 6.8); other transitions are labeled with 1, -1, and $\frac{1}{2}$. We observe that the sum of the weighted transitions is 1, e.g., for state s_0, we have $-1 + \frac{1}{2} + \frac{1}{2} + (1-p) + \frac{1}{2}p + \frac{1}{2} = 1$. We explain the relation of this WFA to pMC $\mathcal{D}_1$ (Figure 6.8) in Example 6.13.

A derived pMC is a weighted automaton that has the derivative as its semantics.

·derived

Definition 6.3 Given pMC $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ with reward function r and $p \in V$. The *derived pMC* of $\mathcal{D}$ w.r.t. p is the WFA $\partial_p \mathcal{D} = (S', \partial_p s_I, T, V, \mathbf{Q})$ where

$$S' = S \dot\cup \partial_p S \text{ with } \partial_p S = \{\, \partial_p s \mid s \in S \,\}, \text{ and}$$

$$\mathbf{Q}(s,t) \;=\; \begin{cases} \mathbf{P}(s,t) & \text{if } s,t \in S, \\ \mathbf{P}(s',t') & \text{if } s,t \in \partial_p S \text{ and } s = \partial_p s' \text{ and } t = \partial_p t', \\ \dfrac{\partial}{\partial p}\mathbf{P}(s',t) & \text{if } s \in \partial_p S \text{ and } s = \partial_p s' \text{ and } t \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, reward function r' is given by $r'(s) = \begin{cases} r(s) & \text{if } s \in S \\ 0 & \text{if } s \in \partial_p S\,. \end{cases}$

The derived pMC is constructed by first copying the existing pMC; then we apply product and sum rules. The derivative for a state $s \in S$ yields the state $\partial_p s$. Using the product rule, we obtain:

$$\begin{aligned} \frac{\partial}{\partial p}\mathsf{ER}^{s\to T}(\vec{u}) &= \frac{\partial}{\partial p}\left(\sum_{s'\in \mathsf{succ}(s)} \mathbf{P}(s,s')\cdot \mathsf{ER}^{s\to T}\right)(\vec{u}) \\ &= \sum_{s'\in \mathsf{succ}(s)} \left(\mathbf{P}(s,s')\cdot \frac{\partial}{\partial p}\mathsf{ER}^{s\to T} + \left(\frac{\partial}{\partial p}\mathbf{P}\,(s,s')\right)\cdot \mathsf{ER}^{s\to T}\right)(\vec{u}) \end{aligned}$$

Thus for every transition $\mathcal{P}(s,s') \neq 0$ with $s,s' \in S$, we add the transitions $\mathcal{P}(\partial_p s, \partial_p s') = \mathcal{P}(s,s')$ and $\mathcal{P}(\partial_p s, s') = \partial_p \mathcal{P}(s,s')$ to $\partial_p \mathcal{D}$.

Example 6.13 Applying Definition 6.3 to the pMC $\mathcal{D}$ from Figure 6.8 yields the derived pMC $\partial_p \mathcal{D}$ in Figure 6.9. We use the same coloring as for the variables of the equations in Section 6.6.1. The transitions from derived states $\partial_p s$ to pMC states s denote the derivatives for the transition functions. E.g., for state $\partial_p s_2$, we have transitions to s_3 and ☹, with weight $\frac{\partial}{\partial p}(1-p) = 1$ and $\frac{\partial}{\partial p}p = 1$, respectively.

As mentioned before, $\partial_p \mathcal{D}$ is not a pMC, as some transitions have negative weights. However, as we restrict ourselves to graph-preserving regions, ensuring that $\mathsf{ER}^{s\to T}_{\mathcal{D}}(\vec{u})$ is continuously differentiable, we can still compute the expected total reward $\mathsf{ER}^{\partial_p s_I \to T}_{\partial_p \mathcal{D}}(\vec{u})$ as in Definition 6.1. The derivative of the expected total reward in $\mathcal{D}$ can now be obtained by computing the expected total reward in $\partial_p \mathcal{D}$.

Proposition 6.2 For each pMC $\mathcal{D}$, we have:

$$\mathsf{ER}^{\partial_p s_I \to T}_{\partial_p \mathcal{D}}(\vec{u}) \;=\; \frac{\partial}{\partial p}\mathsf{ER}^{s_I\to T}_{\mathcal{D}}(\vec{u})\,.$$

The proposition above states that the expected total reward of the derived pMC $\partial_p \mathcal{D}$ equals the partial derivative of the expected total reward of $\mathcal{D}$ w.r.t. parameter p.

6.7 Empirical Evaluation

In this section, we evaluate the different GD methods (Sections 6.3 and 6.4) and the discussed region restriction methods (Section 6.5). We compare the best method to three baselines:

- particle swarm optimization (PSO) [CHHK+13], which is a sampling-based approach,
- quadratically-constrained quadratic programming (QCQP) [CJJK+18], which uses convex optimization methods, and
- sequential convex programming (SCP) [CJJK+22], which produces an approximate linear program instead of a quadratic program as for QCQP.

All methods are present in Prophesy [DJJC+15]. We use the setup for SCP as described in [CJJJ+21]. All methods use the same version of Storm for model building, simplification, model checking, and solving the linear equation systems.

To 1) evaluate the region restriction methods and GD methods and 2) compare the best region restriction/GD method combination to the baseline methods, we answer the following questions:

Q6.1 Which region restriction method works best?

Q6.2 Which GD method works best?

Q6.3 How does GD compare to the baseline methods PSO, QCQP, and SCP?

6.7.1 Set-up

We implemented all GD methods and region restriction methods described in this chapter in the probabilistic model checker Storm [HJKQ+22]. The implementation uses mini-batch GD (Section 6.2.2) w.r.t. k parameters, as this allows for stochastic GD and batch GD by setting k to 1 or $|V|$, respectively. The GD method (Sections 6.3 and 6.4), batch-size k, and region restriction method (Section 6.5) are set via the command line. Also, the GD parameters: learning rate (η), average decay (β_1), and squared average decay (β_2) are configurable via the command line. As described in Section 6.6.1, we create one sparse matrix per parameter and one additional sparse matrix for the expected total reward. We instantiate these matrices at the current $\vec{u}$.

For the experiments, we solve equation systems with GMRES from the `gmm++` linear equation solver library included in Storm, which uses floating-point arithmetic. All

Table 6.1: Number 'easy-parameters' for GD benchmarks

	benchmark	instance	$\lvert V \rvert$	$\lvert V_{\text{easy}} \rvert$
Reachability Probabilites	child	(240)	223	170
	drone	(5,1)	756	667
		(5,2)	2640	404
	4x4-grid	(5)	99	42
	-avoid	(10)	399	158
	hailfinder	(2000)	1249	0
	newgrid	(8,10)	399	244
		(15,10)	399	79
	nrp	(16,2)	95	32
		(16,5)	704	340
	refuel	(5,3)	452	317
		(8,3)	794	570
Expected Reward	4x4-grid	(5)	99	38
		(10)	399	177
	maze	(15)	2624	1257
		(50)	29749	N/A
	network_dp	(8,5,1)	888	537
	network_ps	(8,5,1)	140	128
	samplerocks	(8)	2844	644

experiments run on a single thread and perform some preprocessing (e.g., bisimulation minimization). The times reported are the run-times for GD, PSO, QCQP, and SCP and do not include preprocessing. As for the other experiments (see Appendix B.1), we use machines with an Intel Xeon Platinum 8160 CPU and 32GB of RAM and have a time-out of two hours. For the comparison with PSO, QCQP, and SCP, we report the average run-time over five runs.

Benchmarks | We consider pMC benchmarks with a large number of parameters. We excluded the typical pMC examples [HKPQ+19] with only two or four parameters. The following benchmarks are obtained from POMDPs, as described in Appendix B.2.1, and also considered in Chapters 3 and 5: 4x4-grid, maze, network_dp, network_ps, nrp, and samplerocks. Furthermore, we consider the following pMCs obtained from POMDPs (Appendix C): drone, 4x4-grid-avoid, newgrid, and refuel. Their description is found in Appendix B.3. Finally, we consider pMCs obtained from parametric Bayesian networks [SK20; SK21] with a large number of parameters. We took a medium (child) and large (hailfinder) Bayesian network, both originating from [Scu21] and are parameterized by Salmani and Katoen [SK20; SK21].

The model characteristics of the pMCs are found in Table B.6 (page 197) in Appendix B.3. We observe for some benchmarks (e.g., drone and refuel) that the found instantiation for several parameters is often at the boundary of the region for these parameters. We refer to these parameters as 'easy-parameters'. Table 6.1 shows the number of 'easy-parameters' ($\lvert V_{\text{easy}} \rvert$) for each of the benchmarks. To be able to compare the result, Table 6.1 also contains the number of parameters ($\lvert V \rvert$). The existence

of 'easy-parameters' might imply monotonicity. For benchmark maze at instance (50), the number of 'easy-parameters' is not available, as all runs for the GD methods either timed out or have an internal error. Therefore, no feasible instantiation was found, so the number of 'easy-parameters' could not be determined.

For those benchmarks where the optimal value is not available in the literature, we approximated it by applying the different GD methods several times (until it converges at an instantiation) on different initial instantiations and picking the largest solution found. We checked feasibility against the optimum bounds and the relaxed bounds, where we relaxed all bounds by 10% and 20%, respectively. The plots for 10% are similar to those for 20% and therefore omitted.

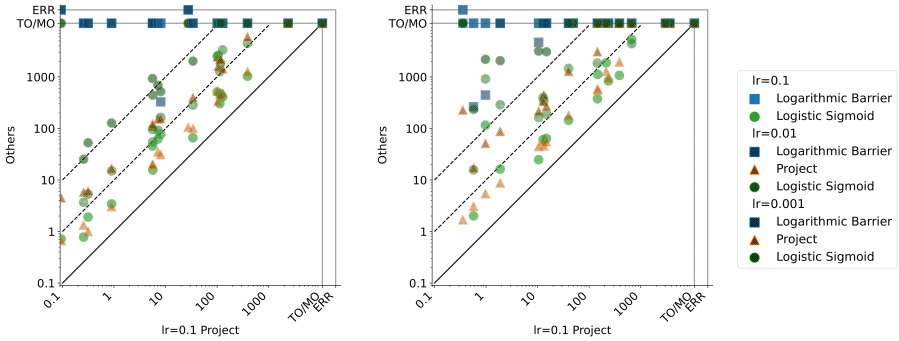
Settings | We fix the region to $R = (0, 1)$. We set the batch size (k) to 32, the average decay (β_1) to 0.9, and the squared average decay (β_2) to 0.999. These are the default values from the literature (see e.g., [TH12; KB15; Rud16; LJHC+20]). In the literature, the learning rate (η) is set between 0.001 and 0.1, we set it to 0.1, as we are interested in finding a feasible instantiation, thus we can take the risk of jumping over a local optimum due to a too high learning rate. Our experiments justify this choice, as lower learning rates slow down finding a feasible instantiation (see Example 6.7 and Section 6.7.3). For the barrier region restriction method, we initially set μ to 0.1. If no feasible solution is found, and we re-pick our instantiation, we divide μ by 10. We continue this procedure until a feasible solution is found, or $\mu < 10^{-6}$.

Initializing $\vec{u}$ and restarting | Initially, we set $\vec{u}$ at the region's center. To overcome possible saddle points at this center, we add $\varepsilon = 10^{-6}$ to the instantiation for each parameter; we thus have $\vec{u}_i = 0.5 + \varepsilon$ for all parameter p_i.. After every parameter has performed a step of less than 10^{-6} in sequence, we conclude a local optimum has been found. When this local optimum is infeasible, we restart by selecting a new instantiation $\vec{u}$ randomly (see Algorithm 8 line 4). Consequently, the GD methods may yield different run-times on different invocations on the same benchmark, though in practice we observe only a small deviation in the run-times.

6.7.2 Results

The results are given as scatter plots in Figures 6.10 through 6.13 with a log-log scale. For all plots, the points on the TO/MO line denote that the method either a time-out or memory-out and the points on the ERR line denotes that the method has encountered some internal error. The dashed lines denote differences of factor 10 and 100.

Figure 6.10a and Figure 6.10b display how projection with learning rate $\eta = 0.1$ (x-axis) compares to all other restriction methods for the optimum bounds of all benchmarks using momentum-Sign and Adam, respectively. Furthermore, it shows results



(a) Momentum-Sign GD (b) Adam

Figure 6.10: Comparison of different region restriction methods

for a learning rate of 0.01 and 0.001. We chose these two GD methods as they turn out to be the best for the classical (momentum-Sign) and adaptive (Adam) GD methods, see also Section 6.7.4. Point (x, y) denotes that the restriction method projection took x seconds and the alternative took y seconds to find a feasible instantiation for the given GD method. A learning rate of 0.1 is denoted in the legend by lr=0.1. We use Figures 6.10a and 6.10b to justify our learning rate choice and answer question Q6.1: Which region restriction method works best?

The scatter plot in Figure 6.11a displays how momentum-Sign GD (x-axis) compares to the classical methods for the optimum bounds of all benchmarks. Furthermore, Figure 6.11b displays how momentum-Sign GD (x-axis) relates to its non-sign equivalent. In Figure 6.11c, we compare the adaptive methods, i.e., it displays how Adam (x-axis) compares to RAdam and RMSProp. Finally, Figure 6.11d shows how momentum-Sign GD compares to Adam and RMSProp. For all scatter plots in Figure 6.11, point (x, y) denotes that the method from the x-axis (momentum-Sign GD or Adam) took x seconds and the alternative took y seconds to find a feasible instantiation. We use the plots of Figure 6.11 to answer question Q6.2: Which GD method works best?

The scatter plots in Figures 6.12 and 6.13 show how the different GD methods and the baseline methods PSO, QCQP, and SCP (y-axis) compare to momentum-Sign GD (x-axis), respectively. Point (x, y) denotes that momentum-Sign GD took x seconds and the alternative took y seconds to find a feasible instantiation. For PSO, QCQP, and SCP we compared on the optimum bounds (left) and 20% relaxed bounds (right).

6.7.3 Comparison of region restriction methods

Before we answer the question on the region restriction methods (Q6.1), we observe from Figure 6.10 that a learning rate of 0.1 yields the lowest run-times. We now compare the different restriction methods for momentum-Sign and Adam. First of all, we observe that the barrier function had a time-out or memory-out or an error for (almost) all learning rates. Thus both projection and the logistic function perform better than the barrier function. The errors can be explained as follows. Recall that the error occurs if the barrier function is not preventing us from going outside of the region. For several of the benchmarks, many 'easy-parameters' occur; for those parameters, the optima often lie at the edges of the region. As we choose a relatively larger learning rate, the new instantiations quickly move to the edges of the region as the optimum is there. The barrier function cannot prevent the instantiation of getting out of the region. Thus errors may occur. The time-outs are caused by the barrier function method not allowing us to get close enough to the edge of the region. Secondly, we observe from Figure 6.10a that for momentum-Sign GD, the logistic function is slightly outperformed by projection. Furthermore, we observe from Figure 6.10b that for Adam, the logistic function is outperformed by projection. The projection outperforming logistic function can again be explained from the 'easy-parameters'. As many 'easy-parameters' occur; the optima often lie at the edges of the region and projection is computationally easier than the logistic function.

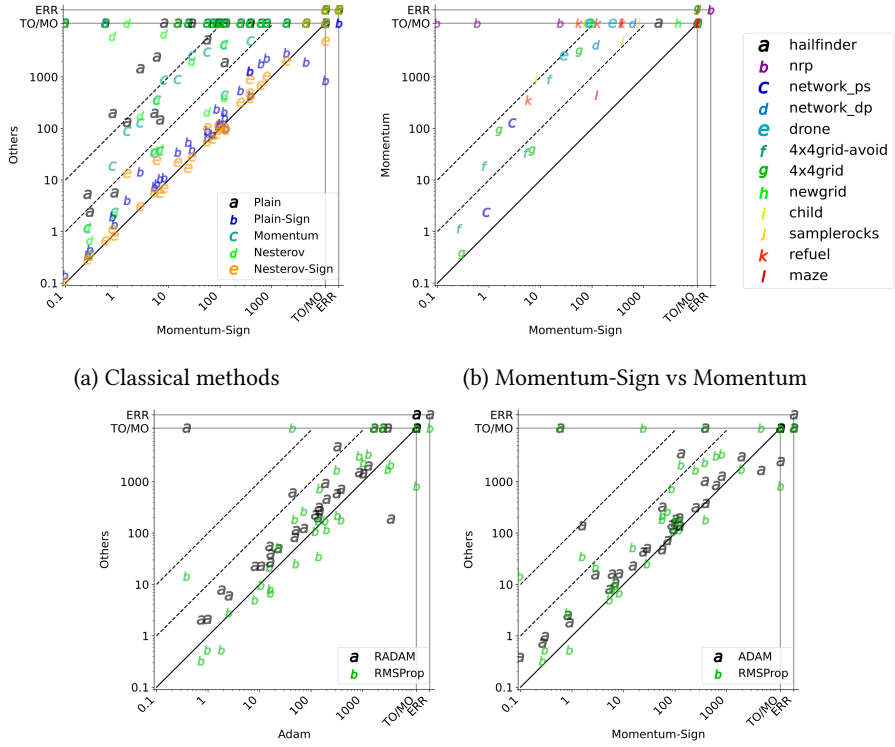
Answer to Q6.1 (Which region restriction method works best?)

Projection is the best region restriction method for pMCs.

6.7.4 Comparison of GD methods

Classical Methods | For the classical methods, we, first of all, observe from Figure 6.11a that momentum-Sign GD often outperforms the other classical methods from Section 6.3. Secondly, we observe from Figures 6.11a and 6.11b that for the classical methods, those methods where only the sign of the gradient is respected (and not the value gradient itself) often outperform their alternative. There are two possible explanations for this.

1. For most benchmarks, several 'easy-parameters' occur, the sign method does not respect the magnitude of the gradient, so the step size does not change too much over time. Therefore, the parameter might reach its bound on the region faster or even jump over it. As we use projection, we fix the parameter to this bound and do not need to update it anymore.



(a) Classical methods

(b) Momentum-Sign vs Momentum

(c) Adaptive methods

(d) Momentum-Sign vs best adaptive

Figure 6.11: Comparison of different GD methods

2. There might be one or more parameters that have a large influence on the expected total reward. If one of these parameters gets updated to a large enough value during its first update, we might obtain a feasible solution before considering other parameters. The sign method may update a parameter with larger values than its non-sign alternative.

Monotonicity could be a cause of the 'easy-parameters'; also the ordering of parameters based on their influence on the expected total reward needs further investigation (see Section 6.8).

Adaptive Methods | We observe from Figure 6.11c that Adam outperforms RAdam on most benchmarks, although not by orders of magnitude. This can be explained

by the choice of β_2, as we pick this to be 0.999, we have $\hat{s}_i^{t+1} < 1$ (cf. Equations (6.6) and (6.9)), thus the adapted learning rate for the first iteration is larger than 0.1 (cf. Equation (6.10)). As several 'easy-parameters' occur, we get to the boundary of the region faster with Adam than with RAdam. If we compare Adam and RMSProp, we see that Adam slightly outperforms RMSProp on some benchmarks, whereas RMSProp slightly outperforms Adam on other benchmarks.

Best Method | To determine the best GD method, we compare the best classical GD method (momentum-Sign GD) to the best adaptive GD methods (Adam and RMSProp). As we can see from Figure 6.11d, momentum-Sign GD outperforms Adam and RMSProp on most benchmarks. This can be explained by the fact the adaptive methods are meant to tailor the learning rate to the gradient of the single parameters, i.e., a parameter with a larger gradient (over time) has a lower learning rate than a parameter with a smaller gradient. Having a lower learning rate for a larger gradient also prevents one from jumping over a (local) optimum. As we search for a feasible solution, we do not care jumping over a (local) optimum, as the found value might still be feasible. Another explanation is the occurrence of 'easy-parameters', for which the magnitude of the gradient is possibly large. Note that this is important for Adam and RMSProp as they use the magnitude of the gradient. If we were able to analyze the expected total reward function, we would probably have an optimal value at ∞ or $-\infty$, as the parameters are possibly monotonic (see Section 6.8). For these parameters, the magnitude of the gradient is probably larger, and the optimum lies at the edge of the region. As we use projection to map these parameters back to the region, a lower learning rate (as for Adam/RMSProp) only slows down the search.

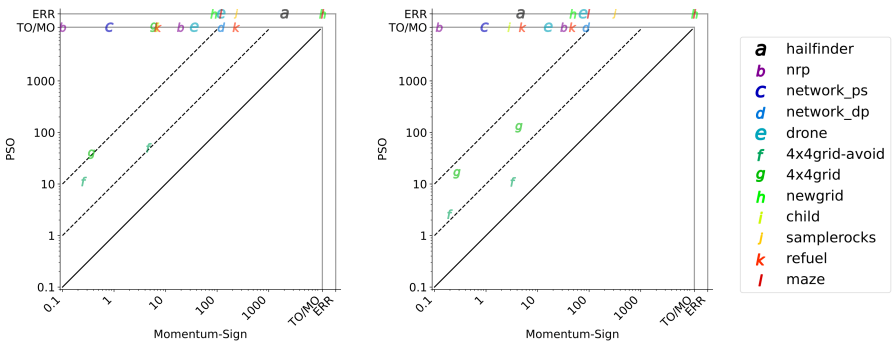
Answer to Q6.2 (Which GD method works best?)

Momentum-Sign GD is the best GD method for pMCs.

6.7.5 Comparison to state-of-the-art feasibility methods

In this section, we compare momentum-Sign GD to the state-of-the-art methods PSO, QCQP and SCP. We use the results to answer question Q6.3.

Comparison to PSO | First of all, if we compare PSO to momentum-Sign GD, our experiments (Figure 6.12) show that momentum-Sign GD always outperforms PSO on both the optimum bounds and the relaxed bounds. PSO has an error during preprocessing of the pMC on some benchmarks; this happens as these benchmarks violate an implicit assumption by the PSO-implementation. Furthermore, for several benchmarks, PSO has time-outs and memory-outs due to the large number of parameters.



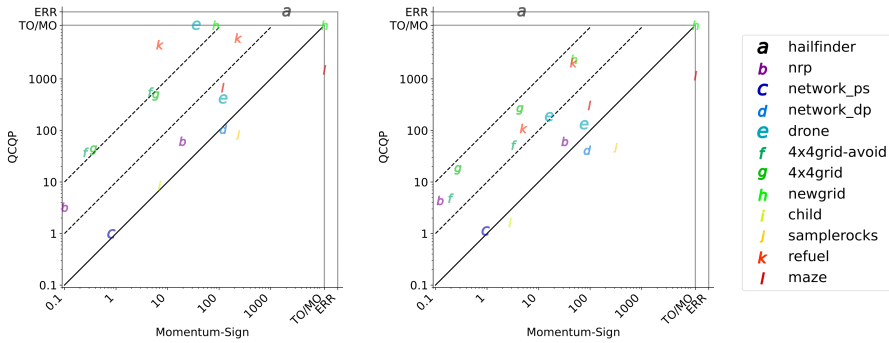
(a) Optimum bounds (b) 20% bounds

Figure 6.12: Comparison of GD with PSO

Only for 4x4-grid-avoid and one instance of 4x4-grid, PSO obtains results. However, momentum-Sign GD is at least one order of magnitude faster for these benchmarks.

Comparison to QCQP | Now consider Figure 6.13. First of all, we observe that for both the optimum bounds and the relaxed bounds momentum-Sign GD outperforms QCQP for most benchmarks by at least one order of magnitude. Secondly, we observe that if we relax the bounds (Figure 6.13b), QCQP performs better. QCQP's number of time-outs is reduced to almost none, except for the newgrid benchmark that neither momentum-Sign GD nor QCQP can solve. It is fair to say that QCQP benefits at least slightly more from the more relaxed bounds than momentum-Sign GD, but even then, momentum-Sign GD remains significantly faster on most benchmarks. Finally, we observe that QCQP outperforms momentum-Sign GD for the samplerocks benchmarks. As Table 5.3 (page 118) suggests, the samplerocks benchmarks might have some monotonic parameters. For better performance on this benchmark, it is possible to use monotonicity checking as preprocessing. All parameters that are monotonic can then be set to their boundary immediately and do not need to be considered in the updating of the parameters. All possibly monotonic parameters can initially be set to their boundary, but they still need to be taken into account when updating the parameters.

Comparison to SCP | We now compare momentum-Sign GD to SCP. The results are depicted in Figure 6.14. First of all, we observe that for several of the benchmarks with optimum bounds, SCP throws an error, as SCP cannot find a feasible instantiation. Secondly, we observe for the optimum bounds that there are several benchmarks for



(a) Optimum bounds (b) 20% bounds

Figure 6.13: Comparison of GD with QCQP

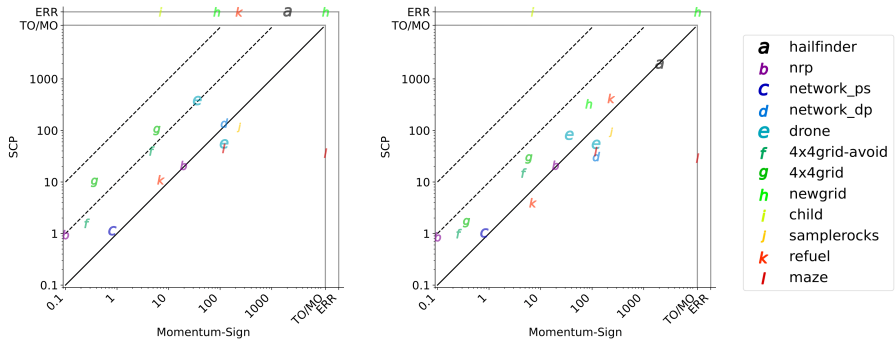
which momentum-Sign GD outperforms SCP by at least one order of magnitude. As for QCQP, SCP can solve the maze instance for which momentum-Sign GD has a TO/MO. Finally, if we relax the bounds (Figure 6.13b), SCP performs better. With relaxed bounds, the run-time for SCP is reduced, e.g., for `4x4-grid` we see that the run-time is reduced by almost an order of magnitude. Moreover, we observe that SCP and momentum-Sign GD on the relaxed bounds perform within an order of magnitude for most benchmarks. As already discussed in the comparison to QCQP, we could improve momentum-Sign GD by adding a check for monotonicity to the preprocessing.

Answer to Q6.3 (How does GD compare to PSO, QCQP, and SCP?)

- On optimum bounds momentum-Sign GD outperforms PSO, QCQP, and SCP.
- On relaxed bounds momentum-Sign GD outperforms PSO, and QCQP, and momentum-Sign GD and SCP perform alike.

6.8 Conclusion

This chapter showed how GD could be used to solve the feasibility problem, as introduced in Section 2.4. We discussed several classical (plain, momentum, Nesterov-accelerated, plain-Sign, momentum-Sign, Nesterov-accelerated-Sign) and adaptive (RMSProp, Adam, RAdam) GD methods. Also, several region restriction methods were discussed. Furthermore, we have shown that we can efficiently evaluate the gradient



(a) Optimum bounds (b) 20% bounds

Figure 6.14: Comparison of GD with SCP

at a given instantiation using an equation-based system. We also showed that this equation-based system could be obtained via the derived pMC. We evaluate the GD approach for the feasibility problem by running several benchmarks and comparing the different region restriction methods and GD methods. We also compared the GD approach to the existing methods: PSO, QCQP, and SCP. From the experiments, we obtain that the projection outperforms the other region restriction methods. Furthermore, momentum-Sign GD outperforms the other GD methods. Finally, we see that momentum-Sign GD outperforms PSO on all benchmarks. For most benchmarks, momentum-Sign GD outperforms QCQP; however, not for all. For the relaxed bounds, momentum-Sign GD and SCP perform akin. To improve momentum-Sign GD, we could use, e.g., monotonicity checking as a preprocessing step. As for monotonic parameters, the optimal values lie at the bounds of the region. Other future work includes investigating the influence of a single parameter on the expected total reward.

7 Parametric Probabilistic Timed Automata

Summary | A probabilistic timed automaton (PTA) is a non-deterministic model that exhibits both timed behavior (as in timed automata) and probabilistic behavior (as in MDPs). As for MC and MDPs, these probabilities are often bounded to a given interval and can be dependent on each other. Therefore, we introduce parametric probabilistic timed automata (pPTAs), in which the probabilities are given symbolically by parameters in rational functions. First of all, we introduce until properties in Section 7.2, which are a slight extension to the properties we have seen so far. Secondly, in Section 7.3, we introduce PTA and define its semantics as an infinite state MDP. Next, we define pPTAs as an extension of PTAs (Section 7.4). Finally, we show how we can obtain finite-state pMDPs (Section 7.5) using digital clocks semantics and backward reachability semantics and discuss the usability for pPTA of other existing methods to obtain finite-state MDPs for PTAs. This transformation is helpful as it allows us to analyze the pMDPs to obtain results for the pPTA; we can, e.g., check for monotonicity of parameters in the pPTA by obtaining the associated pMDP and applying the monotonicity approach from Chapter 4.

Origins | This chapter is an extended version of [2], which builds upon [Koh20]. Section 7.5 follows the lines of [NPS13]. The digital clocks approach for PTAs is originally discussed in [HMP92; AMP98; KNPS06], the backward reachability approach originates from [KNSW07], and the region graph approach and forward reachability approach originates from [KNSS02]. Finally, the stochastic games abstraction is introduced in [KNP09]. Furthermore, we refer to the results of Hartmanns and Kohlen in [HK22].

Background | We use the basic notation from Section 2.1. Furthermore, we consider pMDPs as introduced in Section 2.2. The other preliminary knowledge is given in Section 7.3 and based on [AD94; GJ95; KNSS02; BK08].

Assumption 7.1 We assume graph-preserving region R to be given.

7.1 Related Work

Analyzing probabilistic timed automata | Most approaches that analyze PTAs, abstract from the continuous-time behavior by turning the PTA into an equivalent MDP, which can be verified using standard MDP model checking [BAFK18; BHK19]. These methods include the region graph [KNSS02], backward reachability [KNSW07], and digital clocks [HMP92; KNPS06]. Both PRISM [KNP11] and the Modest Toolset's [HH14; HK22] mcsta model checker implement backward reachability and digital clocks. PRISM also contains a stochastic games approach [KNP09]. Further research includes statistical model checking (SMC), which interprets the non-determinism in the PTA probabilistically [AP18]. The obtained probability is thus between the minimum and maximum probability in the PTA. This is implemented in, e.g., Uppaal smc [DLLM+15]. Also (an extension to) lightweight scheduler sampling [LST14] is used to analyze PTAs [DHLS16; HSD17]. The Modest Toolset's modes simulator [BDHS20] can deliver upper (lower) bounds on the minimal (maximal) probabilities via SMC.

Parametric probabilities in timed automata | Closest to pPTA are interval PTA (iPTA) [KG11]. An iPTA is a pPTA in which all parameters are independent, i.e., no parameter occurs at multiple states (cf. iMCs and pMCs). Krause and Giese [KG11] provide a tool to model-check interval PTA.

Parameters in timed automata | Instead of parameterizing the probabilities in the PTA, one can also parameterize the delays [AHV93] by introducing parameters into the clock constraints of edge guards and location invariants. The resulting model is a constraint-parametric TA (cpTA), where the clock constraints are parameterized. For cpTA, basic problems, like finding a parameter instantiation such that a goal state is reachable, are undecidable [AHV93; And19], except in restricted cases such as bounded integer values [JLR15] or L/U TA [HRSV02]. Recent work includes a semi-algorithm for liveness in cpTA [AAPP21]. André et al. [ACFE09] introduce an inverse method for cpTA. This method extends a parameter instantiation (for a delay) to a set of instantiations, such that all instantiations in this set satisfy the same reachability properties. In [AFS13], André et al. adapt this such that a set of instantiations results in the same reachability probabilities. In [ADF20], André et al. combine cpPTA with iMCs [JL91; CSH08] to obtain parametric interval PTA (piPTA). In piPTAs, the delays are parameterized, and intervals of probabilities replace the probabilities. The main disadvantage of this model is that dependencies between probabilities can not be expressed. Research on piPTAs currently remains focused on the question of consistency [ADF20].

7.2 Until Properties

In the previous chapters, we considered the probability and expected total reward of eventually reaching a state in T. In this chapter, we consider the probability of staying in a set of *safe* states $U \subseteq S \setminus T$ until we reach some state in T, denoted $U \cup T$. We modify the definition of (p)MDPs by removing the set of target states from the model. Instead, we assume a set of safe states U and a set of target states T is given. For a (p)MDP $\mathcal{M}$ with $\pi \in Paths_{inf}(\mathcal{M})$, we have $\pi \models U \cup T$ if there exists an $i \in \mathbb{N}$ such that for all $0 \leq j < i$ we have $\pi_j \in U \setminus T$ and $\pi_i \in T$. For an MDP, we let $\Pr^s(U \cup T)$ denote the probability of eventually reaching T from s while staying in U before. Analogously, we let $\Pr^s_{min}(U \cup T)$ and $\Pr^s_{max,\mathcal{M}}(U \cup T)$ denote the minimal and maximal probability for this event, respectively. For a formal definition of these probabilities in MDPs, we refer to, e.g., [BK08, Ch. 10]. By applying the construction of Section 2.3.4, we obtain the solution functions ($\Pr^{s \xrightarrow{U} T}$, $\Pr^{s \xrightarrow{U} T}_{min}$, and $\Pr^{s \xrightarrow{U} T}_{max}$) for pMDPs.

·safe

> **Remark 7.1** Note that $\Diamond T \equiv U \cup T$, so the probability of eventually reaching a state in T from s can be obtained through $\Pr^{s \xrightarrow{S} T}$.

7.3 Probabilistic Timed Automata

A *timed automaton* (TA) [AD94] is a labeled transition system [BK08, Ch. 2] extended with a finite set $\mathcal{X}$ of *clocks*. Probabilistic timed automata [GJ95; KNSS02] extend timed automata with probabilistic transitions.

timed automaton
clock

The clocks are used to represent time, have non-negative real values and increase at rate 1 over time. A *clock valuation* is a function $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ which assigns non-negative real values to the clocks. We denote the set of all clock valuations by $\mathbb{R}^{\mathcal{X}}_{\geq 0}$. For $v \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$ for all $x \in \mathcal{X}$. For $X \subseteq \mathcal{X}$, $v[X := 0]$ is the clock valuation where $v[X := 0](x) = 0$ if $x \in X$ and $v[X := 0](x) = v(x)$ otherwise. Finally, $\mathbf{0}$ is the zero valuation, i.e., $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$. In order to reason about clock values, we introduce clock constraints.

·valuation

> **Definition 7.1** A *clock constraint* over set $\mathcal{X}$ of clocks is formed according to the grammar
>
> $$\mathcal{X} ::= x < c \mid x \leq c \mid x > c \mid x \geq c \mid \mathcal{X} \wedge \mathcal{X}$$
>
> where $x \in \mathcal{X}$ and $c \in \mathbb{N}$. Let $CC(\mathcal{X})$ denote the set of all clock constraints over $\mathcal{X}$.

·constraint

For valuation $v \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ and clock constraint $\mathcal{X} \in CC(\mathcal{X})$, we have $v \models \mathcal{X}$ iff the expression $\mathcal{X}$ evaluates to true after replacing each $x \in \mathcal{X}$ with $v(x)$.

Remark 7.2 To keep the presentation simple, we omit diagonal ($x - y \sim c$) and disjunctive clock constraints ($\mathcal{X} \vee \mathcal{X}$). The diagonal constraints can be added at the expense of a slightly more involved theory. Furthermore, we often abbreviate clock constraints, e.g., $(x > 1) \wedge (x \leq 3)$ may be written as $1 < x \leq 3$ or $x \in (1, 3]$.

Example 7.1 Consider set of clocks $\mathcal{X} = \{x, y\}$. A possible clock valuation is v with $v(x) = 1$ and $v(y) = 2$. For $t = 3$, we obtain valuation $v' = v + t$ with $v'(x) = 4$ and $v'(y) = 5$. For $X = \{x\}$, we have $v[X := 0](x) = 0$ and $v[X := 0](y) = 2$. The clock constraint denoting x being at most 2 and y being at most 24 is given by $\mathcal{X} = (x \leq 2) \wedge (y \leq 24)$.
We observe that we can write $\mathcal{X} = x \leq 4 \vee y \leq 10$ as $\mathcal{X}' = x > 4 \wedge y > 10$ and check for $v \not\models \mathcal{X}'$ instead of $v \models \mathcal{X}$.

zone

The semantics of clock constraint $\mathcal{X}$ is the *zone* $\zeta_{\mathcal{X}} = \{v \in \mathbb{R}^{\mathcal{X}}_{\geq 0} \mid v \models \mathcal{X}\}$. We let $Zones(\mathcal{X})$ denote the set of zones over the constraints $\mathcal{X}$. For zones, normal set operations apply, e.g., $\zeta_{\mathcal{X}_1} \cup \zeta_{\mathcal{X}_2} = \zeta_{\mathcal{X}_1 \vee \mathcal{X}_2}$.

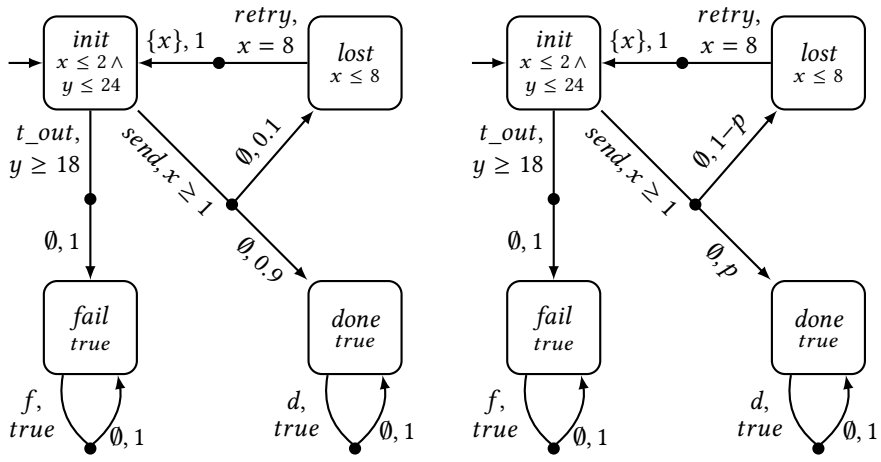
Example 7.2 We continue Example 7.1 with $\mathcal{X} = (x \leq 2) \wedge (y \leq 24)$, $v(x) = 1$, $v(y) = 2$, and $v' = v + 3$. We have $v \models \mathcal{X}$ and $v' \not\models \mathcal{X}$. The zone $\zeta_{\mathcal{X}}$ contains all valuations of clocks x and y such that $\mathcal{X}$ is satisfied, i.e., $\{v \in \mathbb{R}^{\mathcal{X}}_{\geq 0} \mid v(x) \leq 2 \wedge v(y) \leq 24\}$.

probabilistic TA

Definition 7.2 A *probabilistic timed automaton (PTA)* is a tuple $\mathcal{A} = (Loc, l_I, Act, \mathcal{X}, \mathbf{P}, inv)$ with a finite set Loc of *locations*, an *initial location* $l_I \in Loc$, a finite set Act of *actions*, a finite set $\mathcal{X}$ of clocks, *partial probabilistic transition function* $\mathbf{P}\colon Loc \times CC(\mathcal{X}) \times Act \nrightarrow \mathrm{Distr}(2^{\mathcal{X}} \times Loc)$, and *location invariant* $inv\colon Loc \to CC(\mathcal{X})$.

edge

We display a PTA as a graph, where the vertices are the locations. Only if $p = \mathbf{P}(l, g, \alpha)(X, l') > 0$, then there is an *edge* (l, g, α, p, X, l') between l and l' labeled with action $\alpha \in Act(l)$, guard g, and set of reset clocks $X \subseteq \mathcal{X}$. We assume that every edge is uniquely identified by its source location and action. We write $(l, \alpha) \in \mathbf{P}$ whenever there exists an edge from l labeled with action α. Furthermore, for $(l, \alpha) \in \mathbf{P}$, we let $edges(l, \alpha) = \{(l, g, \alpha, p, X, l') \mid \exists \alpha \in Act(l), g \in CC(\mathcal{X}), \mathrm{X} \subseteq \mathcal{X}, l' \in Loc. \mathbf{P}(l, g, \alpha)(X, l') = p > 0\}$ denote the set of all edges for (l, α). Note that by assumption for each location l and action α exactly one edge exists. A timed automaton (TA) is a PTA in which all edges have probability 1.



(a) PTA $\mathcal{A}$ (b) pPTA $\mathcal{P}$

Figure 7.1: Example (parametric) PTA adapted from [2]

Remark 7.3 For a given set U of safe locations and set $T \subseteq Loc$ of target locations, we only discuss reaching T safely in unbounded time. In case of a time-bounded property, like reaching T within n time units after staying in U before, one could modify the PTA by adding clock $z \in \mathcal{X}$, which does not occur in any invariant or guard of $\mathcal{A}$, that is checked against bound $n \in \mathbb{N}$. This goes as follows. For all $l \in T \cup U$, we modify the state invariants and transitions going to l such that $z \leq n$ holds, i.e., $inv'(l) = inv(l) \wedge z \leq n$ and $\mathbf{P}'(l', g \wedge z \leq n, \alpha)(X, l) = \mathbf{P}(l', g, \alpha)(X, l)$. For the remainder of this chapter, we focus on unbounded reachability properties.

Example 7.3 Figure 7.1a depicts PTA $\mathcal{A}$ with four locations (*init*, *lost*, *fail*, *done*) and two clocks x and y. Each location is labeled with its name and its location invariant. Initially, the system is in location *init*. As long as the location invariant holds, the system can remain in its initial location. After one time unit, the probabilistic edge (with action *send*) to *done* and *lost* is enabled as its guard is satisfied. After 2 time units, the transition with action *send* must be taken, as we otherwise violate $inv(init)$. If the system moves to the *lost* location, it remains there as long as x is at most 8. Once x equals 8, the edge back to *init* is taken, and clock x is reset to 0. Note that we cannot stay in *lost* any longer, as the location invariant is then violated. Once 18 time units passed, and we are again in *init*, the

edge to *fail* is enabled.

We use (infinite-state) MDPs to define the semantics of PTAs; this is a slight modification of Definition 2.3, as we now allow for an infinite number of states in the MDP.

dense-time

Definition 7.3 The *dense-time* semantics of a $\mathcal{A}$ is the MDP $[\![\mathcal{A}]\!]_{dense} = (S, s_I, Act', \mathbf{P}')$ with set of states $S = \{(l,v) \in Loc \times \mathbb{R}^{\mathcal{X}}_{\geq 0} \mid v \models inv(l)\}$, initial state $s_I = (l_I, \mathbf{0})$, set of actions $Act' = Act \cup \mathbb{R}_{\geq 0}$, and if $\mathbf{P}(l,g,\alpha,\cdot,\cdot)$ is defined, then

$$\mathbf{P}'((l,v),\alpha,(l',v')) =$$

$$\begin{cases} \sum\limits_{\substack{X \subseteq \mathcal{X}: \\ v'=v[X:=0]}} \mathbf{P}(l,g,\alpha,X,l') & \text{if } \alpha \in Act \wedge v \models g \\ 1 & \text{if } \alpha \in \mathbb{R}_{\geq 0} \wedge l = l' \wedge v' = v + \alpha \models inv(l) \\ 0 & \text{otherwise.} \end{cases}$$

The set of target states (T) is obtained by $T = \{(l,v) \in T \times \mathbb{R}^{\mathcal{X}}_{\geq 0} \mid v \models inv(l)\}$, the set of safe states (U) is obtained analogously. We refer to transitions due to taking action $\alpha \in Act$ as *probabilistic jumps* and those due to taking $\alpha = d \in \mathbb{R}_{\geq 0}$ as *delays* of duration d.

well-formed

We assume a PTA to be *well-formed*: if $\mathbf{P}'((l,v),\alpha,(l',v')) > 0$ then $v' \models inv(l')$, for all reachable (l,v).

Observe that $[\![\mathcal{A}]\!]_{dense}$ has uncountably many states and is infinitely branching in general. As $[\![\mathcal{A}]\!]_{dense}$ is an MDP, the definitions of paths (see Section 2.3.1) apply. We assume, w.l.o.g. that the first transition of a path is a delay and that delays and jumps alternate. $[\![\mathcal{A}]\!]_{dense}$ may contain non-divergent paths [KNSS02], i.e., paths along which infinitely many jumps happen in a finite amount of time.

divergent

Definition 7.4 The path $\pi \in Paths_{inf}[\![\mathcal{A}]\!]_{dense}$ is *divergent* if the sum of the duration of its delays—its *elapsed time*—is ∞.

We lift the notion of a scheduler to PTA $\mathcal{A}$ using the above semantics, i.e., $Sched(\mathcal{A}) := Sched([\![\mathcal{A}]\!]_{dense})$ and $\mathcal{A}^\sigma := [\![\mathcal{A}]\!]^\sigma_{dense}$. For a PTA $\mathcal{A}$ and MDP $\mathcal{M} = [\![\mathcal{A}]\!]_{dense}$, we restrict ourselves to *divergent* schedulers ($Sched_{div}(\mathcal{M})$), i.e., for all $\sigma \in Sched_{div}(\mathcal{M})$ and $\pi \in Paths_{inf}(\mathcal{M}^\sigma)$ we have π is divergent.

Assumption 7.2 From now on we only consider divergent schedulers and drop subscript *div*.

Probabilities in PTA | Let $\zeta_l = \{v \in \mathbb{R}^{\mathcal{X}}_{\geq 0} \mid v \models inv(l)\}$ be the zone where the location invariant of l is satisfied. We let $\Pr^{l_I}_{\mathcal{A}}(U \cup T)$ denote the probability mass of all infinite paths in $\mathcal{A}$ that start in $(l_I, \mathbf{0})$, then visits pairs in $\{(l, \zeta_l) \mid l \in U\}$ and finally visits some pair (l, ζ_l) for $l \in T$. Furthermore, we let $\Pr^{l_I}_{min,\mathcal{A}}(U \cup T) = \min\{\Pr^{l_I}_{[\![\mathcal{A}]\!]^{\sigma}_{dense}}(U \cup T') \mid \sigma \in Sched([\![\mathcal{A}]\!]_{dense})\}$ and define $\Pr^{l_I}_{max,\mathcal{A}}(U \cup T)$ analogously. We omit subscript $\mathcal{A}$ whenever $\mathcal{A}$ is clear from the context.

7.4 Parametric Probabilistic Timed Automata

As for MC and MDPs, the probabilities in PTAs are often bounded to a given interval and can be dependent on each other. Therefore, we consider parametric PTAs, which extend PTAs by replacing the probabilistic transition function with a parametric one.

parametric PTA

Definition 7.5 A *parametric PTA (pPTA)* is a tuple $\mathcal{P} = (Loc, l_I, Act, \mathcal{X}, V, \mathbf{P}, inv)$ where $Loc, l_I, Act, \mathcal{X}$ and inv are as in Definition 7.2, V is as in Definition 2.3, and the *parametric probabilistic transition function* is given by:

$$\mathbf{P}\colon Loc \times CC(\mathcal{X}) \times Act \nrightarrow \mathrm{Distr}_{\mathbb{Q}(V)}(2^{\mathcal{X}} \times Loc)\ .$$

Example 7.4 Consider pPTA $\mathcal{P}$ in Figure 7.1b. The probability of successful transmission is p instead of 0.9 (cf. Figure 7.1a). All other transitions are non-parametric.

As for pMDPs, we can apply an instantiation $\vec{u}$ to pPTA $\mathcal{P}$ (yielding PTA $\mathcal{P}[\vec{u}]$) by replacing each $f \in \mathbb{Q}(V)$ in $\mathcal{P}$ by $f(\vec{u})$. As we consider distributions over $\mathbb{Q}(V)$, a pPTA is well-defined (see Section 2.2.1 (page 12)). The definition of graph-preserving instantiations directly transfers from pMDPs to pPTAs.

Example 7.5 We continue from Example 7.4. Applying instantiation $\vec{u}(p) = 0.9$ to $\mathcal{P}$ yields PTA $\mathcal{A}$ as depicted in Figure 7.1a. Instantiation $\vec{u}(p) = 1$ is well-defined but not graph-preserving, as the edge from *init* to *lost* has probability 0.

dense-time

Definition 7.6 The *dense-time* semantics of a pPTA $\mathcal{P}$ is the pMDP $[\![\mathcal{P}]\!]_{dense}$ over V defined analogously to Definition 7.3.

We observe that parameter instantiations and the dense-time semantics commute.

Lemma 7.1 For pPTA $\mathcal{P}$ and parameter instantiation $\vec{u}$:

$$[\![\mathcal{P}[\vec{u}]]\!]_{dense} = [\![\mathcal{P}]\!]_{dense}[\vec{u}] \ .$$

Proof. From Definition 7.6, we see that we need to prove that the transition functions are equivalent, i.e.,

$$\mathbf{P}'((l,v),\alpha,(l',v'))[\vec{u}] = \mathbf{P}'[\vec{u}]((l,v),\alpha,(l',v')) \ . \tag{7.1}$$

Recall that $\mathbf{P}'((l,v),\alpha,(l',v')) =$

$$\begin{cases} \sum\limits_{\substack{X\subseteq\mathcal{X}:\\ v'=v[X:=0]}} \mathbf{P}(l,g,\alpha,X,l') & \text{if } \alpha \in Act \wedge v \models g \\ 1 & \text{if } \alpha \in \mathbb{R}_{\geq 0} \wedge l = l' \wedge v' = v+\alpha \models inv(l) \\ 0 & \text{otherwise.} \end{cases}$$

For the latter two cases, Equation (7.1) follows trivially as no parameters occur. Let $l, l' \in Loc$, $v, v' \in \mathbb{R}^{\mathcal{X}}_{\geq 0}$ and $\alpha \in Act$ (wit guard g) be given such that $v \models g$ we have to show for $\vec{u} \in R$:

$$\mathbf{P}'((l,v),\alpha,(l',v'))(\vec{u}) = \sum_{\substack{X\subseteq\mathcal{X}:\\ v'=v[X:=0]}} \mathbf{P}(l,g,\alpha,X,l')(\vec{u}) \ .$$

Clearly this holds by definition, so Equation (7.1) holds thus parameter instantiations and the dense-time semantics commute. ∎

As for pMCs and pMDPs, we have for a pPTA $\mathcal{P}$ with set of safe location U and set of target locations T that $\Pr^{l_I}_{\mathcal{P}}(U \cup T)$ is not a constant, but rather a function $\Pr^{l_I \xrightarrow{U} T}_{\mathcal{P}}$ such that $\Pr^{l_I \xrightarrow{U} T}_{\mathcal{P}}(\vec{u}) = \Pr^{l_I}_{\mathcal{P}[\vec{u}]}(U \cup T)$. We define $\Pr^{l_I \xrightarrow{U} T}_{min,\mathcal{P}}(\vec{u})$ and $\Pr^{l_I \xrightarrow{U} T}_{max,\mathcal{P}}(\vec{u})$ analogously. We omit subscript $\mathcal{P}$, whenever $\mathcal{P}$ is clear from the context.

7.5 From pPTAs to pMDPs

The dense-time semantics, as introduced in the previous section, has an infinite state space. However, for PTAs, there exist techniques that transform them into finite-state MDPs. In this section, we show which of the techniques carry over to the parametric case. First, we discuss in detail the adaptation of the digital clocks semantics [HMP92; AMP98; KNPS06] and backward reachability semantics [KNSW07]. Then, we discuss

shortly how we can modify the region graph approach [KNSS02] to deal with parameters. Furthermore, we explain why the forward reachability approach [KNSS02] is unsuitable for pPTA. Finally, we discuss the stochastic games abstraction approach [KNP09]. The presentation of this section is along the lines of [NPS13].

7.5.1 Digital clocks

The main idea of the digital clocks approach is to only consider integer clock valuations, i.e., valuations in $\mathbb{N}^{\mathcal{X}}$, and delays of 1 time unit. Furthermore, the digital clocks approach requires the pPTA to be *closed*, i.e., all clock constraints must only contain non-strict comparisons such as $x \leq c$ and $x \geq c$ for $x \in \mathcal{X}$ and $c \in \mathbb{N}$. Furthermore, it requires the pPTA to be *diagonal-free*, i.e., clock values cannot be compared, e.g., $x + c \leq y + d$ is not allowed for $x, y \in \mathcal{X}$ and $c, d \in \mathbb{N}$.

·closed

·diagonal-free

For each clock x, we store its upper bound $\mathbf{k}_x + 1$, where $\mathbf{k}_x$ is the maximal constant to which x is compared in the (guards and location invariants of the) PTA. In this way, digital clocks give rise to a finite-state MDP. To that end, we define a new addition operator ($\oplus$) for clock valuations that bounds addition by the clock upper bounds, i.e., $(v \oplus t)(x) = \min\{v(x) + t, \mathbf{k}_x + 1\}$ for each $x \in \mathcal{X}$.

Assumption 7.3 When discussing the digital clocks approach, we assume the pPTAs to be closed and diagonal-free. Note that by this assumption T and U are closed as well.

Example 7.6 Recall pPTA $\mathcal{P}$ of Figure 7.1b. All clock constraints in $\mathcal{P}$ are closed. For clock x, we have $\mathbf{k}_x = 8$, so we have an upper bound of 9. We compare clock y to at most 24 at state *init*, this gives us $\mathbf{k}_y = 24$ and the upper bound for y of 25.

Next, we define the digital clocks semantics for pPTA. It is similar to [KNPS06, Def. 12], however, we allow transitions to be parametric.

Definition 7.7 The *digital clocks* semantics of a pPTA $\mathcal{P}$ is the pMDP $[\![\mathcal{P}]\!]_{dc} = (S, s_I, T', Act', V, \mathbf{P}')$ with $S = \{ (l, v) \in l \times \mathbb{N}^{\mathcal{X}} \mid v \models inv(l) \}$ and s_I, T', Act', and $\mathbf{P}'$ are as in Definition 7.6 (restricted to S), except that for time delays we use

digital clocks

$$\mathbf{P}'((l, v), \alpha, (l', v')) = 1 \quad \text{if } \alpha = 1 \wedge l = l' \wedge v' = v \oplus 1 \models inv(l').$$

Remark 7.4 As PTAs are a special case of pPTAs, we have for PTA $\mathcal{A}$, $[\![\mathcal{A}]\!]_{dc}$ defined by Definition 7.7.

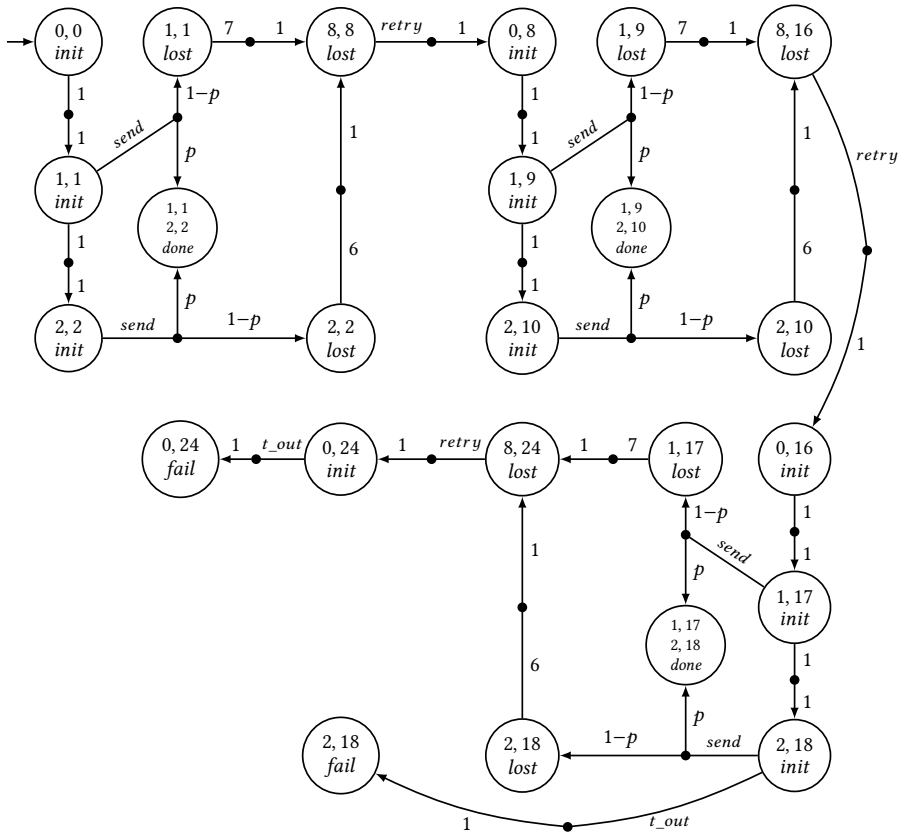


Figure 7.2: Digital clocks semantics of $\mathcal{P}$ of Figure 7.1b ($[\![\mathcal{P}]\!]_{dc}$)

Example 7.7 Reconsider pPTA $\mathcal{P}$ of Figure 7.1b. Figure 7.2 shows the digital clocks semantics $[\![\mathcal{P}]\!]_{dc}$. Note that for conciseness, we omit the self-loops at the *done* and *fail* states, and merged several *lost* states in which only time passes. Initially, there is only the time action available, moving from $(0, 0, \mathit{init})$ to $(1, 1, \mathit{init})$. Here we can choose between either letting time pass (by picking action 1) or taking an action (*send*). If we let time pass, we move to state $2, 2, \mathit{init}$, where we only can choose action *send*. When taking action *send*, the message is received with probability p and gets lost with probability $1-p$. At the lost states, we let 6 and 7 time units pass, respectively, and move to $(8, 8, \mathit{lost})$. This message sending is

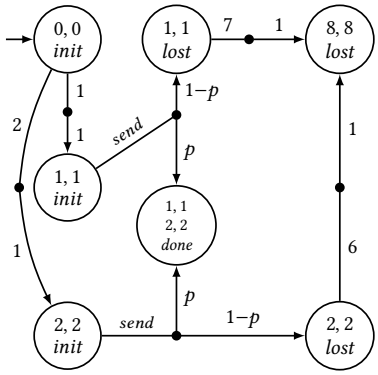


Figure 7.3: Merging time transitions

repeated until either the submission is successful or the clock y reaches a value of 18. At $y = 18$ (i.e., state 2, 18, *init*), there are two possible actions, as the action *t_out* is now enabled. If we take action *t_out*, we move to a *fail* state. If we take the action *send*, we again try to submit the message. Note that no time delay actions can be taken at this point, as this invalidates the state invariant of 2, 18, *init*. If the submission of the message at 2, 18, *init* was unsuccessful, then we reach state 0, 24, *init*, now the only possible action is a time-out (*t_out*), leading to the *fail* state.

Recall that in Section 7.3, we assume that the first transition of a path is a delay and that delays and jumps alternate. We can make up for this by merging n successive time edges of delay 1 into a single time edge of delay n.

Example 7.8 Figure 7.3 shows the result of merging the time transitions on the path from $(0, 0, \mathit{init})$ to $(8, 8, \mathit{lost})$. Instead of one choice at state $(0, 0, \mathit{init})$ we can now choose between taking one time unit or two time units. At state $(1, 1, \mathit{init})$, we now only have one choice, taking action *send*.

Correctness | To prove the correctness of the digital clocks semantics for pPTAs, we first show that parameter instantiation and digital clocks semantics commute.

Lemma 7.2 For pPTA $\mathcal{P}$ and parameter instantiation $\vec{u}$: $[\![\mathcal{P}[\vec{u}]]\!]_{dc} = [\![\mathcal{P}]\!]_{dc}[\vec{u}]$.

Proof. The proof is analogous to the proof of Lemma 7.1, however, we now use: $\mathbf{P}'((l,v),\alpha,(l',v')) =$

$$\begin{cases} \sum\limits_{\substack{X\subseteq\mathcal{X}:\\ v'=v[X:=0]}} \mathbf{P}(l,g,\alpha,X,l') & \text{if } \alpha \in Act \wedge v \models g \\ 1 & \text{if } \alpha = 1 \wedge l = l' \wedge v' = v \oplus 1 \models inv(l') \\ 0 & \text{otherwise.} \end{cases} .$$

■

Furthermore, minimal and maximal probabilities in the pPTA remain the same for digital clocks semantics.

Lemma 7.3 For any pPTA $\mathcal{P}$, set U of safe locations, set T of target locations, and $\vec{u}$ in region R, we have

$$\Pr^{l_I \overset{U}{\to} T}_{min,\mathcal{P}}(\vec{u}) = \Pr^{l_I \overset{U}{\to} T}_{min,[\![\mathcal{P}]\!]_{dc}}(\vec{u}) \text{ and } \Pr^{l_I \overset{U}{\to} T}_{max,\mathcal{P}}(\vec{u}) = \Pr^{l_I \overset{U}{\to} T}_{max,[\![\mathcal{P}]\!]_{dc}}(\vec{u}) .$$

Proof. We show the proof for minimal reachability, maximal reachability is proven analogously. From [KNPS06, Thm. 24], we know that Lemma 7.3 holds for PTA. As parameter instantiations and semantics commute (cf. Lemma 7.2), we have for any $\vec{u} \in R$:

$$\Pr^{l_I \overset{U}{\to} T}_{min,\mathcal{P}}(\vec{u}) = \Pr^{l_I \overset{U}{\to} T}_{min,\mathcal{P}[\vec{u}]} = \Pr^{l_I \overset{U}{\to} T}_{min,[\![\mathcal{P}[\vec{u}]]\!]_{dc}} = \Pr^{l_I \overset{U}{\to} T}_{min,[\![\mathcal{P}]\!]_{dc}}(\vec{u}) .$$

Thus the lemma holds. ■

Complexity | Since the parametric transition function does not influence the number of states in the semantics, the complexity of the digital clocks semantics for pPTA is similar to that of the digital clocks semantics for PTA. Therefore, an upper bound on the number of states in the digital clocks semantics is $|l| \cdot \prod_{x\in\mathcal{X}}(\mathbf{k}_x{+}1)$.

7.5.2 Backward reachability

As the digital clocks semantics yields a state space explosion and restricts the expressiveness of the pPTA (as it only allows for closed and diagonal-free pPTA), we consider backward reachability [KNSW07]. Instead of using pairs of locations and clock valuations to denote the state, the backward reachability algorithm computes a finite set of symbolic states that can reach the target location set T. The main drawback of the

backward reachability approach is that it is only designed to compute upper-bounded reachability properties (i.e., $\Pr^{s \xrightarrow{U} T} \gtrsim \lambda$ with $\gtrsim \in \{\geq, >\}$). However, as we observe in [KNSW07; HK22], we can reason about lower-bounded reachability properties as well.

A symbolic state is a pair $(l, \zeta) \in \mathit{Loc} \times \mathit{Zones}(\mathcal{X})$, for a set of symbolic states $V \subseteq \mathit{Loc} \times \mathit{Zones}(\mathcal{X})$, let $\zeta_V^l = \bigcup\{\zeta \mid (l, \zeta) \in V\}$ be all zones in V that are paired with location l. Let the zone consisting of all clock valuations that are within ζ or can reach ζ from ζ' while only using delays be defined by

$$\swarrow_{\zeta'}(\zeta) := \{v \mid \exists t \geq 0.\ (v + t \models \zeta \wedge \forall t' \leq t.\ (v + t' \models \zeta \cup \zeta'))\}\ .$$

Furthermore, we define the zone that represents ζ after resetting all clocks in X by:

$$[X := \mathbf{0}]\zeta := \{v \mid v[X := \mathbf{0}] \models \zeta\}\ .$$

We let $[\![\mathit{true}]\!] = \{(l, \zeta_{\mathit{inv}(l)}) \mid l \in \mathit{Loc}\}$ denote the set of all possible symbolic states.

Next, we define the time (tpre_U) and discrete (dpre) predecessor operations, which we later need to determine the reachable symbolic states in a backward manner. For set U of symbolic states, $\mathit{tpre}_U(T)$ determines the symbolic states that can reach a state in V by delays, all the while staying in U; dpre are those that can do so via probabilistic jumps.

Definition 7.8 Given a pPTA $\mathcal{P}$, and sets of symbolic states U and V, let:

$$\mathit{tpre}_U(V) := \{\left(l, \swarrow_{\left(\zeta_U^l \cap \zeta_l\right)} \left(\zeta_V^l \cap \zeta_l\right)\right) \mid l \in \mathit{Loc}\}\ , \text{and}$$

$$\mathit{dpre}((l, g, \alpha, p, X, l'), V) := \{\left(l, \zeta_g \cap \zeta_l \cap \left([X := 0]\zeta_V^{l'}\right)\right)\}\ .$$

Note that in the definition above, we use U and T for symbolic states instead of locations. If U and T represent locations, we obtain the symbolic states by considering the zone fulfilling the location invariant.

Example 7.9 Reconsider the pPTA of Figure 7.1b. Let $\mathcal{X}_1 = 1 \leq x \leq 2 \wedge y \leq 24$ and let $V = \{(\mathit{init}, \zeta_{\mathcal{X}_1})\}$ and $U = [\![\mathit{true}]\!]$. We then obtain $\mathit{tpre}_U(V) = \{(\mathit{init}, \zeta_{\mathcal{X}_2})\}$ with $\mathcal{X}_2 = (1 \leq x \leq 2 \wedge y \leq 24) \vee (x \leq 1 \wedge y = 23 + x)$. This is the zone that can reach the symbolic state $(\mathit{init}, \zeta_{\mathcal{X}_1})$ while using only time delays.
Next, we consider the discrete predecessor operation for $V = \{(\mathit{init}, \zeta_{\mathcal{X}_2})\}$ and edge $e = (\mathit{lost}, x = 8, \mathit{retry}, 1, \{x\}, \mathit{init})$. We obtain

$$\mathit{dpre}(e, V) = \{\left(\mathit{lost}, \zeta_{x=8} \cap \zeta_{\mathit{lost}} \cap \left([X := 0]\zeta_V^{\mathit{init}}\right)\right)\} = \{\left(\mathit{lost}, \zeta_{x=8 \wedge y \leq 23}\right)\},$$

as $[X := 0]\zeta_V^{\mathit{init}} = [X := 0]\zeta_{\mathcal{X}_2} = \zeta_{y \leq 23}$.

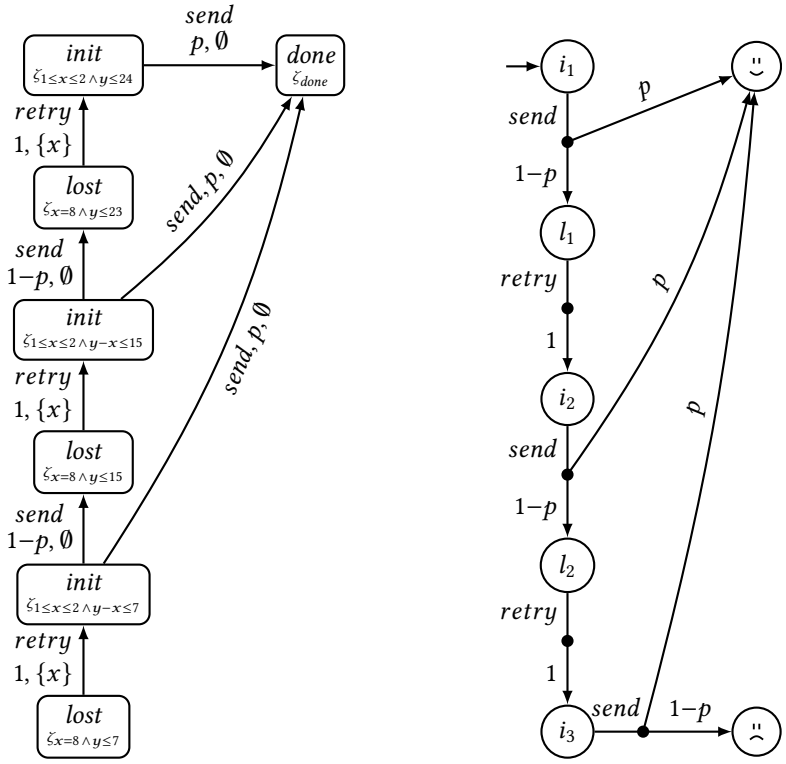
Algorithm 9 *MaxU*(U, T) [KNSW07]

1: $\mathtt{Z} \leftarrow tpre_{\mathtt{U} \cup \mathtt{T}}(\mathtt{T})$
2: **for** $(l, \alpha) \in \mathbf{P}$ **do**
3: $\quad E_{(l,\alpha)} \leftarrow \emptyset$
4: **repeat**
5: $\quad \mathtt{Y} \leftarrow \mathtt{Z}$
6: $\quad$ **for** $(l', \zeta) = \mathtt{y} \in \mathtt{Y} \wedge (l, \alpha) \in \mathbf{P} \wedge e = (l, g, \alpha, p, X, l') \in edges(l, \alpha)$ **do**
7: $\quad\quad \mathtt{z} \leftarrow \mathtt{U} \cap dpre(e, tpre_{\mathtt{U} \cup \mathtt{T}}(\mathtt{y}))$
8: $\quad\quad$ **if** $\mathtt{z} \neq \emptyset \wedge (\mathtt{z} \notin tpre_{\mathtt{U} \cup \mathtt{T}}(\mathtt{Z}))$ **then**
9: $\quad\quad\quad \mathtt{Z} \leftarrow \mathtt{Z} \cup \mathtt{z}$
10: $\quad\quad\quad E_{(l,\alpha)} \leftarrow E_{(l,\alpha)} \cup \{\mathtt{z}, (X, l'), \mathtt{y}\}$
11: $\quad\quad\quad$ **for** $(\overline{\mathtt{z}}, (\overline{X}, l'), \overline{\mathtt{y}}) \in E_{(l,\alpha)}$ **do**
12: $\quad\quad\quad\quad$ **if** $(\mathtt{z} \cap \overline{\mathtt{z}} \neq \emptyset) \wedge ((\overline{X}, l') \neq (X, l')) \wedge (\mathtt{z} \cap \overline{\mathtt{z}} \notin tpre_{\mathtt{U} \cup \mathtt{T}}(\mathtt{y}))$ **then**
13: $\quad\quad\quad\quad\quad \mathtt{Z} \leftarrow \mathtt{Z} \cup (\mathtt{z} \cap \overline{\mathtt{z}})$
14: **until** $\mathtt{Z} = \mathtt{Y}$

Algorithm 9 contains the *MaxU* algorithm [KNSW07] to find maximum probabilities of until formulae. This also works for pPTA with graph-preserving regions, as we only consider the fact that an edge exists and not its probability. As arguments, it takes U and T, which contain the symbolic states for safe locations U and target locations T, respectively. Z contains the reachable symbolic states, and $E_{(l,\alpha)}$ contains the transitions for these symbolic states. Initially, Z is set to *all* time predecessors of T (line 1), and $E_{(l,\alpha)}$ contains the empty set (line 3). We then copy the contents of Z to Y (line 5). For each symbolic state y in Y, which corresponds to location l', and edge yielding to l', we look which symbolic state can reach y over the chosen edge while staying in the safe symbolic states of U (line 7). If this symbolic state z exists and is not a time predecessor of the symbolic states observed so far (line 8), then we add z to the set of reachable symbolic states (line 9) and compute the edge leading to this state (line 10). For every edge, we now check if there already exists an edge starting in symbolic state $\overline{\mathtt{z}}$, which overlaps with z. If the clocks that are reset do not agree and the intersection z and $\overline{\mathtt{z}}$ is not in the time predecessors of y, then we discover a new symbolic state and add it to the set of symbolic states. This process is repeated until no new symbolic states are added.

backward reachability

Given a pPTA $\mathcal{P} = (Loc, l_I, Act, \mathcal{X}, V, \mathbf{P}, inv)$, set U of safe locations and set T of target locations, we construct the *backward reachability* semantics of a pPTA $\mathcal{P}$ as the pMDP $[\![\mathcal{P}]\!]_{br} = (S, s_I, T, Act, V, \mathbf{P}')$ by using Algorithm 9 as follows: The set of states S consists of all computed symbolic states, together with an absorbing bad state $\lightning$, i.e., $S = \mathtt{Z} \cup \{\lightning\}$. The initial state is a symbolic state that has $(l_I, \mathbf{0})$ as time predecessor, i.e.,



(a) Graph obtained via Algorithm 9 (b) $[\![\mathcal{P}]\!]_{br}$

Figure 7.4: Backward reachability semantics of pPTA $\mathcal{P}$ of Figure 7.1b

$s_I \in \{\mathbf{z} \in \mathbb{Z} \mid (l_I, \mathbf{0}) \in tpre_{U\cup T}(\mathbf{z})\}$. If there are several initial states, the optimal one can be found by calculating reachability probabilities for every possible initial state. The set of target states is given by $T = tpre_{U\cup T}(T)$. Finally, the transition function is obtained by going over all edges $E_{(l,\alpha)}$ (see [KNSW07, Fig.5]). If the probabilities of the outgoing transitions of a state do not sum up to 1, we add a transition from the state to $\frown$ to get proper distributions.

Example 7.10 We now apply the *MaxU* algorithm to our running example pPTA of Figure 7.1b. Recall, that the goal is to eventually reach location *done*, so $T = \{(done, \zeta_{done})\}$. Furthermore, as we consider an eventually formula we have $U = [\![true]\!]$, thus any computed symbolic state is in U. Initially, we set $\mathbb{Z}$ to T as all

time predecessors of T are already included in T since $\zeta_{done} = \zeta_{true}$. The obtained graph is given in Figure 7.4a. The pMDP $[\![\mathcal{P}]\!]_{br}$ we obtain from this is depicted in Figure 7.4b. Note that for conciseness, we renamed the states. Furthermore, we omit the symbolic state $(lost, \zeta_{x=8 \wedge y \leq 7})$ as this state is unreachable.

Correctness | In order to prove the correctness of the backward reachability semantics, we first show that (as for the digital clocks semantics) parameter instantiations and the backward reachability semantics commute.

Lemma 7.4 For pPTA $\mathcal{P}$, set of safe locations U, set of target locations T, and instantiation $\vec{u}$: $[\![\mathcal{P}[\vec{u}]]\!]_{br} = [\![\mathcal{P}]\!]_{br}[\vec{u}]$.

The proof is analogous to the proof of Lemma 7.2 and uses the fact that the *MaxU* algorithm functions similarly for PTA and pPTA. Next, we show that the backward reachability semantics preserves maximal probabilities, and minimal probabilities can be reconstructed.

Lemma 7.5 For any pPTA $\mathcal{P}$, set of safe locations U, set of target locations T, and $\vec{u}$ in region R, we have

$$\Pr^{l_I \overset{U}{\to} T}_{max,\mathcal{P}}(\vec{u}) = \Pr^{l_I \to T}_{max,[\![\mathcal{P}]\!]_{br}}(\vec{u}) \ .$$

Proof. The proof for maximal probabilities is analogous to that of Lemma 7.3, using Lemma 7.4 and that for any location l with symbolic state $s = (l, \zeta)$ such that $\emptyset \neq \zeta \subseteq \zeta_l$, we have $\Pr^{l}_{max,\mathcal{A}}(U \cup T) = \Pr^{s}_{max,[\![\mathcal{A}]\!]_{br}}(\Diamond T)$ for PTA $\mathcal{A}$ [KNSW07, Prop. 29]. ∎

To deal with minimal reachability probabilities, we must convert them into a maximal reachability problem. Instead of calculating the minimal probability for reaching T, while staying in the safe locations U, we now compute the maximum probability of the reverse, i.e., always seeing a location in $locs \setminus T$ until we see a location that is not in U. We refer to [KNSW07; Koh20] for details of this conversion.

Complexity | In the worst case, running the algorithm on a PTA $\mathcal{A}$ generates an MDP in which the set of symbolic states is the set $l \times Zones(\mathcal{X})$, which is doubly exponential in the number of clocks for PTA $\mathcal{A}$ [KNSW07]. As parameters in pPTAs do not affect the size of this result, we have the same complexity for pPTAs. Case studies have shown that for PTA, the state space is significantly smaller than the worst case [KNSW07]. Hartmanns and Kohlen [HK22] implemented the backward

reachability method in Modest Toolset for both minimal and maximal reachability properties. They compared their backward reachability implementation against the digital clocks implementation in the mcsta tool of the Modest Toolset. For time-bounded properties with a lower time bound, the backward reachability approach has a significantly smaller state space than the digital clocks approach. E.g., for their version of firewire, they get, for a time bound of 2000 time units, less than 100 states, whereas the digital clocks approach has 7670 states. However, for a larger time bound, the digital clocks approach outperforms backward reachability, e.g., for a time bound of 40000 time units, the backward reachability approach has when checking for the maximal probability, 10300 states. In contrast, the digital clocks approach has 7670 states.

7.5.3 Other methods

The main disadvantage of the digital clocks semantics is that they are only compatible with closed pPTAs. The backward reachability approach only allows us to calculate reachability properties. However, also other methods are developed to model check PTAs that do neither restrict the PTA nor the properties. We briefly discuss these methods and whether they are suitable for pPTAs.

region graph

Region graph | In the *region graph* approach [KNSS02], clock equivalence classes are considered. All clock valuations that satisfy the same constraints are grouped and used to build a *clock region*. This is equivalent to the symbolic states of backward reachability, where the clock regions are minimal. This leads to a relatively large state space, although this problem is tackled by other variants, like probabilistic time-abstract bisimulation [CHK08]. The algorithm is applicable to pPTA, and the proof is similar to that of digital clocks and backward reachability: The substitution of parameters before/after obtaining the region graph semantics is equivalent, and minimal/maximal reachability probabilities are preserved.

forward reachability

Forward reachability | As for backward reachability, *forward reachability* [KNSS02] semantics only considers relevant symbolic states. We denote the obtained pMDP by $[\![\mathcal{P}]\!]_{fr}$. However, this method performs a forward search from the initial state to the target states. The complexity of forward reachability is exponential in the number of clocks. The forward algorithm is generally faster than its backward equivalent [KNSW07], but it only provides upper (lower) bounds on the maximal (minimal) probability [DKN04]. So for pPTA $\mathcal{P}$:

$$\Pr^{l_I \xrightarrow{U} T}_{max,\mathcal{P}} \leq \Pr^{s_I \to T}_{max,[\![\mathcal{P}]\!]_{fr}} . \tag{7.2}$$

This makes forward reachability unsuited for parameter synthesis. Assume we want to check for a region R if the reachability probability is larger than λ. By applying the forward reachability semantics, we obtain a pMDP, which will give us an upper bound on the probability. Recall from Section 5.2 that if we want to check if the probability is larger than λ on the entire region, we consider a scheduler that minimizes over the parameter choices. We obtain a value $c \in \mathbb{R}_{\geq 0}$ such that $c \leq \Pr^{s_I \xrightarrow{U} T}_{max,[\![\mathcal{P}]\!]_{fr}}$. However, using Equation (7.2) we cannot obtain anything about the minimal value of $\Pr^{l_I \xrightarrow{U} T}_{max,\mathcal{P}}$.

stochastic game abstraction

Stochastic games | The *stochastic game abstraction* [KNP09] transforms a PTA into a 2-player stochastic game. This approach is (in practice) usually faster than both digital clocks and backward reachability [KNP09]. We conjecture that this approach can be applied directly to a pPTA resulting in a parametric stochastic game. However, as parameter synthesis is not implemented for this model in Storm or other tools, we are currently unable to analyze the obtained models.

7.6 Conclusion

This chapter introduced PTAs with parameterized probabilities and showed how we could obtain a finite pMDP that is equivalent to the pPTA for a given property. We can analyze the pMDP to, e.g., find monotonicity for one or more parameters in the corresponding pPTA. We focused on unbounded and time-bounded reachability properties. However, when using, e.g., the digital clocks semantics, we can also analyze expected total reward properties [2]. The usefulness of this approach is shown in [2; HK22]. Furthermore, we use the obtained pMDPs as benchmarks in Chapter 4. We see three possible directions for future work. First of all, one could extend pPTAs by having parameters in clock constraints as well. Secondly, one could consider pPTAs, in which transition probabilities can depend on clocks [Spr21]. Finally, as the stochastic games abstraction is currently the most competitive technique for PTA, developing techniques to analyze parametric stochastic games is an interesting direction for future work.

8 Conclusion and Future Work

8.1 Conclusion

Monotonicity in Markov models | We introduced monotonicity for both pMCs (Chapter 3) and pMDPs (Chapter 4) and showed how we could automatically infer monotonicity for both reachability and expected total reward properties for both model types. The algorithmic inference uses a state ordering to find local monotonicity for states. We obtain global monotonicity from local monotonicity in the following manner: if all states are locally monotonic increasing (decreasing), then the model is globally monotonic increasing (decreasing). For pMDPs, we additionally computed an optimal scheduler on-the-fly. We empirically evaluated the approach. For pMCs, we observed that if the solution functions can be used to find monotonicity, more monotonic parameters are found. However, for the cases where the solution function approach did not work, our approach can still often find monotonicity. Furthermore, we observed for pMCs that finding monotonicity for simple pMCs and pMCs with less parameters is slightly faster than the analysis for non-simple pMCs and pMCs with more parameters. There was no difference between acyclic and cyclic models. Finally, for both pMCs and pMDPs, we have seen that the use of assumptions helps if there are only a few assumptions but requires too much time and memory if more assumptions are needed.

Improving parameter synthesis with monotonicity | We also used monotonicity in pMCs to improve ε-optimal synthesis in pMCs. In the vanilla case, a divide-and-conquer approach with parameter lifting was used to synthesize an ε-optimal parameter instantiation. In our integrated approach, we extended the vanilla approach with monotonicity checking. The results are encouraging: where we can only solve a few instances for reachability properties with the vanilla approach, the integration with monotonicity checking allows us to check models with hundreds of parameters. Also, the results for the integrated approach are more precise. Moreover, the results for globally monotonic parameters are provably optimal. Furthermore, we discussed the influence of the heuristics on the performance of the integrated approach. For globally monotonic parameters, the heuristics have no influence. The heuristic `h-CurMax` allows us to solve more benchmarks, as the initial guess for `CurMax` is closer to the optimum. The heuristic `h-order` reduces the number of additional parameter invocations.

Gradient descent for pMCs | We showed how we can use GD to solve the feasibility problem for pMCs, i.e., finding an instantiation such that a given property holds. We experimented with several classical and adaptive GD methods. Furthermore, we considered different region restriction methods to ensure graph-preserving regions. The projection method, where the parameter instantiations are projected on the bounds of the region, performs best for pMCs. Finally, we evaluated our GD approach against the state-of-the-art methods PSO, QCQP, and SCP. From this empirical evaluation, we obtained that momentum-Sign GD outperforms the other GD methods. Furthermore, momentum-Sign GD outperforms PSO and QCQP on most benchmarks. While momentum-Sign GD and SCP perform alike.

Parametric probabilistic timed automata | We defined parametric PTAs, which have parameterized probabilities. Also, we showed how we could obtain a finite pMDP that is equivalent to the pPTA for a given property. To obtain this finite pMDP, we discussed the following methods: digital clocks, backward reachability, region graph, forward reachability, and stochastic game abstraction. We used the backward reachability approach to obtain pMDPs which are used in Chapter 4. We can, e.g., find monotonicity in the pPTA by checking for monotonicity on the pMDP.

8.2 Future Work

We conclude this dissertation by listing ideas and possible directions for future work.

Improving the reward order | If we cannot order a state in the reward order, assumptions are made. If many assumptions are necessary, this yields an explosion in the number of annotated reward orders. In the initial approaches to solving problems for pMCs, the focus was on computing a closed form for the solution function by, e.g., eliminating states. An interesting direction of future work is eliminating states that can not be ordered instead of making assumptions.

Finding monotonicity based on the modeling language | Another approach would be to define monotonicity on the modeling language (e.g., `prism` [KNP07; FKNP11] or `jani` [BDHH+17]). This way, one could find monotonicity for several instances of a benchmark at once. This approach is similar in spirit to the approach in [Spe18]; the main challenge is to find deterministic building blocks of the modeling language.

Improving synthesis methods with monotonicity | First of all, we can improve parameter lifting for pMDPs by using monotonicity checking. This extension follows

the lines of Chapter 5. Secondly, we can improve gradient descent by adding monotonicity checking as a preprocessing stage. As for parameter lifting (see Section 5.4), we can shrink the region using monotonicity. This can reduce the number of 'easy-parameters', i.e., we expect 'easy-parameters' to be monotonic, by shrinking the region for monotonic parameters, these parameters are fixed to their lower or upper bound, reducing the number of 'easy-parameters' present in GD.

Parameterized timed automata | For pPTAs, we see several directions for future work. First, we can extend pPTA by additionally allowing parameters in the delays by introducing parameters into the clock constraints of edge guards and location invariants. Although this easily leads to reachability in parametric timed automata becoming undecidable. One could take the approach of Hune et al. [HRSV02] and extend it with parametric probabilistic transitions. Secondly, one can also consider pPTA, in which the transition probabilities depend on clocks [Spr21].

Parametric stochastic games | Stochastic game abstraction is currently the most competitive technique for PTAs. Therefore, developing techniques to analyze parametric stochastic games is an interesting direction for future work. Also, monotonicity of parametric stochastic games belongs to this development.

A Algorithms

This chapter contains the full algorithms to obtain monotonicity for pMCs and pMDPs; the algorithms are discussed in Appendix A.1 and Appendix A.2, respectively. Both sections can be read independently from each other, i.e., Appendix A.2 does not depend on the explanation in Appendix A.1.

A.1 Checking Monotonicity in Markov Chains

Algorithm 10 contains the algorithm to automatically infer monotonicity on pMCs, as introduced in Section 3.5 (page 49). It takes as input a pMC and the bounds for the states. The algorithm computes a set of annotated reward orders $\preceq^{\mathsf{rew},\mathcal{A}}$, by keeping track of a queue consisting of the states we still need to handle for the order, the state assumptions made, and the annotated reward order (line 1). We loop over the items in the queue. We first pop an order from the queue (line 3). If the set S' of states we need to handle is empty, we annotated the order with the assumptions made (line 5) such that we can distinguish between states ordered by the graph rules and states ordered through assumptions. We add this annotated order to the result set. If the set S' of states to handle is not empty, we pick a state s topologically last w.r.t. $\sqcup$ (line 7). A simple breadth-first search can compute this. We now try to extend the order in the following manner:

1. **Extending the order based on bounds:** We first check if we can extend the order based on the bounds (lines 8-18). This reasoning is split into three parts:
   - lines 8-10, which build on Lemma 3.16 (page 60),
   - lines 11-14, which build on Lemma 3.17 (page 60), and
   - lines 15-18, which build on Theorem 3.18 (page 61).
2. **Extending the order based on graph-structure:** Now, we try to extend the order based on the graph rules for pMCs (lines 19-38). We can divide this into two parts:
   - lines 19-27, which build on Lemma 3.11 (page 50) and
   - lines 28-38, which build on Lemma 3.12 on page 54.

If the extension is successful, we add the remaining states to handle for the order ($S' \setminus \{s\}$, existing assumptions ($\mathcal{A}$), and extended order ($\preceq^{\mathsf{rew}}$) to the queue.

3. **Making assumptions:** If extending the order is unsuccessful, we pick two successors states we could not order and make three assumptions (see also Section 3.5.3 (page 55).) for these states. We push three copies of the order to the queue, each of it with its own new assumption.

A.2 Checking Monotonicity in Markov Decision Processes

Algorithm 11 contains the algorithm to automatically infer monotonicity on pMDPs, as introduced in Section 4.5 (page 83). Algorithm 11 takes as input a pMDP and the bounds for the states. It computes a set of pairs of annotated reward orders $\preceq^{\mathsf{rew},\mathcal{A}}$ and schedulers $\sigma^{\mathcal{A}^{Act}}$ by keeping track of a queue consisting of the annotated reward order, the annotated scheduler, the state and action assumptions, and the states we still need to handle for this order (line 1). We loop over the items in the queue and pop an order and scheduler from the queue (line 3) together with the set S' of states we need to handle and the state and action assumptions (denoted $\mathcal{A}$ and $\mathcal{A}^{Act}$, respectively). If S' is empty, we annotate the order and scheduler with, respectively, the state assumptions and action assumptions made (line 5). Through this annotation, we can distinguish between 1) states ordered by the graph rules and states ordered through state assumptions and 2) scheduler choices based on a local NLP and scheduler choices based on action assumptions. Finally, we add the annotated order and scheduler to the result set. If the set S' of states to handle is not empty, we pick a state s topologically last w.r.t. $\sqsubset$ (line 7). We chose s by applying a simple breadth-first search to the graph-structure of the pMDP. Next, we determine for s the set of minimizing actions $Act'(s)$ (line 8); more on this is found in Section 4.5.2 (page 85). We now try to extend the order in the following manner:

1. **Extending the order based on bounds:** We check if we can extend the order based on the bounds (lines 9-11). This reasoning is based on Lemma 3.16 (page 60) and holds for any number of actions available in $Act'(s)$.

2. **Extending the order for $|Act'(s)| = 1$**: We first set the scheduler for this state to the only action α in $Act'(s)$ (line 13). We then extend the order based on the bounds; this consists of the following two parts:

   - lines 14-17, which build on Lemma 3.17 (page 60) and
   - lines 18-21, which build on Theorem 3.18 (page 61).

   Next, we use the graph rules for pMCs (lines 22-41) to extend the order. This reasoning consists of:

- lines 22-30, which build on Lemma 3.11 (page 50) and
- lines 31-41, which build on Lemma 3.12 (page 54).

If the extension is successful, we push the remaining states to handle for the order ($S' \setminus \{s\}$), existing state and action assumptions ($\mathcal{A}/\mathcal{A}^{Act}$), extended order ($\preceq^{\mathsf{rew}}$), and scheduler ($\sigma$) to the queue. Otherwise, we pick two successors states we could not order and make three assumptions (see also Section 3.5.3 (page 55)) for these states. We push three copies of the order to the queue, each of it with its own new assumption (lines 42-46).

3. **Extending the order for $|Act'(s)| \geq 2$**: If there are two or more possible minimizing actions, we follow the approach of Algorithm 6 (page 91). We can divide this into two parts:
   - lines 47-54, which build on Lemma 4.23 on page 88 and
   - lines 55-65, which build on Lemma 4.24 on page 89.

   If the extension is successful, we push the remaining states to handle for the order ($S' \setminus \{s\}$), existing state and action assumptions ($\mathcal{A}/\mathcal{A}^{Act}$), extended order ($\preceq^{\mathsf{rew}}$), and scheduler ($\sigma$) to the queue.
4. **Making action assumptions**: If the extension of the order is unsuccessful and no state assumptions are made, we make for each possible action for s a copy of the order to the queue, each of it with its own new action assumption (line 68).

Algorithm 10 Construction of a Reward Order for a pMC

Input: pMC $\mathcal{D} = (S, s_I, \{\lightning\}, V, \mathbf{P})$ and bounds L_R, U_R s.t. $L_R(s) \leq \mathsf{ER}^{s \to \lightning}(\vec{u}) \leq U_R(s)$ for all $\vec{u} \in R$ and $s \in S$
Output: Result = set of annotated orders $\preceq^{\mathsf{rew},\mathcal{A}}$

Result $\leftarrow \emptyset$, Queue $\leftarrow \big(S' : S \setminus \{\lightning\}, \mathcal{A} : \emptyset, \prec^{\mathsf{rew}}: \{\lightning\} \times S'\big)$
while Queue not empty
  $S', \mathcal{A}, \preceq^{\mathsf{rew}} \leftarrow$ Queue.pop()
  if $S' = \emptyset$ **then**
    Result $\leftarrow$ Result $\cup \{\preceq^{\mathsf{rew},\mathcal{A}}\}$
  else
    select $s \in S'$ with s topologically last w.r.t. $\lightning$
    // Extending the order based on bounds
    for all $(s_1, s_2) \in U^2$ with $U = \mathrm{succ}(s) \cup \{s\}$ s.t. $s_1 \not\preceq^{\mathsf{rew}} s_2$ and $s_2 \not\preceq^{\mathsf{rew}} s_1$ **do**
      if $U_R(s_2) \leq L_R(s_1)$ **then**
        extend $\preceq^{\mathsf{rew}}$ with: $s_2 \preceq^{\mathsf{rew}} s_1$
    if $\mathrm{succ}(s) = \{s_1, s_2\}$ and $\forall \vec{u} \in R.\, r(s) \geq \mathbf{P}(s, s_1)(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ **then**
      extend $\preceq^{\mathsf{rew}}$ with: $s_2 \preceq^{\mathsf{rew}} s$
    else if $\mathrm{succ}(s) = \{s_1, s_2\}$ and $\forall \vec{u} \in R.\, r(s) \leq \mathbf{P}(s, s_1)(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ **then**
      extend $\preceq^{\mathsf{rew}}$ with: $s \preceq^{\mathsf{rew}} s_2$
    if $\mathrm{succ}(s) = \{s_1, \ldots, s_n\}$ and $n > 2$ **then**
      for all $j \in \{1, \ldots, n\}$ **do**
        if $\forall\, 1 \leq i < j.\, s_i \preceq^{\mathsf{rew}} s_j$ and $r(s) \geq \mathbf{P}(s, s_j)(\vec{u}) \cdot \big(U_R(s_j) - L_R(s_1)\big)$ **then**
          extend $\preceq^{\mathsf{rew}}$ with: $s_j \preceq^{\mathsf{rew}} s$
    // Extending the order based on graph-structure
    if $\preceq^{\mathsf{rew}}$ is a total order for $\mathrm{succ}(s)$ **then**
      if $r(s) = 0$ **then**
        if $\exists s' \in \mathrm{succ}(s)$ s.t. $\mathrm{succ}(s) \subseteq [s']$ **then**
          extend $\preceq^{\mathsf{rew}}$ with: $s \equiv^{\mathsf{rew}} \mathrm{succ}(s)$
        else
          extend $\preceq^{\mathsf{rew}}$ with: $\max(\mathsf{lb}(\mathrm{succ}(s))) \prec^{\mathsf{rew}} s$ and $s \prec^{\mathsf{rew}} \min(\mathsf{ub}(\mathrm{succ}(s)))$
      else
        extend $\preceq^{\mathsf{rew}}$ with: $\max(\mathsf{lb}(\mathrm{succ}(s))) \prec^{\mathsf{rew}} s$
      Queue.push$(S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}})$
    else if $\mathrm{succ}(s) = \{s_1, s_2\}$ and $\preceq^{\mathsf{rew}}$ is not a total order for $\{s_1, s_2\}$ and $\big(s_1 \equiv^{\mathsf{rew}} s$ or $s \prec^{\mathsf{rew}} s_1$ or $(s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0)\big)$ **then**

29: **if** $s_1 \equiv^{\mathsf{rew}} s$ **then**
30: **if** $r(s) = 0$ **then**
31: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \equiv^{\mathsf{rew}} s$
32: **else**
33: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$
34: **else if** $s \prec^{\mathsf{rew}} s_1$ **then**
35: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$
36: **else if** $s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0$ **then**
37: extend $\preceq^{\mathsf{rew}}$ with: $s \prec^{\mathsf{rew}} s$
38: Queue.push($S' \setminus \{s\}, \mathcal{A}, \preceq^{\mathsf{rew}}$)
39: **else**
// Making assumptions
40: pick $s_1, s_2 \in \mathsf{succ}(s)$ s.t. neither $s_1 \preceq^{\mathsf{rew}} s_2$ nor $s_2 \preceq^{\mathsf{rew}} s_1$
41: Queue.push($S', (\mathcal{A}_{\prec} \cup \{(s_1, s_2)\}, \mathcal{A}_{\equiv}), \preceq^{\mathsf{rew}}$ extended with $s_1 \prec^{\mathsf{rew}} s_2$)
42: Queue.push($S', (\mathcal{A}_{\prec}, \mathcal{A}_{\equiv} \cup \{(s_1, s_2)\}), \preceq^{\mathsf{rew}}$ extended with $s_1 \equiv^{\mathsf{rew}} s_2$)
43: Queue.push($S', (\mathcal{A}_{\prec} \cup \{(s_2, s_1)\}, \mathcal{A}_{\equiv}), \preceq^{\mathsf{rew}}$ extended with $s_2 \prec^{\mathsf{rew}} s_1$)
44: **return** Result

Algorithm 11 Construction of a reward order for a pMDP

Input: pMDP $\mathcal{M} = (S, s_I, \{\lightning\}, Act, V, \mathbf{P})$ and bounds L_R, U_R
s.t. $L_R(s) \leq \mathsf{ER}^{s \to \lightning}(\vec{u}) \leq U_R(s)$ for all $\vec{u} \in R$ and $s \in S$
Output: Result = set of pairs of annotated order and scheduler $\left(\preceq^{\mathsf{rew},\mathcal{A}}, \sigma^{\mathcal{A}^{Act}}\right)$

Result $\leftarrow \emptyset$, Queue $\leftarrow \left(S' : S \setminus \{\lightning\}, \mathcal{A} : \emptyset, \mathcal{A}^{Act} : \emptyset, \prec^{\mathsf{rew}}: \{\lightning\} \times S', \sigma(s) : \bot\right)$
while Queue not empty
  $S', \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma \leftarrow$ Queue.pop()
  if $S' = \emptyset$ **then**
    Result $\leftarrow$ Result $\cup \{(\preceq^{\mathsf{rew},\mathcal{A}}, \sigma^{\mathcal{A}^{Act}})\}$
  else
    select $s \in S'$ with s topologically last w.r.t. $\lightning$
    determine $Act'(s)$ based on NLP and σ
    // Extending the order based on bounds
    for all $(s_1, s_2) \in U^2$ with $U = \mathrm{succ}(s, Act') \cup \{s\}$ s.t. $s_1 \not\preceq^{\mathsf{rew}} s_2$ and $s_2 \not\preceq^{\mathsf{rew}} s_1$ **do**
      if $U_R(s_2) \leq L_R(s_1)$ **then**
        extend $\preceq^{\mathsf{rew}}$ with: $s_2 \preceq^{\mathsf{rew}} s_1$
    // Extending the order based on graph-structure for $|Act'(s)| = 1$
    if $Act'(s) = \{\alpha\}$ **then**
      $\sigma(s) \leftarrow \alpha$
      // Extending the order based for $|Act'(s)| = 1$
      if $\mathrm{succ}(s, Act') = \{s_1, s_2\}$ and $\forall \vec{u} \in R.\, r(s) \geq \mathbf{P}(s, s_1)(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ **then**
        extend $\preceq^{\mathsf{rew}}$ with: $s_2 \preceq^{\mathsf{rew}} s$
      else if $\mathrm{succ}(s, Act') = \{s_1, s_2\}$ and $\forall \vec{u} \in R.\, r(s) \leq \mathbf{P}(s, s_1)(\vec{u}) \cdot (U_R(s_2) - L_R(s_1))$ **then**
        extend $\preceq^{\mathsf{rew}}$ with: $s \preceq^{\mathsf{rew}} s_2$
      if $\mathrm{succ}(s, Act') = \{s_1, \ldots, s_n\}$ and $n > 2$ **then**
        for all $j \in \{1, \ldots, n\}$ **do**
          if $\forall\, 1 \leq i < j.\, s_i \preceq^{\mathsf{rew}} s_j$ and $r(s) \geq \mathbf{P}(s, s_j)(\vec{u}) \cdot \left(U_R(s_j) - L_R(s_1)\right)$ **then**
            extend $\preceq^{\mathsf{rew}}$ with: $s_j \preceq^{\mathsf{rew}} s$
      if $\preceq^{\mathsf{rew}}$ is a total order for $\mathrm{succ}(s, \alpha)$ **then**
        if $r(s) = 0$ **then**
          if $\exists s' \in \mathrm{succ}(s, \alpha)$ s.t. $\mathrm{succ}(s, \alpha) \subseteq [s']$ **then**
            extend $\preceq^{\mathsf{rew}}$ with: $s \equiv^{\mathsf{rew}} \mathrm{succ}(s, \alpha)$
          else

27: extend $\preceq^{\mathsf{rew}}$ with:
$\max(\mathsf{lb}(\mathsf{succ}(s,\alpha))) \prec^{\mathsf{rew}} s$ and $s \prec^{\mathsf{rew}} \min(\mathsf{ub}(\mathsf{succ}(s,\alpha)))$

28: **else**

29: extend $\preceq^{\mathsf{rew}}$ with: $\max(\mathsf{lb}(\mathsf{succ}(s,\alpha))) \prec^{\mathsf{rew}} s$

30: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma)$

31: **else if** $\mathsf{succ}(s,\alpha) = \{s_1, s_2\}$ and
$\preceq^{\mathsf{rew}}$ is not a total order for $\{s_1, s_2\}$ and
$\big(s_1 \equiv^{\mathsf{rew}} s$ or $s \prec^{\mathsf{rew}} s_1$ or $(s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0)\big)$ **then**

32: **if** $s_1 \equiv^{\mathsf{rew}} s$ **then**

33: **if** $r(s) = 0$ **then**

34: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \equiv^{\mathsf{rew}} s$

35: **else**

36: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$

37: **else if** $s \prec^{\mathsf{rew}} s_1$ **then**

38: extend $\preceq^{\mathsf{rew}}$ with: $s_2 \prec^{\mathsf{rew}} s$

39: **else if** $s_1 \prec^{\mathsf{rew}} s$ and $r(s) = 0$ **then**

40: extend $\preceq^{\mathsf{rew}}$ with: $s \prec^{\mathsf{rew}} s$

41: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma)$

42: **else**

// Making action assumptions

43: pick $s_1, s_2 \in \mathsf{succ}(s)$ s.t. neither $s_1 \preceq^{\mathsf{rew}} s_2$ nor $s_2 \preceq^{\mathsf{rew}} s_1$

44: Queue.push$(S', (\mathcal{A}_{\prec} \cup \{(s_1, s_2)\}, \mathcal{A}_{\equiv}), \mathcal{A}^{Act},$
$\preceq^{\mathsf{rew}}$ extended with $s_1 \prec^{\mathsf{rew}} s_2$)

45: Queue.push$(S', (\mathcal{A}_{\prec}, \mathcal{A}_{\equiv} \cup \{(s_1, s_2)\}), \mathcal{A}^{Act},$
$\preceq^{\mathsf{rew}}$ extended with $s_1 \equiv^{\mathsf{rew}} s_2$)

46: Queue.push$(S', (\mathcal{A}_{\prec} \cup \{(s_2, s_1)\}, \mathcal{A}_{\equiv}), \mathcal{A}^{Act},$
$\preceq^{\mathsf{rew}}$ extended with $s_2 \prec^{\mathsf{rew}} s_1$)

// Extending the order for $|Act'(s)| \geq 2$

47: **else if** $r(s) > 0$ and $\mathsf{lb}(\mathsf{succ}(s, Act'(s))) = S_I$ **then**

48: extend $\preceq^{\mathsf{rew}}$ with: $S_I \preceq^{\mathsf{rew}} s$

49: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma)$

50: **else if** $r(s) = 0$ and $\max(\mathsf{lb}(\mathsf{succ}(s, Act'(s)))) = s_l$ and
$\min(\mathsf{ub}(\mathsf{succ}(s, Act'(s)))) = s_u$ **then**

51: **if** $s_l \equiv^{\mathsf{rew}} s_u$ **then**

52: extend $\preceq^{\mathsf{rew}}$ with: $s \equiv^{\mathsf{rew}} s_u$

53: extend $\preceq^{\mathsf{rew}}$ with:
$s_l \prec^{\mathsf{rew}} s$ and $s \prec^{\mathsf{rew}} s_u$

54: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\mathsf{rew}}, \sigma)$

55: **else if** $\text{succ}(s, \alpha) = \{s_1, s_2\}$ for all $\alpha \in Act'(s)$ and
$\preceq^{\text{rew}}$ is not a total order for $\{s_1, s_2\}$ and
$\big(s_1 \equiv^{\text{rew}} s$ or $s \prec^{\text{rew}} s_1$ or $(s_1 \prec^{\text{rew}} s$ and $r(s) = 0)\big)$ **then**
56: **if** $s_1 \equiv^{\text{rew}} s$ **then**
57: **if** $r(s) = 0$ **then**
58: extend $\preceq^{\text{rew}}$ with: $s_2 \equiv^{\text{rew}} s$
59: **else**
60: extend $\preceq^{\text{rew}}$ with: $s_2 \prec^{\text{rew}} s$
61: **else if** $s \prec^{\text{rew}} s_1$ **then**
62: extend $\preceq^{\text{rew}}$ with: $s_2 \prec^{\text{rew}} s$
63: **else if** $s_1 \prec^{\text{rew}} s$ and $r(s) = 0$ **then**
64: extend $\preceq^{\text{rew}}$ with: $s \prec^{\text{rew}} s$
65: Queue.push$(S' \setminus \{s\}, \mathcal{A}, \mathcal{A}^{Act}, \preceq^{\text{rew}}, \sigma)$
66: **else**
// Making action assumptions
67: **for all** $\alpha \in Act'(s)$ **do**
68: Queue.push$(S', \mathcal{A}, \mathcal{A}^{Act} \cup \{\alpha\}, \preceq^{\text{rew}}, \sigma(s) \leftarrow \alpha)$
69: **return** Result

B Experiments

B.1 Setup

The monotonicity checking (Chapters 3 and 4), vanilla and integrated divide-and-conquer approach (Chapter 5), parameter space partitioning (used in Chapter 7), and gradient descent (Chapter 6) are all implemented in Storm and use the same underlying data structures and version (1.6.4) of Storm. The conversion from pPTA to pMDP is implemented in the Modest Toolset [HH14; HK22]. All experiments were executed on a single-core Intel Xeon Platinum 8160 CPU. We neither use any parallel processing nor randomization. We used a time-out of 2 hours (7200s) and a memory limit of 32GB. We used value iteration to solve the MDPs and sound value iteration to solve the SGs.

B.2 Benchmarks

In this section, we introduce the benchmarks used for the empirical evaluation of the given chapters. We first discuss pMCs as used in Sections 3.6 and 5.6. Next, we introduce the pMDPs (and pMDPs obtained from pPTAs) are used in Section 4.6. Finally, we discuss some additional benchmarks used to evaluate the gradient-descent approach (Section 6.7).

B.2.1 pMC benchmarks

In the following, we describe the pMC benchmarks we consider. As the existing pMC benchmarks from the literature have a limited number of parameters, we also consider pMCs we obtained through translating POMDPs [JJWQ+18; Jun20]. All model characteristics of the pMCs are found in Tables B.1 and B.2. For each benchmark, we denote the instance and whether it is cyclic or acyclic (**A/C**) before preprocessing. Furthermore, we denote the number of parameters (|**V**|) and the number of states and transitions (**#state** and **#trans**) before and after the preprocessing. Finally, we show the model building time (in seconds).

Considered pMCs | We consider all existing benchmarks with pMC and non-trivial reachability properties from [HKPQ+19], [CK18], and the Param website [HHZW19]. The pMC benchmark sets craps [BK08], nand [NPKS05], and Herman [Her90; KNP12a]

Table B.1: pMC benchmark characteristics for reachability properties

benchmark	instance	A/C	$\lvert V \rvert$	#state	#trans	#state	#trans	model building
brp	(2,16)	A	2	613	803	99	195	<1
	(15,4096)			847890	1163267	131075	262147	19
	(15, 8192)			1695762	2326531	262147	524291	39
	(15, 16384)			3391506	4653059	524291	1048579	78
crowds	(3,5)	C	2	1145	1955	7	11	<1
	(6,5)			15233	26573	13	23	<1
evade	(1,2,0,1)	C	40	129	249	43	116	<1
	(1,2,1,1)		80	257	497	83	228	<1
	(1,2,3,1)		160	513	993	163	452	<1
gambler	(6400)	C	1	7041	14080	7041	14080	<1
	(102400)			112641	225280	112641	225280	3
	(204800)			225281	450560	225281	450560	6
maze	(25)	C	24	360	660	51	101	<1
	(1000)		999	14985	26985	2001	4001	9
	(10000)		9999	149985	269985	20001	40001	21
message authentication	(30)	A	2	151	240	5	7	<1
	(15360)			76801	122880	5	7	2
nrp	(5,1)	A	5	56	75	18	37	<1
	(10,1)		10	186	250	58	122	<1
	(15,1)		15	391	525	123	257	<1
zeroconf	(6400)	C	2	6404	12805	6403	12804	<1
	(12800)			12804	25605	12803	25604	<1
	(409600)			409604	819205	409603	819204	11

are not monotonic; this can easily be checked by sampling over the parameter space. We thus omit these benchmarks. The benchmark haddad-monmege [HM18] has a reachability property and contains only a single non-sink state after preprocessing; it is trivially monotonic and thus omitted for analysis. This gives us the following benchmarks:

- brp [DJJL01]: The bounded retransmission protocol sends a file in a number of chunks. The number of retransmission of each chunk is bounded. The goal is to successfully send all chunks of a file. The parameters represent the reliability of the channels over which the chunks are sent.

- crowds [RR98]: The Crowds protocol provides a mechanism to ensure anonymity on the internet. The user's identity is hidden by randomly rerouting messages, so even if someone sees a message being sent by a particular user, he can never be sure that the user is the actual sender. The goal is to find the probability that the sender is identified using this protocol. The parameters represent how the message is randomly rerouted and the probability that a member is evil.

- egl [KNP12b]: The egl protocol is a randomized protocol for signing contracts introduced by Even, Goldreich, and Lempel. The goal is to sign the contract only if the other signs it. We check the expected number of messages a person needs to send before the protocol is finished. The parameters represent the way the algorithm is randomized.

Table B.2: pMC benchmark characteristics for expected total reward properties

benchmark	instance	A/C	$\|V\|$	#state	#trans	#state	#trans	model building
egl	(5,2)	A	10	33790	34813	54	73	<1
	(6,2)		12	159742	163837	71	94	3
	(7,2)		14	737278	753661	90	117	16
4x4-grid	(1)	C	3	47	106	47	106	<1
	(2)		11	155	335	139	302	<1
	(4)		26	550	1138	393	823	<1
	(8)		55	1460	2971	1022	2094	<1
	(16)		176	3522	7108	2521	5105	<1
	(32)		393	8372	16821	5242	10508	<1
	(64)		846	19766	39622	11758	23605	2
maze	(1)	C	15	41	82	41	82	<1
	(2)		58	135	281	134	278	<1
	(4)		131	478	978	358	737	<1
	(8)		296	1268	2569	918	1868	<1
	(16)		841	3058	6160	2253	4549	<1
	(32)		2049	7268	14591	4812	9678	1
	(64)		4373	17158	34382	10682	21429	7
network_dp	(2,1,1)	C	9	229	435	98	229	<1
	(2,3,1)		84	855	4090	552	2910	<1
	(4,1,1)		36	469	4161	201	807	<1
	(4,3,1)		216	1741	10548	1230	7708	<1
network_ps	(2,1,1)	C	14	157	291	26	66	<1
	(2,5,1)		143	1871	6531	1085	3731	<1
	(4,1,1)		41	469	4161	157	607	<1
	(4,5,1)		302	3965	15817	2208	12450	<1
samplerocks	(1,1)	C	37	561	1092	282	565	<1
	(2,1)	C	189	2485	4848	1129	2252	<1

- gambler [CK18]: The gambler tries to increase its wealth by tossing a biased coin. The profit is at most 10% of the initial wealth. After each coin toss, the gambler either wins or loses 1 euro. We are interested in the probability of eventually winning 10% of the initial wealth. The parameter represents the bias of the coin.
- message authentication [FGT11]: The message authentication protocol tries to authenticate a message to a random user. It is successful if we 1) manage to log in and 2) successfully send the message. We are interested in the probability of eventually being successful. The parameters represent the probability of logging in successfully and the probability of successfully transmitting a message.
- zeroconf [BSHV03]: The zeroconf protocol allows a device to obtain a unique IP address without needing any manual intervention or a special host. We assume one concrete device is trying to obtain an IP address when an unknown number of other devices already have an IP address. We consider the probability of eventually obtaining an IP address. The parameters represent the probability of picking a new IP address and the probability of a message getting lost.

Considered pMCs obtained from POMDPs | To obtain pMCs with more parameters, we consider POMDPs, which we translated into pMCs [JJWQ+18; Jun20] (see also Appendix C). We took at least one variant of all POMDP benchmarks with reachability

or expected total reward properties of which we had a PRISM [KNP11] model available from [NPZ17; BJKQ20]. Below we discuss these models and their properties. Note that for all benchmarks, the parameters represent the scheduler choices at states with the same observation in the POMDPs (see Appendix C).

- evade [EGL82]: This benchmark is a modification of the drone benchmark (see Appendix B.3). The drone moves around the grid and wants to reach a target location while avoiding a randomly moving obstacle. The obstacle is only visible within a limited radius. Next to moving around, the drone can also scan the environment and stay in its current position. We are interested in the probability of eventually safely reaching the target.
- 4x4-grid [CKL94; LCK95; BJKQ20]: This benchmark has a 4x4-grid where the goal for the robot is to reach the target without reaching a bad state. We consider the expected number of steps needed to reach the target.
- maze [McC92; LCK95; BJKQ20]: In this benchmark, a mouse has to find the cheese in a maze. We consider two benchmark versions. First of all, we consider the version in which there are unsafe states in which the mouse dies; we want to find the probability of eventually reaching the cheese. Secondly, we consider the version in which the probability of finding the cheese is 1; we consider the expected number of steps the mouse needs.
- network_dp and network_ps [YMZ11]: The network_dp and network_ps benchmarks are an unitization example. The network contains two channels, and the packets are divided over these channels. For the network_dp benchmark, we consider the expected number of dropped packets (“dp”). Whereas for network_ps, we consider the number of successfully delivered packets (“ps”).
- nrp [LMT05]: This benchmarks models the non-repudiation protocol. Repudiation is the denial of being involved in (a part of) the communication. E.g., if the sender of a message denies being the send, or the receiver of a message denies having received the message. This protocol tries to guarantee fairness without needing a trusted party. We are interested in the probability that some malicious recipient denies that it received a message.
- samplerocks [SS04; BG09; BJKQ20]: The samplerocks benchmarks model a rover science exploration. The rover has a limited amount of fuel and can only increase this by sampling rocks. The problem, however, is that the rover does not know in advance whether a rock will increase the fuel limit or not. We are interested in the expected number of steps to reach a given position.

We amended the properties from [BJKQ20] such that a target state is almost surely reached; this ensures that the expected total reward is finite.

> **Remark B.1** We leave out the drone [BJKQ20] and refuel [BJKQ20] benchmarks, as they are similar to evade and samplerocks, respectively.

B.2.2 pMDP benchmarks

In the following, we describe the pMDP benchmarks we consider. Next to existing pMDP benchmarks from the literature, we consider the underlying pMDP of pPTAs, which we obtained by parameterizing PTAs from the literature. We translated these pPTAs into pMDPs by using the backward reachability semantics as implemented in the Modest Toolset. Tables B.3 through B.5 show the model characteristics of the benchmarks described above. It lists the benchmark and their instances. Furthermore, we list whether it is acylic (**A**) or cyclic (**C**), the number of parameters $|V|$, and the size of the pMDP (**#state** and **#trans**) before and after preprocessing. **Model building** includes the time (in seconds) for construction, default preprocessing by Storm, and bisimulation minimization.

Table B.3: pMDP benchmark characteristics for reachability properties

benchmark	instance	A/C	$\|V\|$	#state	#trans	#state	#trans	model building
consensus	(2,1,min)	C	2	144	252	53	128	<1
	(2,2,min)			272	492	125	304	<1
csma	(2,2,min/max)	A	2	1038	1282	12	20	<1
	(2,4,min/max)			7958	10594	18	28	<1
	(2,6,min/max)			66718	93072	24	36	<1
	(3,2,min/max)		3	36850	55862	150	1074	2
	(3,4,min/max)			1460287	2396727	556	5292	5512
	(3,6,min/max)							MO
	(2,2,min2)		2	966	1204	9	16	<1
	(2,4,min2)			7670	10216	15	24	<1
	(2,6,min2)			64828	90304	21	32	1
	(3,2,min2)		3	33808	51830	25	60	<1
	(3,4,min2)			1430119	2342871	251	906	27
	(3,6,min2)							MO
zeroconf	(20,2,true,min)	C	2	659	965	40	85	<1
	(20,4,true,min)			1077	1581	48	101	<1
	(20,6,true,min)			1495	2197	56	117	<1
	(20,8,true,min)			1913	2813	64	133	<1
	(20,2,false,min/max)			88858	203550	27605	81451	409
	(20,4,false,min/max)			306585	703189	96574	296585	6552
	(20,6,false,min/max)							TO
	(20,2,true,max)			659	965	21	42	<1
	(20,4,true,max)			1077	1581	29	58	<1
	(20,6,true,max)			1495	2197	37	74	<1
	(20,8,true,max)			1913	2813	45	90	<1
	(20,2,false,min/max)			88858	203550	19827	58640	409
	(20,4,false,min/max)			306585	703189	79163	243960	6552
	(20,6,false,min/max)							TO

Considered pMDPs | In order to obtain parametric pMDPs, we first of all parameterized existing pMDPs of [HKPQ+19] by replacing all simple probabilistic transitions

Table B.4: pMDP benchmark characteristics for expected total reward properties

benchmark	instance	A/C	|V|	#state	#trans	#state	#trans	model building
consensus	(2,1,min/max)	C	2	144	252	74	166	<1
	(2,2,min/max)			272	492	154	350	<1
csma	(2,2,min/max)	C	2	1038	1282	16	29	<1
	(2,4,min/max)			7958	10594	22	41	<1
	(2,6,min/max)			66718	93072	29	54	6
	(3,2,min/max)		3	36850	55862	194	1244	5
	(3,4,min/max)							MO
eajs	(2,30,max)	A	5	914	1446	17	28	<1
	(2,35,max)			1333	2191	77	132	<1
	(2,40,max)			1870	3089	170	282	<1
	(3,30,max)		9	5371	9318	29	60	<1
	(3,35,max)			9052	16185	110	229	<1
	(3,40,max)			13052	23841	370	778	<1
	(4,30,max)			33148	62830	30	71	1
	(4,35,max)			57664	112825	148	385	3
	(4,40,max)			82339	163885	866	2422	6
	(5,30,max)		11	85079	176650	36	97	4
	(5,35,max)			152163	327482	202	639	10
	(5,40,max)			219523	481502	1258	4225	28
	(6,30,max)		13	174784	391482	42	127	9
	(6,35,max)			317434	741017	262	981	25
	(6,40,max)			460509	1097967	1868	7393	93
firewire	(3,min)	C	1	611	718	42	98	<1
	(10,min)			646	865	84	203	<1
	(15,min)			671	970	114	970	<1
	(20,min)			696	1075	144	1075	<1
	(25,min)			721	1180	174	1180	<1
	(3,max)			611	718	43	718	<1
	(10,max)			646	865	85	865	<1
	(15,max)			671	970	115	970	<1
	(20,max)			696	1075	145	355	<1
	(25,max)			721	1180	175	1180	<1

(see Section 2.2.3 (page 17)) by parametric ones. We omit the following benchmarks:

- ij [IJ90], dining philosophers [LR81], pnueli-zuck [PZ86], and rabin [Rab82]: For these benchmarks, we obtain, after preprocessing, a reachability probability of 1, which is trivially monotonic.
- pacman [JKJB18]: After preprocessing, all parameters are discharged, so the solution function is trivially monotonic.
- resource-gathering [BN08]: For this benchmark, the associated properties are not supported by the monotonicity checking implementation in Storm.
- wlan [KNS02]: All probabilistic transitions are non-simple.

This leaves us with the following benchmarks:

- consensus [AH90; KNS01]: This is a coin-flipping protocol in which coins are flipped until a consensus is reached. We use biased coins and check for the (minimal) probability of reaching a consensus and the expected number of steps to reach a consensus. The parameters represent the bias of the coins.

Table B.5: Characteristics for the underlying pMDPs of pPTAs

benchmark	instance	A/C	|V|	#state	#trans	#state	#trans	model building
AODV	(1/2/3/4/5,br,no,delivery)	A	1/2/3/4/5	3562	9522	968	2964	<1
	(1/2/3/4/5,br,no,nextHop)			5887	16098	1721	5331	1
	(1/2/3/4/5,dc,no,nextHop)			6698	18716	1734	5622	6
	(1/2/3/4/5,dc,no,delivery)			7418	20100	964	3024	5
	(1/2/3/4/5,br,delivery)			3562	9522	968	2964	<1
	(1/2/3/4/5,br,nextHop)			5887	16098	1721	5331	<1
	(1/2/3/4/5,dc,nextHop			147586	338508	1734	5622	62
	(1/2/3/4/5,dc,delivery)			148306	339892	964	3024	54
brp (pPTA)	(S1/S2,br,D)	A	1/2	16904	132328	10614	56285	36
	(S1/S2,br,1)			1134	2963	792	1905	<1
	(S1/S2,br,2)			801	2115	577	1360	<1
	(S1/S2,br,3)			889	2336	631	1503	<1
	(S1/S2,br,4)			10	13	6	9	<1
	(S1/S2,dc,D)			169572	198224	5631	11160	74
	(S1/S2,dc,1)			2319	2953	162	258	<1
	(S1/S2,dc,2)			3878	4512	162	258	<1
	(S1/S2,dc,3)			3259	3893	152	242	<1
	(S1/S2,dc,4)			3970	4604	6	9	<1
firewire (pPTA)	(br,400/600/800/1000)	A	1	7	13	5	7	<1
	(dc,400)			322374	326702	4	8	3
	(dc,600)			611374	636002	404	1208	6
	(dc,800)			908492	1003793	724	2166	10
	(dc,1000)			1442516	1955116	724	2166	18
nrp (pPTA)	(honest,ns/s,br,20)	A	1	82	312	41	217	<1
	(honest,ns/s,br,40)			162	1022	81	941	<1
	(honest,ns/s,br,60)			242	2132	121	2011	<1
	(honest,ns/ss,br,80)			322	3642	161	3481	<1
	(honest,ns/s,br,100)			402	5552	201	5351	2
	(honest,ns,dc,20)			195	360	71	207	<1
	(honest,ns,dc,40)			395	740	151	447	<1
	(honest,ns,dc,60)			595	1120	231	1120	<1
	(honest,ns,dc,80)			795	1500	311	927	<1
	(honest,ns,dc,100)			995	1880	391	1167	<1
	(malicious,ns/s,br,5)	A	3	62	200	29	200 <1	
	(malicious,ns/s,br,10)			591	4626	385	2646	<1
	(malicious,ns/s,br,20)			2181	41036	1390	20260	3
	(malicious,ns/s,br,event)	C		115	463	63	242	<1
	(malicious,ns,dc,5)	A		425	888	20	51	<1
	(malicious,ns,dc,10)			1814	4541	87	284	<1
	(malicious,ns,dc,20)			4984	13391	227	824	<1
	(malicious,ns,dc,event)	C		318	886	27	79	<1
zeroconf (pPTA)	(br,100)	A	2	13	23	12	22	<1
	(br,150)			41	87	33	75	<1
	(br,200)			202	721	129	483	<1
	(br,wrong)	C		25	45	21	41	<1
	(dc,100)	A		10621	13060	27	68	<1
	(dc,150)			31724	39747	106	255	<1
	(dc,200)			56527	71613	524	1280	1
	(dc,wrong)	C		498	641	31	75	<1

- csma [KNSW07]: The carrier sense multiple access with collision detection protocol is a network protocol for carrier transmission. A sender/receiver using this protocol listens to check whether a shared channel for transmission is busy and only starts transmitting a message once the channel is free. If there is a collision, it stops transmitting and sends a collision signal. It then waits for a random time interval before retransmission. We check for the minimal and maximal probability of submitting all messages without collision. Furthermore, we consider the minimal

probability that it eventually happens that we send the next message too soon (denoted in the table by min2). Finally, we consider the minimal and maximal expected time to deliver all messages. The parameters are used to determine how long we wait before retransmission.

- eajs [KBCD+18]: This benchmark represents an energy-aware job scheduling logarithm. Two processes need to enter a critical section, and each process has its own deadline. The scheduler grants access to the critical section after a request of a process. Each process consumes energy, especially if a task is performed. A task can also be performed at double speed at the cost of triple energy consumption. We want to compute the maximal expected number of tasks completed before the battery is empty. The time a tasks takes is randomized.

- firewire [SV99; KNS03]: This benchmarks concerns the Tree Identify Protocol of the IEEE 1394 High Performance Serial Bus. The bus is used to transport video and audio signals with a network of multimedia devices. It works as a leader election protocol. After each bus reset, all nodes in the network have equal states, and a leader has to be chosen to act as the bus manager. We use an abstraction of the original model [SV99]. We consider the minimal and maximum expected time needed to find a new manager. The parameter represents the probability of being in a fast or slow, i.e., the delay we wait before sending the next message to find a new manager.

- zeroconf (MDP) [BSHV03]: This benchmark is equivalent to the pMC. However, we now have a non-deterministic choice between which of the received messages we consider first. We consider the non-timed version of the MDP. We are interested in the minimal and maximal probability of obtaining an IP address. As before, the parameters model the reliability of the channels.

Considered pMDPs obtained from pPTAs | To obtain more pMDPs, we consider pPTAs which we translate into pMDPs using the Modest Toolset [HH14; HK22]. Below we discuss the benchmarks. For all benchmarks, the available properties are reachability properties.

- AODV [Eli03; KK20]: The Ad-hoc On-demand Distance Vector (AODV) routing protocol is one of the four protocols standardized by the IETF MANET Working Group. AODV finds routes on demand whenever needed. It intends to establish a route between the source and destination node (route discovery) by injecting a data packet. The parameters represent the uncertainty factors such as message loss and message collision(s). We are interested in the probability of package delivery in a network using AODV.

Table B.6: Model characteristics of GD benchmarks

	benchmark	bound	instance	#state	#trans	$\lvert V \rvert$
Reachability Probabilites	child	≤ 0.43	(240)	243	3277	223
	drone	≥ 0.85	(5,1)	3678	27376	756
			(5,2)	3678	27376	2640
	4x4-grid -avoid	≥ 0.9	(5)	1216	2495	99
			(10)	4931	9990	399
	hailfinder	≥ 0.145	(2000)	1540	324982	1249
	newgrid	≥ 0.99	(8,10)	30191	60410	399
			(15,10)	98441	196910	399
	nrp	≤ 0.001	(16,2)	787	1602	95
			(16,5)	5806	11685	704
	refuel	≥ 0.35	(5,3)	1564	4206	452
			(8,3)	7507	21468	794
Expected Reward	4x4-grid	≤ 4.2	(5)	1410	2879	99
			(10)	5780	11659	399
	maze	≤ 6	(15)	5340	10799	2624
			(50)	61000	121799	29749
	network_dp	≤ 3.5	(8,5,1)	2822	69688	888
	network_ps	≤ 0.1	(8,5,1)	397	2837	140
	samplerocks	≤ 40	(8)	11278	25205	2844

- brp (pPTA) [HH09] and nrp (pPTA) [MR99; NPS13; KNP19]: These benchmarks are the timed version of brp and nrp as discussed in Appendix B.2.1. For nrp (pPTA) we use the honest and malicious variant.
- firewire (pPTA) [SV99] and zeroconf (pPTA) [CAG05; KNPS06]: These benchmarks are the timed version of the firewire and zeroconf (MDP) benchmarks as discussed in Appendix B.2.2.

B.3 Benchmarks for GD

In the following, we describe the benchmarks we consider in Section 6.7. Table B.6 shows the characteristics of the benchmarks. The first seven benchmarks in this table consider reachability properties, whereas the latter four consider expected total rewards. The table includes the required property (**bound**) and the instance of the benchmark (**instance**). For each benchmark, we denote the number of states (**#state**), transitions (**#trans**), and parameters ($|V|$) after preprocessing.

Considered pMCs obtained from POMDPs | For GD, we consider the following benchmarks 4x4-grid, maze, network_dp, network_ps, nrp, and samplerocks, which are already discussed in Appendix B.2.1. Furthermore, we additionally consider the following pMCs obtained from POMDPs:

- drone [BJKQ20]: The drone moves around the grid and wants to reach a target location while avoiding a randomly moving obstacle. The obstacle is only visible

within a limited radius. We are interested in the probability of eventually safely reaching the target.

- 4x4-grid-avoid and newgrid [CKL94; LCK95; BJKQ20]: These benchmarks contain modifications to the grid world in 4x4-grid.
- refuel [BJKQ20]: As for the samplerocks benchmark, we search for a plan to arrive at a target. The movement in this benchmark is uncertain, and the current position is only partially observable. To move around, we need energy, which can be refilled at recharging stations. All obstacles are at fixed points on the grid. We are interested in the probability of reaching the target without hitting an obstacle or getting out of fuel.

Considered pMCs obtained from pBNs | We also consider two pMCs obtained from parametric Bayesian networks with a large number of parameters [SK20; SK21; Scu21]:

- child [CCW92; SK20]: This benchmark represents a model to determine whether a baby who turned blue at birth has a given disease. It is used to help a doctor to provide a proper diagnosis. Based on data, conditional probability tables are generated. The data can, however, be incomplete. Therefore, we parameterized the probabilities in the tables. We are interested in the probability that the CO2 report returns true.
- hailfinder [ABEM+96; SK20]: The hailfinder benchmark models a weather forecasting system that uses a combination of meteorological data, models, and expert judgment to predict the weather in Northeastern Colorado. All information is combined in conditional probability tables, for which the data can be incomplete. Therefore, we parameterize the probabilities in the table. For this benchmark, we consider the probability that the dew point occurs on a given day.

C Obtaining pMCs from POMDPs

Summary | In this chapter, we describe how can obtain pMCs from partially observable MDPs (POMDPs) [KLC98; RN20]. These pMCs can be analyzed using the techniques discussed in this dissertation. The results for the pMC can than be used to find a controller in the POMDP.

Origins | This chapter contains the existing work of Junges as found in [JJWQ+18] and [Jun20, Ch.7].

Background | We use pMCs and MDPs as introduced in Section 2.2.

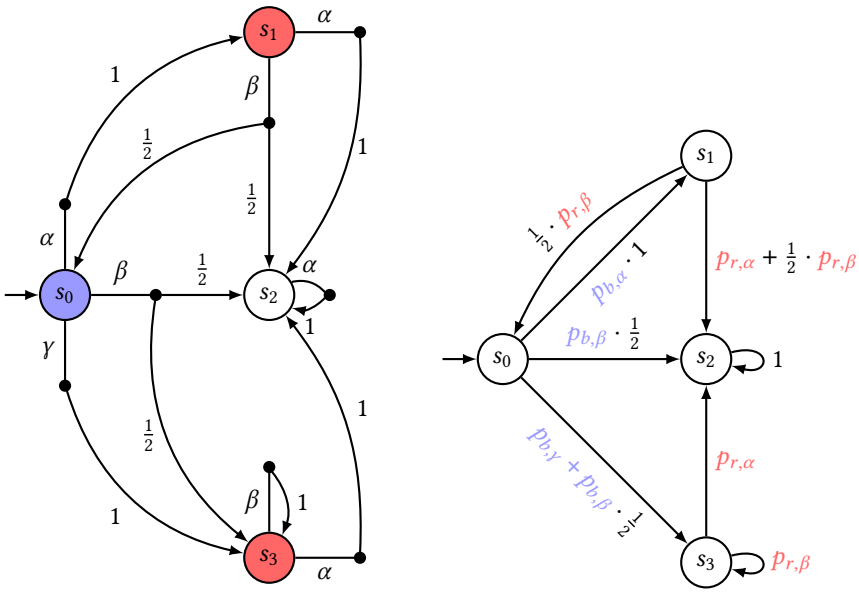
C.1 Partially Observable MDPs

partially observable MDP

Partially observable MDPs (POMDPs) are models that extend probabilistic and non-deterministic behavior with partial observability [Åst65; KLC98; Spa12; RN20]. Rather than knowing precisely the state of the process, one only knows the state's observation. As multiple states may have the same observation, the current state cannot be uniquely identified. POMDPs are key models in AI and planning [RN20], e.g., for robots that only have a partial perception of their environment. POMDP controllers (aka. schedulers or policies) try to resolve the non-determinism based on the (a part of) the observation history rather than on the state history (as is the case in MDPs). This limited information makes optimal decision making harder. Indeed finding a controller to maximize reachability probabilities is undecidable for POMDPs, whereas this is a polynomial-time problem for MDPs.

Definition C.1 A partially observable MDP (POMDP) is a tuple $\Omega = (\mathcal{M}, \mathsf{Z}, \mathtt{Obs})$, with underlying MDP $\mathcal{M}$ of Ω, a finite set Z of observations, and an observation function $\mathtt{Obs} \colon S \to \mathsf{Z}$.

We let $\mathtt{Obs}(s)$ denote the set of observations at state $s \in S$. For states with the same observations, we have the same available actions, i.e., $\mathtt{Obs}(s) = \mathtt{Obs}(s')$ implies $Act(s) = Act(s')$ for $s, s' \in S$.



(a) POMDP Ω (b) pMC $\mathcal{D}_6$

Figure C.1: From POMDPs to pMCs [Jun20, p. 182].

Example C.1 Figure C.1a shows POMDP Ω. The state colors match the observations at a state. We cannot distinguish between states s_1 and s_3 as the observation at those states is the same. Note that the set of available actions is also equivalent at states with the same observations.

C.2 Obtaining pMCs from POMDPs

In this section, we show how we can obtain a pMC from a POMDP. We can then analyze the pMC with the techniques discussed in this dissertation and map this result back to the original POMDP (see also Appendix C.3).

·corresponding pMC

Definition C.2 For POMDP $\Omega := (\mathcal{M}, \mathsf{Z}, \mathtt{Obs})$ with $\mathcal{M} = (S, s_I, T, Act, \mathbf{P})$, we define the *corresponding pMC* $\mathcal{D}_\Omega = (S, s_I, T, V, \mathbf{P}')$ with

- $V := \{p_{z,\alpha} \mid z \in \mathsf{Z}, \alpha \in Act(z)\}$, and

- $\mathbf{P}(s,s') \coloneqq \begin{cases} \mathbf{P}(s,\alpha,s') & \text{if } Act(s) = \{\alpha\} \\ \sum_{\alpha \in Act(\mathtt{Obs}(s))} p_{\mathtt{Obs}(s),\alpha} \cdot \mathbf{P}(s,\alpha,s') & \text{otherwise .} \end{cases}$

Example C.2 Reconsider our running example POMDP Ω of Figure C.1a. Figure C.1b shows the pMC $\mathcal{D}_6$ obtained from Ω via Definition C.2. When observing a red state, s_1 or s_3, with probability $p_{r,\alpha}$ action α is taken, and with probability $p_{r,\beta}$ action β is taken. At state s_0, action α (β, γ) is taken with probability $p_{b,\alpha}$ ($p_{b,\beta}$, $p_{b,\gamma}$).

C.3 Obtaining POMDPs from pMCs

We now show how we can use the result for the pMC to obtain an optimal controller in the pMC. Moreover, we show how we can obtain a POMDP from any *simplex* pMC.

Definition C.3 Let $\mathcal{D}$ be a pMC with parameters V, and let $V = V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_m$ be a disjoint partitioning of the parameters. $\mathcal{D}$ is *simplex* if

simplex

- it has *grouped parameters*, i.e., for all $s \in S$: $\mathsf{occur}(s) = V_i$ for some $1 \leq i \leq m$, and
- for all transition probabilities we have:

$$\mathbf{P}(s,s') = c \text{ or } \mathbf{P}(s,s') = c_1 \cdot p_1 + c_2 \cdot p_2 + \ldots + c_k \cdot p_k \text{for } c, c_i \in \mathbb{N},\ p_i \in V\ .$$

We now define how we can obtain a POMDP from a given pMC.

Definition C.4 Let $\mathcal{D} = (S, s_I, T, V, \mathbf{P})$ be a pMC such that if $\mathsf{occur}(s) \cap \mathsf{occur}(s') \neq \emptyset$ then $\mathsf{occur}(s) = \mathsf{occur}(s')$. Let $V_1, \ldots, V_n$ be the parameter groups of $\mathcal{D}$. We define the *corresponding POMDP* $\Omega_{\mathcal{D}} = (\mathcal{M}, \mathtt{Z}, \mathtt{Obs})$ with underlying MDP $\mathcal{M} = (S, s_I, T, Act, \mathbf{P}')$, where $Act \coloneqq \{\alpha_p \mid p \in V\} \cup \{\bot\}$,

·corresponding POMDP

$$\mathbf{P}'(s,\alpha,s') \coloneqq \begin{cases} c_i & \text{if } \mathbf{P}(s,s') = c_1 \cdot p_1 + c_2 \cdot p_2 + \ldots + c_k \cdot p_k \text{ and } \alpha = \alpha_{p_i} \\ \mathbf{P}(s,s') & \text{if } \mathsf{occur}(s) = \emptyset \text{ and } \alpha = \bot \\ 0 & \text{otherwise} \end{cases},$$

and $\mathtt{Z} \coloneqq V_1, \ldots, V_n$ with $\mathtt{Obs}(s) = \mathsf{occur}(s)$.

Example C.3 Reconsider pMC $\mathcal{D}_6$ of Figure C.1b. Through Definition C.4 we obtain POMDP Ω of Figure C.1a.

To obtain an ε-optimal (probabilistic) scheduler for a given POMDP, we can now transform the POMDP into a pMC, solve the ε-optimal synthesis problem, and map the found parameter instantiation back to the actions in the POMDP.

Remark C.1 Above we discuss how to obtain a pMC from a POMDP with no memory; the extension to history-dependent schedulers can be found in [Jun20, Sec. 7.1.3].

Declaration of authorship

Eidesstattliche Erklärung

According to §12 (3) of the doctoral regulations of RWTH Aachen University, I am required to give the following declaration.

I, Jip Josephine Spel,
declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.
Hiermit erkläre ich an Eides statt / I do solemnly swear that:

1. This work was done wholly or mainly while in candidature for the doctoral degree at this faculty and university;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others or myself, this is always clearly attributed;
4. Where I have quoted from the work of others or myself, the source is always given. This thesis is entirely my own work, with the exception of such quotations;
5. I have acknowledged all major sources of assistance;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published before as indicated in Section 1.3 on page 4.

Jip Josephine Spel, 2nd January 2024, Aachen

Bibliography

References

[AAPP21] É. André, J. Arias, L. Petrucci and J. van de Pol. '*Iterative Bounded Synthesis for Efficient Cycle Detection in Parametric Timed Automata*'. *TACAS*. Volume 12651. LNCS. Springer, 2021, pages 311–329. DOI: 10.1007/978-3-030-72016-2_17 (cited on page 160).

[Abe03] D. A. Aberdeen. '*Policy-Gradient Algorithms for Partially Observable Markov Decision Processes*'. PhD thesis. The Australian National University, 2003 (cited on pages 126, 141, 142).

[ABEM+96] B. Abramson, J. Brown, W. Edwards, A. Murphy and R. L. Winkler. '*Hailfinder: A Bayesian system for forecasting severe weather*'. *Int. J. Forecast.* 12.1 (1996), pages 57–71. DOI: 10.1016/0169-2070(95)00664-8 (cited on page 198).

[ABZ10] C. Amato, D. S. Bernstein and S. Zilberstein. '*Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs*'. *Auton. Agents Multi Agent Syst.* 21.3 (2010), pages 293–320. DOI: 10.1007/s10458-009-9103-z (cited on page 127).

[ACFE09] É. André, T. Chatain, L. Fribourg and E. Encrenaz. '*An Inverse Method for Parametric Timed Automata*'. *Int. J. Found. Comput. Sci.* 20.5 (2009), pages 819–836. DOI: 10.1016/j.entcs.2008.12.029 (cited on page 160).

[ACJK21] R. Andriushchenko, M. Ceska, S. Junges and J.-P. Katoen. '*Inductive Synthesis for Probabilistic Programs Reaches New Horizons*'. *TACAS*. Volume 12651. LNCS. Springer, 2021, pages 191–209. DOI: 10.1007/978-3-030-72016-2_11 (cited on page 127).

[AD00] R. Ash and C. Doleans-Dade. '*Probability and Measure Theory*'. Elsevier, 2000 (cited on page 19).

[AD94] R. Alur and D. L. Dill. '*A Theory of Timed Automata*'. *Theor. Comput. Sci.* 126.2 (1994), pages 183–235. DOI: 10.1016/0304-3975(94)90010-8 (cited on pages 3, 159, 161).

[ADF20] É. André, B. Delahaye and P. Fournier. '*Consistency in Parametric Interval Probabilistic Timed Automata*'. *J. Log. Algebraic Methods Program.* 110 (2020). DOI: 10.1016/j.jlamp.2019.04.007 (cited on page 160).

[AFS13] É. André, L. Fribourg and J. Sproston. '*An extension of the inverse method to probabilistic timed automata*'. *Form. Methods Syst. Des.* 42.2 (2013), pages 119–145. DOI: 10.1007/s10703-012-0169-x (cited on page 160).

[AH90] J. Aspnes and M. Herlihy. '*Fast Randomized Consensus Using Shared Memory*'. *J. Algorithms* 11.3 (1990), pages 441–461. DOI: 10.1016/0196-6774(90)90021-6 (cited on page 194).

[AHV93] R. Alur, T. A. Henzinger and M. Y. Vardi. '*Parametric real-time reasoning*'. *STOC*. ACM Press, 1993, pages 592–601. DOI: 10.1145/167088.167242 (cited on page 160).

[AMP98] E. Asarin, O. Maler and A. Pnueli. '*On Discretization of Delays in Timed Automata and Digital Circuits*'. *CONCUR*. Volume 1466. LNCS. Springer, 1998, pages 470–484. DOI: 10.1007/BFb0055642 (cited on pages 159, 166).

[And19] É. André. '*What's decidable about parametric timed automata?*' *Int. J. Softw. Tools Technol. Transf.* 21.2 (2019), pages 203–219. DOI: 10.1007/s10009-017-0467-0 (cited on page 160).

[AP18] G. Agha and K. Palmskog. '*A Survey of Statistical Model Checking*'. *ACM Trans. Model. Comput. Simul.* 28.1 (2018), 6:1–6:39. DOI: 10.1145/3158668 (cited on page 160).

[Åst65] K. Åström. '*Optimal control of Markov processes with incomplete state information*'. *J. Math. Ana. Appl.* 10.1 (1965), pages 174–205. DOI: 10.1016/0022-247X(65)90154-X (cited on page 199).

[AVBK+17] S. Aflaki, M. Volk, B. Bonakdarpour, J. Katoen and A. Storjohann. '*Automated Fine Tuning of Probabilistic Self-Stabilizing Algorithms*'. *SRDS*. IEEE Computer Society, 2017, pages 94–103. DOI: 10.1109/SRDS.2017.22 (cited on page 2).

[BAFK18] C. Baier, L. de Alfaro, V. Forejt and M. Kwiatkowska. '*Model Checking Probabilistic Systems*'. *Handbook of Model Checking*. Springer, 2018, pages 963–999. DOI: 10.1007/978-3-319-10575-8_28 (cited on pages 1, 4, 160).

[BB04] D. Braziunas and C. Boutilier. '*Stochastic Local Search for POMDP Controllers*'. *AAAI*. AAAI Press / The MIT Press, 2004, pages 690–696 (cited on page 127).

[BCDS13] L. Brim, M. Ceska, S. Drazan and D. Safránek. '*Exploring Parameter Space of Stochastic Biochemical Systems Using Quantitative Model Checking*'. *CAV*. Volume 8044. LNCS. Springer, 2013. DOI: 10.1007/978-3-642-39799-8_7 (cited on page 30).

[BDHH+17] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges and A. Turrini. '*JANI: Quantitative Model and Tool Interaction*'. *TACAS*. Volume 10206. LNCS. 2017, pages 151–168. DOI: 10.1007/978-3-662-54580-5_9 (cited on page 178).

[BDHS20] C. E. Budde, P. R. D'Argenio, A. Hartmanns and S. Sedwards. '*An efficient statistical model checker for nondeterminism and rare events*'. *Int. J. Softw. Tools Technol. Transf.* 22.6 (2020), pages 759–780. DOI: 10.1007/s10009-020-00563-2 (cited on page 160).

[Bel57] R. Bellman. '*A Markovian decision process*'. *J. math. mech.* 6.4 (1957), pages 679–684. DOI: https://doi.org/10.1512/iumj.1957.6.56038 (cited on page 11).

[BG09] B. Bonet and H. Geffner. '*Solving POMDPs: RTDP-Bel vs. Point-based Algorithms*'. *IJCAI*. DOI: https://dl.acm.org/doi/10.5555/1661445.1661709. 2009, pages 1641–1646 (cited on page 192).

[BGKR+11] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan and S. A. Smolka. '*Model Repair for Probabilistic Systems*'. *J. math. mech.* Volume 6605. LNCS. Springer, 2011. DOI: 10.1007/978-3-642-19835-9_30 (cited on pages 2, 58, 100).

[BHHJ+20] C. Baier, C. Hensel, L. Hutschenreiter, S. Junges, J. Katoen and J. Klein. '*Parametric Markov chains: PCTL complexity and fraction-free Gaussian elimination*'. *Inf. Comput.* 272 (2020), page 104504. DOI: 10.1016/j.ic.2019.104504 (cited on pages 40, 44, 54, 100).

[BHK19] C. Baier, H. Hermanns and J. Katoen. '*The 10, 000 Facets of MDP Model Checking*'. *Computing and Software Science*. Volume 10000. LNCS. Springer, 2019, pages 420–451. DOI: 10.1007/978-3-319-91908-9_21 (cited on pages 4, 160).

[BJKQ20] A. Bork, S. Junges, J.-P. Katoen and T. Quatmann. '*Verification of Indefinite-Horizon POMDPs*'. *ATVA*. Volume 12302. LNCS. Springer, 2020, pages 288–304. DOI: 10.1007/978-3-030-59152-6_16 (cited on pages 127, 192, 193, 197, 198).

[BK08] C. Baier and J.-P. Katoen. '*Principles of model checking*'. MIT Press, 2008 (cited on pages 9, 19, 20, 58, 141, 143, 144, 159, 161, 189).

[BKQ22] A. Bork, J. Katoen and T. Quatmann. '*Under-Approximating Expected Total Rewards in POMDPs*'. *TACAS*. Volume 13244. LNCS. Springer, 2022, pages 22–40. DOI: 10.1007/978-3-030-99527-0_2 (cited on page 127).

[BMM12] M. Bani Yassein, S. Manaseer and A. Momani. '*Adaptive Backoff Algorithm for Wireless Internet*'. *J. Emerg. Technol. Web Intell.* 4 (May 2012). DOI: 10.4304/jetwi.4.2.155-163 (cited on page 1).

[BN08] L. Barrett and S. Narayanan. '*Learning all optimal policies with multiple criteria*'. *ICML*. Volume 307. ACM International Conference Proceeding Series. ACM Press, 2008, pages 41–47. DOI: 10.1145/1390156.1390162 (cited on page 194).

[Bon98] B. Bonet. '*Solving large POMDPs using real time dynamic programming*'. *Proc. AAAI Fall Symp. on POMDPs*. 1998 (cited on page 127).

[BRPT17] S. Bharadwaj, S. L. Roux, G. A. Pérez and U. Topcu. '*Reduction Techniques for Model Checking and Learning in MDPs*'. *IJCAI*. ijcai.org, 2017. DOI: 10.24963/ijcai.2017/597 (cited on page 30).

[BSHV03] H. C. Bohnenkamp, P. van der Stok, H. Hermanns and F. W. Vaandrager. '*Cost-Optimization of the IPv4 Zeroconf Protocol*'. *DSN*. IEEE CS, 2003. DOI: 10.1109/DSN.2003.1209963 (cited on pages 191, 196).

[CAG05] S. Cheshire, B. Aboba and E. Guttman. '*Dynamic Configuration of IPv4 Link-Local Addresses*'. *RFC* 3927 (2005), pages 1–33 (cited on page 197).

[CCD16] K. Chatterjee, M. Chmelik and J. Davies. '*A Symbolic SAT-Based Algorithm for Almost-Sure Reachability with Small Strategies in POMDPs*'. *AAAI*. DOI: https://dl.acm.org/doi/10.5555/3016100.3016355. AAAI Press, 2016, pages 3225–3232 (cited on page 127).

[CCGK+18] R. Calinescu, M. Ceska, S. Gerasimou, M. Kwiatkowska and N. Paoletti. '*Efficient synthesis of robust models for stochastic systems*'. *J. Syst. Softw.* 143 (2018), pages 140–158. DOI: 10.1016/j.jss.2018.05.013 (cited on page 59).

[CCW92] K. C. C. Chan, J. Y. Ching and A. K. C. Wong. '*A probabilistic inductive learning approach to the acquisition of knowledge in medical expert systems*'. *CBMS*. IEEE CS, 1992, pages 572–581. DOI: 10.1109/CBMS.1992.245017 (cited on page 198).

[CDPK+17] M. Ceska, F. Dannenberg, N. Paoletti, M. Kwiatkowska and L. Brim. '*Precise parameter synthesis for stochastic biochemical systems*'. *Acta Inform.* 54.6 (2017), pages 589–623. DOI: 10.1007/s00236-016-0265-2 (cited on page 3).

[CFRS14] T. Chen, Y. Feng, D. S. Rosenblum and G. Su. '*Perturbation Analysis in Verification of Discrete-Time Markov Chains*'. *CONCUR*. Volume 8704. LNCS. Springer, 2014, pages 218–233. DOI: 10.1007/978-3-662-44584-6_16 (cited on page 100).

[CHHK+13] T. Chen, E. M. Hahn, T. Han, M. Z. Kwiatkowska, H. Qu and L. Zhang. '*Model Repair for Markov Decision Processes*'. *TASE*. IEEE, 2013. DOI: 10.1109/TASE.2013.20 (cited on pages 2, 3, 6, 59, 100, 125, 126, 148).

[CHK08] T. Chen, T. Han and J. Katoen. '*Time-Abstracting Bisimulation for Probabilistic Timed Automata*'. *TASE*. IEEE CS, 2008, pages 177–184. DOI: 10.1109/TASE.2008.29 (cited on page 175).

[Chu67] K. L. Chung. '*Markov chains*'. Springer, 1967 (cited on page 11).

[CJJJ+21] M. Cubuktepe, N. Jansen, S. Junges, Joost-Pieter Katoen and U. Topcu. '*Experiments for Convex Optimization for Parameter Synthesis in MDPs*'. 2021. DOI: 10.5281/ZENODO.5745851 (cited on page 148).

[CJJK+17] M. Cubuktepe, N. Jansen, S. Junges, J. Katoen, I. Papusha, H. A. Poonawala and U. Topcu. '*Sequential Convex Programming for the Efficient Verification of Parametric MDPs*'. *TACAS*. Volume 10206. LNCS. 2017, pages 133–150. DOI: 10.1007/978-3-662-54580-5_8 (cited on pages 3, 100).

[CJJK+18] M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen and U. Topcu. '*Synthesis in pMDPs: A Tale of 1001 Parameters*'. *ATVA*. Volume 11138. LNCS. Springer, 2018, pages 160–176. DOI: 10.1007/978-3-030-01090-4_10 (cited on pages 3, 6, 57, 58, 100, 125, 126, 148).

[CJJK+22] M. Cubuktepe, N. Jansen, S. Junges, J. Katoen and U. Topcu. '*Convex Optimization for Parameter Synthesis in MDPs*'. *IEEE Trans. Autom. Control.* 67.12 (2022), pages 6333–6348. DOI: 10.1109/TAC.2021.3133265 (cited on pages 3, 6, 100, 125, 126, 148).

[CJJK19] M. Ceska, N. Jansen, S. Junges and J. Katoen. '*Shepherding Hordes of Markov Chains*'. *TACAS*. Volume 11428. LNCS. Springer, 2019, pages 172–190. DOI: 10.1007/978-3-030-17465-1_10 (cited on page 3).

[CJT20] S. Carr, N. Jansen and U. Topcu. '*Verifiable RNN-Based Policies for POMDPs Under Temporal Logic Constraints*'. *IJCAI*. ijcai.org, 2020, pages 4121–4127. DOI: 10.24963/ijcai.2020/570 (cited on pages 126, 127).

[CJWS+19] S. Carr, N. Jansen, R. Wimmer, A. C. Serban, B. Becker and U. Topcu. '*Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks*'. *IJCAI*. ijcai.org, 2019, pages 5532–5539. DOI: 10.24963/ijcai.2019/768 (cited on page 127).

[CK18] G. Chatzieleftheriou and P. Katsaros. '*Abstract model repair for probabilistic systems*'. *Inf. Comput.* 259.1 (2018), pages 142–160. DOI: 10.1016/j.ic.2018.02.019 (cited on pages 2, 189, 191).

[CKL94] A. R. Cassandra, L. P. Kaelbling and M. L. Littman. '*Acting Optimally in Partially Observable Stochastic Domains*'. *AAAI*. Volume 2. DOI: https://dl.acm.org/doi/10.5555/199480.199520. AAAI Press / The MIT Press, 1994, pages 1023–1028 (cited on pages 192, 198).

[CKWL19] J. Chen, A. Kyrillidis, C. Wolfe and Z. Li. '*Demon: Improved Neural Network Training with Momentum Decay*'. *CoRR* abs/1910.04952 (2019). DOI: 10.48550/arXiv.1910.04952 (cited on page 126).

[CRSL22] M. Cleaveland, I. Ruchkin, O. Sokolsky and I. Lee. '*Monotonic Safety for Scalable and Data-Efficient Probabilistic Safety Analysis*'. *ICCPS*. IEEE, 2022, pages 92–103. DOI: 10.1109/ICCPS54341.2022.00015 (cited on pages 70, 97).

[CSH08] K. Chatterjee, K. Sen and T. A. Henzinger. '*Model-Checking omega-Regular Properties of Interval Markov Chains*'. *FoSSaCS*. Volume 4962. LNCS. Springer, 2008. DOI: 10.1007/978-3-540-78499-9_22 (cited on pages 2, 30, 160).

[Dal68] D. J. Daley. '*Stochastically monotone Markov chains*'. *Probab. Theory Relat. Fields* 10 (1968), pages 305–317. DOI: 10.1007/BF00531852 (cited on page 29).

[Daw04] C. Daws. '*Symbolic and Parametric Model Checking of Discrete-Time Markov Chains*'. *ICTAC*. Volume 3407. LNCS. Springer, 2004, pages 280–294. DOI: 10.1007/978-3-540-31862-0_21 (cited on pages 2, 3, 11, 24, 100).

[DDB20] A. Dieuleveut, A. Durmus and F. Bach. '*Bridging the gap between constant step size stochastic gradient descent and Markov chains*'. *The Annals of Statistics* 48.3 (2020), pages 1348–1382. DOI: 10.1214/19-AOS1850 (cited on pages 3, 126).

[DHLS16] P. R. D'Argenio, A. Hartmanns, A. Legay and S. Sedwards. '*Statistical Approximation of Optimal Schedulers for Probabilistic Timed Automata*'. *IFM*. Volume 9681. LNCS. Springer, 2016, pages 99–114. DOI: 10.1007/978-3-319-33693-0_7 (cited on page 160).

[DHS11] J. C. Duchi, E. Hazan and Y. Singer. '*Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*'. *J. Mach. Learn. Res.* 12 (2011). DOI: https://dl.acm.org/doi/10.5555/1953048.2021068, pages 2121–2159 (cited on page 126).

[DJJC+15] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen and E. Ábrahám. '*PROPhESY: A PRObabilistic ParamEter SYnthesis Tool*'. *CAV*. Volume 9206. LNCS. Springer, 2015. DOI: 10.1007/978-3-319-21690-4_13 (cited on pages 58, 59, 64, 148).

[DJJL01] P. R. D'Argenio, B. Jeannet, H. E. Jensen and K. G. Larsen. '*Reachability Analysis of Probabilistic Systems by Successive Refinements*'. *PAPM-PROBMIV*. Volume 2165. LNCS. Springer, 2001. DOI: 10.1007/3-540-44804-7_3 (cited on page 190).

[DJKG16] C. Dombrowski, S. Junges, J. Katoen and J. Gross. '*Model-Checking Assisted Protocol Design for Ultra-reliable Low-Latency Wireless Networks*'. *SRDS*. IEEE CS, 2016, pages 307–316. DOI: 10.1109/SRDS.2016.048 (cited on page 3).

[DKN04] C. Daws, M. Z. Kwiatkowska and G. Norman. '*Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM*'. *Int. J. Softw. Tools Technol. Transf.* 5.2-3 (2004), pages 221–236. DOI: 10.1007/s10009-003-0118-5 (cited on pages 3, 175).

[DKV09] M. Droste, W. Kuich and H. Vogler. '*Handbook of weighted automata*'. Springer, 2009 (cited on page 145).

[DLLM+15] A. David, K. G. Larsen, A. Legay, M. Mikucionis and D. B. Poulsen. '*Uppaal SMC tutorial*'. *Int. J. Softw. Tools Technol. Transf.* 17.4 (2015), pages 397–415. DOI: 10.1007/s10009-014-0361-y (cited on page 160).

[DLM10] P. Dai Pra, P.-Y. Louis and I. G. Minelli. '*Realizable monotonicity for continuous-time Markov processes*'. *Stoch. Process. Their Appl.* 120.6 (2010), pages 959–982. DOI: 10.1016/j.spa.2010.03.002 (cited on page 30).

[DPR08] F. Doshi, J. Pineau and N. Roy. '*Reinforcement learning with limited reinforcement: using Bayes risk for active learning in POMDPs*'. *ICML*. Volume 307. ACM International Conference Proceeding Series. ACM Press, 2008, pages 256–263. DOI: 10.1016/j.artint.2012.04.006 (cited on page 127).

[EGL82] S. Even, O. Goldreich and A. Lempel. '*A Randomized Protocol for Signing Contracts*'. *CRYPTO*. Plenum Press, 1982, pages 205–210. DOI: 10.1007/978-1-4757-0602-4_19 (cited on page 192).

[Eli03] C. E. P. and Elizabeth M. Belding-Royer and Samir R. Das. '*Ad hoc On-Demand Distance Vector (AODV) Routing*'. *RFC* 3561 (2003), pages 1–37 (cited on page 196).

[FCGA21] X. Fang, R. Calinescu, S. Gerasimou and F. Alhwikem. '*Fast Parametric Model Checking through Model Fragmentation*'. *ICSE*. IEEE, 2021, pages 835–846. DOI: 10.1109/ICSE43902.2021.00081 (cited on page 100).

[Fel01] W. Feller. '*An introduction to probability theory and its applications*'. John Wiley & Sons, 2001 (cited on pages 9, 19).

[FGT11] A. Filieri, C. Ghezzi and G. Tamburrelli. '*Run-time efficient probabilistic model checking*'. *ICSE*. ACM Press, 2011. DOI: 10.1145/1985793.1985840 (cited on pages 100, 191).

[FKNP11] V. Forejt, M. Z. Kwiatkowska, G. Norman and D. Parker. '*Automated Verification Techniques for Probabilistic Systems*'. *SFM*. Volume 6659. LNCS. Springer, 2011, pages 53–113. DOI: 10.1007/978-3-642-21455-4_3 (cited on page 178).

[Fru06] M. Fruth. '*Probabilistic Model Checking of Contention Resolution in the IEEE 802.15.4 Low-Rate Wireless Personal Area Network Protocol*'. *ISoLA*. IEEE CS, 2006, pages 290–297. DOI: 10.1109/ISoLA.2006.34 (cited on page 3).

[FS18] D. J. Fremont and S. A. Seshia. '*Reactive Control Improvisation*'. *CAV*. Volume 10981. LNCS. Springer, 2018, pages 307–326. DOI: 10.1007/978-3-319-96145-3_17 (cited on page 1).

[GHS18] P. Gainer, E. M. Hahn and S. Schewe. '*Accelerated Model Checking of Parametric Markov Chains*'. *ATVA*. Volume 11138. LNCS. Springer, 2018, pages 300–316. DOI: 10.1007/978-3-030-01090-4_18 (cited on pages 3, 100).

[GJ95] H. Gregersen and H. E. Jensen. '*Formal Design of Reliable Real Time Systems*'. Master's thesis. Department of Mathematics and Computer Science, Aalborg University, 1995 (cited on pages 3, 159, 161).

[GST19] A. Gouberman, M. Siegle and B. Tati. '*Markov chains with perturbed rates to absorption: Theory and application to model repair*'. *Perform. Evaluation* 130 (2019), pages 32–50. DOI: 10.1016/j.peva.2018.11.006 (cited on pages 2, 30).

[Hau00] M. Hauskrecht. '*Value-Function Approximations for Partially Observable Markov Decision Processes*'. *J. Artif. Intell. Res.* 13 (2000). DOI: https://dl.acm.org/doi/10.5555/1622262.1622264, pages 33–94 (cited on page 127).

[HBC18] K. Horák, B. Bosanský and K. Chatterjee. '*Goal-HSVI: Heuristic Search Value Iteration for Goal POMDPs*'. *IJCAI*. ijcai.org, 2018, pages 4764–4770. DOI: 10.24963/ijcai.2018/662 (cited on page 127).

[HBK17] L. Hutschenreiter, C. Baier and J. Klein. '*Parametric Markov Chains: PCTL Complexity and Fraction-free Gaussian Elimination*'. *GandALF*. Volume 256. EPTCS. 2017. DOI: 10.4204/EPTCS.256.2 (cited on pages 3, 29).

[Her90] T. Herman. '*Probabilistic Self-Stabilization*'. *Inf. Process. Lett.* 35.2 (1990). DOI: 10.1016/0020-0190(90)90107-9 (cited on page 189).

[HH09] A. Hartmanns and H. Hermanns. '*A Modest Approach to Checking Probabilistic Timed Automata*'. *QEST*. IEEE CS, 2009, pages 187–196. DOI: 10.1109/QEST.2009.41 (cited on page 197).

[HH14] A. Hartmanns and H. Hermanns. '*The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification*'. *TACAS*. Volume 8413. LNCS. Springer, 2014, pages 593–598. DOI: 10.1007/978-3-642-54862-8_51 (cited on pages 160, 189, 196).

[HHZ09] E. M. Hahn, H. Hermanns and L. Zhang. '*Probabilistic Reachability for Parametric Markov Models*'. *SPIN*. Volume 5578. LNCS. Springer, 2009, pages 88–106. DOI: 10.1007/s10009-010-0146-x (cited on pages 24, 100).

[HHZ11a] E. M. Hahn, T. Han and L. Zhang. '*Synthesis for PCTL in Parametric Markov Decision Processes*'. *NASA Formal Methods*. Volume 6617. LNCS. Springer, 2011, pages 146–161. DOI: 10.1007/978-3-642-20398-5_12 (cited on pages 3, 100).

[HHZ11b] E. M. Hahn, H. Hermanns and L. Zhang. '*Probabilistic reachability for parametric Markov models*'. *Int. J. Softw. Tools Technol. Transf.* 13.1 (2011), pages 3–19. DOI: 10.1007/s10009-010-0146-x (cited on page 3).

[HHZW19] E. M. Hahn, H. Hermanns, L. Zhang and B. Wachter. '*PARAM website*'. https://depend.cs.uni-saarland.de/tools/param/. 2019 (cited on page 189).

[HJKQ+22] C. Hensel, S. Junges, J. Katoen, T. Quatmann and M. Volk. '*The probabilistic model checker Storm*'. *Int. J. Softw. Tools Technol. Transf.* 24.4 (2022), pages 589–610. DOI: 10.1007/s10009-021-00633-z (cited on pages 1, 62, 93, 116, 148).

[HK22] A. Hartmanns and B. Kohlen. '*Backwards Reachability for Probabilistic Timed Automata: A Replication Report*'. *CoRR* abs/2208.11928 (2022). DOI: 10.48550/arXiv.2208.11928 (cited on pages 159, 160, 171, 174, 176, 189, 196).

[HKPQ+19] A. Hartmanns, M. Klauck, D. Parker, T. Quatmann and E. Ruijters. '*The Quantitative Verification Benchmark Set*'. *TACAS*. Volume 11427. LNCS. Springer, 2019. DOI: 10.1007/978-3-030-17462-0_20 (cited on pages 149, 189, 193).

[HM18] S. Haddad and B. Monmege. '*Interval iteration algorithm for MDPs and IMDPs*'. *Theor. Comput. Sci.* 735 (2018), pages 111–131. DOI: 10.1016/j.tcs.2016.12.003 (cited on page 190).

[HM95] J. Han and C. Moraga. '*The Influence of the Sigmoid Function Parameters on the Speed of Backpropagation Learning*'. *IWANN*. Volume 930. LNCS. Springer, 1995, pages 195–201. DOI: 10.1007/3-540-59497-3_175 (cited on pages 128, 138, 140).

[HMP92] T. A. Henzinger, Z. Manna and A. Pnueli. '*What Good Are Digital Clocks?*' *ICALP*. Volume 623. LNCS. Springer, 1992, pages 545–558. DOI: 10.1007/3-540-55719-9_103 (cited on pages 159, 160, 166).

[How60] R. A. Howard. '*Dynamic programming and Markov processes.*' John Wiley, 1960 (cited on page 11).

[HP09] S. Haddad and N. Pekergin. '*Using Stochastic Comparison for Efficient Model Checking of Uncertain Markov Chains*'. *QEST*. IEEE CS, 2009, pages 177–186. DOI: https://doi.org/10.1109/QEST.2009.42 (cited on page 29).

[HRSV02] T. Hune, J. Romijn, M. Stoelinga and F. W. Vaandrager. '*Linear parametric model checking of timed automata*'. *J. Log. Algebraic Methods Program.* 52-53 (2002), pages 183–220. DOI: 10.1016/S1567-8326(02)00037-1 (cited on pages 160, 179).

[HSD17] A. Hartmanns, S. Sedwards and P. R. D'Argenio. '*Efficient simulation-based verification of probabilistic timed automata*'. *WSC*. IEEE, 2017, pages 1419–1430. DOI: 10.1109/WSC.2017.8247885 (cited on page 160).

[IJ90] A. Israeli and M. Jalfon. '*Token Management Schemes and Random Walks Yield Self-Stabilizing Mutual Exclusion*'. *PODC*. ACM Press, 1990, pages 119–131. DOI: 10.1145/93385.93409 (cited on pages 1, 194).

[JÁHJ+19] S. Junges, E. Ábrahám, C. Hensel, N. Jansen, J.-P. Katoen, T. Quatmann and M. Volk. '*Parameter Synthesis for Markov Models*'. *CoRR* abs/1903.07993 (2019). DOI: 10.48550/arXiv.1903.07993 (cited on page 100).

[JCVW+14] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J. Katoen and B. Becker. '*Accelerating Parametric Probabilistic Verification*'. *QEST*. Volume 8657. LNCS. Springer, 2014, pages 404–420. DOI: 10.1007/978-3-319-10696-0_31 (cited on pages 53, 100).

[JJWQ+18] S. Junges, N. Jansen, R. Wimmer, T. Quatmann, L. Winterer, J.-P. Katoen and B. Becker. '*Finite-State Controllers of POMDPs using Parameter Synthesis*'. *UAI*. AUAI Press, 2018, pages 519–529 (cited on pages 2, 57, 99, 115, 127, 189, 191, 199).

[JKJB18] N. Jansen, B. Könighofer, S. Junges and R. Bloem. '*Safe Reinforcement Learning via Probabilistic Shields*'. *CoRR* abs/1807.06096 (2018). DOI: 10.48550/arXiv.1807.06096 (cited on page 194).

[JKPW21] S. Junges, J. Katoen, G. A. Pérez and T. Winkler. '*The complexity of reachability in parametric Markov decision processes*'. *J. Comput. Syst. Sci.* 119 (2021), pages 183–210. DOI: 10.1016/j.jcss.2021.02.006 (cited on pages 2, 26–28, 44, 45, 80, 105).

[JL91] B. Jonsson and K. G. Larsen. '*Specification and Refinement of Probabilistic Processes*'. *LICS*. IEEE CS, 1991, pages 266–277. DOI: 10.1109/LICS.1991.151651 (cited on pages 2, 30, 100, 101, 160).

[JLR15] A. Jovanovic, D. Lime and O. H. Roux. '*Integer Parameter Synthesis for Real-Time Systems*'. *IEEE Trans. Software Eng.* 41.5 (2015), pages 445–461. DOI: 10.1109/TSE.2014.2357445 (cited on page 160).

[Jun20] S. Junges. '*Parameter synthesis in Markov models*'. PhD thesis. RWTH Aachen University, Germany, 2020 (cited on pages 7, 127, 143, 189, 191, 199, 200, 202).

[Kat16] J. Katoen. '*The Probabilistic Model Checking Landscape*'. *LICS*. ACM Press, 2016, pages 31–45. DOI: 10.1145/2933575.2934574 (cited on page 1).

[KB15] D. P. Kingma and J. Ba. '*Adam: A Method for Stochastic Optimization*'. *ICLR (Poster)*. 2015 (cited on pages 125, 126, 128, 135, 150).

[KBCD+18] J. Klein, C. Baier, P. Chrszon, M. Daum, C. Dubslaff, S. Klüppelholz, S. Märcker and D. Müller. '*Advances in probabilistic model checking with PRISM: variable reordering, quantiles and weak deterministic Büchi automata*'. *Int. J. Softw. Tools Technol. Transf.* 20.2 (2018), pages 179–194. DOI: 10.1007/s10009-017-0456-3 (cited on page 196).

[KG11] C. Krause and H. Giese. '*Model Checking Probabilistic Real-Time Properties for Service-Oriented Systems with Service Level Agreements*'. *INFINITY*. Volume 73. EPTCS. 2011, pages 64–78. DOI: 10.4204/EPTCS.73.8 (cited on page 160).

[KK20] M. Kamali and J. Katoen. '*Probabilistic Model Checking of AODV*'. *QEST*. Volume 12289. LNCS. Springer, 2020, pages 54–73. DOI: 10.1007/978-3-030-59854-9_6 (cited on pages 3, 196).

[KLC98] L. P. Kaelbling, M. L. Littman and A. R. Cassandra. '*Planning and Acting in Partially Observable Stochastic Domains*'. *Artif. Intell.* 101.1-2 (1998), pages 99–134. DOI: 10.1016/S0004-3702(98)00023-X (cited on pages 1, 126, 199).

[KNP07] M. Z. Kwiatkowska, G. Norman and D. Parker. '*Stochastic Model Checking*'. *SFM*. Volume 4486. LNCS. Springer, 2007, pages 220–270. DOI: 10.1007/978-3-540-72522-0_6 (cited on page 178).

[KNP09] M. Z. Kwiatkowska, G. Norman and D. Parker. '*Stochastic Games for Verification of Probabilistic Timed Automata*'. *FORMATS*. Volume 5813. LNCS. Springer, 2009, pages 212–227. DOI: 10.1007/978-3-642-04368-0_17 (cited on pages 159, 160, 167, 176).

[KNP11] M. Z. Kwiatkowska, G. Norman and D. Parker. '*PRISM 4.0: Verification of Probabilistic Real-Time Systems*'. *CAV*. Volume 6806. LNCS. Springer, 2011, pages 585–591. DOI: https://doi.org/10.1007/978-3-642-22110-1_47 (cited on pages 1, 160, 192).

[KNP12a] M. Z. Kwiatkowska, G. Norman and D. Parker. '*Probabilistic verification of Herman's self-stabilisation algorithm*'. *Formal Asp. Comput.* 24.4-6 (2012) (cited on page 189).

[KNP12b] M. Z. Kwiatkowska, G. Norman and D. Parker. '*The PRISM Benchmark Suite*'. *QEST*. IEEE CS, 2012, pages 203–204. DOI: 10.1109/QEST.2012.14 (cited on page 190).

[KNP19] M. Kwiatkowska, G. Norman and D. Parker. '*Verification and Control of Turn-Based Probabilistic Real-Time Games*'. *The Art of Modelling Computational Systems*. Volume 11760. LNCS. Springer, 2019, pages 379–396. DOI: 10.1007/978-3-030-31175-9_22 (cited on page 197).

[KNPS06] M. Z. Kwiatkowska, G. Norman, D. Parker and J. Sproston. '*Performance analysis of probabilistic timed automata using digital clocks*'. *Formal Methods Syst. Des.* 29.1 (2006), pages 33–78. DOI: https://doi.org/10.1007/978-3-540-40903-8_9 (cited on pages 159, 160, 166, 167, 170, 197).

[KNS01] M. Z. Kwiatkowska, G. Norman and R. Segala. '*Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM*'. *CAV*. Volume 2102. LNCS. Springer, 2001, pages 194–206. DOI: 10.1007/3-540-44585-4_17 (cited on page 194).

[KNS02] M. Z. Kwiatkowska, G. Norman and J. Sproston. '*Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol*'. *PAPM-PROBMIV*. Volume 2399. LNCS. Springer, 2002, pages 169–187. DOI: 10.1007/3-540-45605-8_11 (cited on page 194).

[KNS03] M. Z. Kwiatkowska, G. Norman and J. Sproston. '*Probabilistic Model Checking of Deadline Properties in the IEEE 1394 FireWire Root Contention Protocol*'. *Formal Aspects Comput.* 14.3 (2003), pages 295–318. DOI: 10.1007/s001650300007 (cited on page 196).

[KNSS02] M. Z. Kwiatkowska, G. Norman, R. Segala and J. Sproston. '*Automatic verification of real-time systems with discrete probability distributions*'. *Theor. Comput. Sci.* 282.1 (2002), pages 101–150. DOI: 10.1016/S0304-3975(01)00046-9 (cited on pages 3, 159–161, 164, 167, 175).

[KNSW07] M. Z. Kwiatkowska, G. Norman, J. Sproston and F. Wang. '*Symbolic model checking for probabilistic timed automata*'. *Inf. Comput.* 205.7 (2007), pages 1027–1077. DOI: 10.1016/j.ic.2007.01.004 (cited on pages 4, 159, 160, 166, 170–175, 195).

[Koh20] B. Kohlen. '*Parameter synthesis in probabilistic timed automata*'. Masterarbeit, RWTH Aachen University, 2020. Masterarbeit. RWTH Aachen University, 2020, 91 Seiten (cited on pages 5, 159, 174).

[KS76] J. G. Kemeny and J. L. Snell. '*Finite Markov Chains*'. Springer, 1976 (cited on page 11).

[KU02] I. Kozine and L. V. Utkin. '*Interval-Valued Finite Markov Chains*'. *Reliab. Comput.* 8.2 (2002), pages 97–113. DOI: 10.1023/A:1014745904458 (cited on pages 100, 101).

[LCK95] M. L. Littman, A. R. Cassandra and L. P. Kaelbling. '*Learning Policies for Partially Observable Environments: Scaling Up*'. *ICML*. Morgan Kaufmann, 1995, pages 362–370. DOI: 10.1016/B978-1-55860-377-6.50052-9 (cited on pages 192, 198).

[LJHC+20] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao and J. Han. '*On the Variance of the Adaptive Learning Rate and Beyond*'. *ICLR*. OpenReview.net, 2020 (cited on pages 125, 126, 128, 133, 136, 150).

[LMT05] R. Lanotte, A. Maggiolo-Schettini and A. Troina. '*Automatic Analysis of a Non-Repudiation Protocol*'. *Electron. Notes Theor. Comput. Sci.* 112 (2005), pages 113–129. DOI: 10.1016/j.entcs.2004.01.020 (cited on page 192).

[LMT07] R. Lanotte, A. Maggiolo-Schettini and A. Troina. '*Parametric probabilistic transition systems for system design and analysis*'. *Formal Aspects Comput.* 19.1 (2007), pages 93–109. DOI: 10.1007/s00165-006-0015-2 (cited on page 100).

[Lov91] W. S. Lovejoy. '*Computationally Feasible Bounds for Partially Observed Markov Decision Processes*'. *Oper. Res.* 39.1 (1991), pages 162–175. DOI: 10.1287/opre.39.1.162 (cited on page 127).

[LR81] D. Lehmann and M. O. Rabin. '*On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem*'. *POPL*. ACM Press, 1981, pages 133–138. DOI: 10.1145/567532.567547 (cited on page 194).

[LST14] A. Legay, S. Sedwards and L.-M. Traonouez. '*Scalable Verification of Markov Decision Processes*'. *WS-FMDS*. Volume 8938. LNCS. Springer, 2014, pages 350–362. DOI: 10.1007/978-3-319-15201-1_23 (cited on page 160).

[McC92] R. A. McCallum. '*First Results with Utile Distinction Memory for Reinforcement Learning*'. Technical report. DOI: https://dl.acm.org/doi/10.5555/898336. 1992 (cited on page 192).

[MHC03] O. Madani, S. Hanks and A. Condon. '*On the undecidability of probabilistic planning and related stochastic optimization problems*'. *Artif. Intell.* 147.1-2 (2003), pages 5–34. DOI: 10.1016/S0004-3702(02)00378-8 (cited on pages 2, 127).

[MKKC99] N. Meuleau, K. Kim, L. P. Kaelbling and A. R. Cassandra. '*Solving POMDPs by Searching the Space of Finite Policies*'. *UAI*. DOI: https://dl.acm.org/doi/10.5555/2073796.2073844. Morgan Kaufmann, 1999, pages 417–426 (cited on pages 126, 127).

[MKSG+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. A. Riedmiller. '*Playing Atari with Deep Reinforcement Learning*'. *CoRR* abs/1312.5602 (2013). DOI: 10.48550/arXiv.1312.5602 (cited on page 126).

[MPKK99] N. Meuleau, L. Peshkin, K. Kim and L. P. Kaelbling. '*Learning Finite-State Controllers for Partially Observable Environments*'. *UAI*. DOI: https://dl.acm.org/doi/10.5555/2073796.2073845. Morgan Kaufmann, 1999, pages 427–436 (cited on page 126).

[MR99] O. Markowitch and Y. Roggeman. '*Probabilistic non-repudiation without trusted third party*'. *Proc. 2nd Workshop on Security in Communication Networks*. 1999 (cited on page 197).

[MT20] E. Miehling and D. Teneketzis. '*Monotonicity properties for two-action partially observable Markov decision processes on partially ordered spaces*'. *Eur. J. Oper. Res.* 282.3 (2020), pages 936–944. DOI: 10.1016/j.ejor.2019.10.003 (cited on page 29).

[MY18] J. Ma and D. Yarats. '*Quasi-hyperbolic momentum and Adam for deep learning*'. *CoRR* abs/1810.06801 (2018). DOI: 10.48550/arXiv.1810.06801 (cited on page 126).

[Nes83] Y. E. Nesterov. '*A method for solving the convex programming problem with convergence rate $O(1/k\hat{}2)$*'. *Dokl.* Volume 269. 1983, pages 543–547 (cited on pages 125, 126, 128, 131).

[NPKS05] G. Norman, D. Parker, M. Z. Kwiatkowska and S. K. Shukla. '*Evaluating the reliability of NAND multiplexing with PRISM*'. *IEEE TCAD* 24.10 (2005). DOI: 10.1109/TCAD.2005.852033 (cited on page 189).

[NPS13] G. Norman, D. Parker and J. Sproston. '*Model checking for probabilistic timed automata*'. *Formal Methods Syst. Des.* 43.2 (2013), pages 164–190. DOI: 10.1007/s10703-012-0177-x (cited on pages 159, 167, 197).

[NPZ17] G. Norman, D. Parker and X. Zou. '*Verification and control of partially observable probabilistic systems*'. *Real Time Syst.* 53.3 (2017), pages 354–402. DOI: 10.1007/s11241-017-9269-4 (cited on pages 127, 192).

[PÁJT+15] S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella and J.-P. Katoen. '*A Greedy Approach for the Efficient Repair of Stochastic Models*'. *NFM*. Volume 9058. LNCS. 2015. DOI: 10.1007/978-3-319-17524-9_21 (cited on pages 2, 3, 29).

[PGT03] J. Pineau, G. J. Gordon and S. Thrun. '*Point-based value iteration: An anytime algorithm for POMDPs*'. *IJCAI*. DOI: https://dl.acm.org/doi/10.5555/1630659.1630806. Morgan Kaufmann, 2003, pages 1025–1032 (cited on page 127).

[Pit88] A. O. Pittenger. '*Monotonicity in a Markov Decision Process*'. *Math. Oper. Res.* 13.1 (1988), pages 65–73. DOI: 10.1287/moor.13.1.65 (cited on page 70).

[Put14] M. L. Puterman. '*Markov decision processes: discrete stochastic dynamic programming*'. John Wiley & Sons, 2014 (cited on page 11).

[PZ86] A. Pnueli and L. D. Zuck. '*Verification of Multiprocess Probabilistic Protocols*'. *Distrib. Comput.* 1.1 (1986), pages 53–72. DOI: 10.1007/BF01843570 (cited on page 194).

[QDJJ+16] T. Quatmann, C. Dehnert, N. Jansen, S. Junges and J. Katoen. '*Parameter Synthesis for Markov Models: Faster Than Ever*'. *ATVA*. Volume 9938. LNCS. 2016, pages 50–67. DOI: 10.1007/978-3-319-46520-3_4 (cited on pages 3, 5, 6, 24, 30, 99–101, 104, 106, 115, 129).

[Rab82] M. O. Rabin. '*N-Process Mutual Exclusion with Bounded Waiting by 4 Log_2 N-Valued Shared Variable*'. *J. Comput. Syst. Sci.* 25.1 (1982), pages 66–75. DOI: 10.1016/0022-0000(82)90010-1 (cited on page 194).

[RN20] S. Russell and P. Norvig. '*Artificial Intelligence: A Modern Approach (4th Edition)*'. Pearson, 2020 (cited on pages 1, 199).

[RP18] S. L. Roux and G. A. Pérez. '*The Complexity of Graph-Based Reductions for Reachability in Markov Decision Processes*'. *FoSSaCS*. Volume 10803. LNCS. Springer, 2018, pages 367–383. DOI: 10.1007/978-3-319-89366-2_20 (cited on page 30).

[RR98] M. K. Reiter and A. D. Rubin. '*Crowds: Anonymity for Web Transactions*'. *ACM Trans. Inf. Syst. Secur.* 1.1 (1998), pages 66–92. DOI: 10.1145/290163.290168 (cited on page 190).

[Rud16] S. Ruder. '*An overview of gradient descent optimization algorithms*'. *CoRR* abs/1609.04747 (2016). DOI: 10.48550/arXiv.1609.04747 (cited on pages 125, 150).

[Rum89] D. E. Rumelhart. '*Parallel Distributed Processing*'. MIT Press, 1989 (cited on pages 125, 126, 128, 130).

[SCBF+97] A. E. Smith, D. W. Coit, T. Baeck, D. Fogel and Z. Michalewicz. '*Penalty functions*'. *Handbook of evolutionary computation* 97.1 (1997), page C5 (cited on pages 128, 138).

[Scu21] M. Scutari. '*Bayesian Network Repository*'. 2021. URL: https://www.bnlearn.com/bnrepository/ (visited on 06/05/2021) (cited on pages 149, 198).

[SK20] B. Salmani and J.-P. Katoen. '*Bayesian Inference by Symbolic Model Checking*'. *QEST*. Volume 12289. LNCS. Springer, 2020, pages 115–133. DOI: 10.1007/978-3-030-59854-9_9 (cited on pages 149, 198).

[SK21] B. Salmani and J. Katoen. '*Fine-Tuning the Odds in Bayesian Networks*'. *ECSQARU*. Volume 12897. LNCS. Springer, 2021, pages 268–283. DOI: 10.1007/978-3-030-86772-0_20 (cited on pages 149, 198).

[SMDH13] I. Sutskever, J. Martens, G. E. Dahl and G. E. Hinton. '*On the importance of initialization and momentum in deep learning*'. *ICML*. Volume 28. JMLR. https:ttps://dl.acm.org/doi/10.5555/3042817.3043064. JMLR.org, 2013, pages 1139–1147 (cited on pages 125, 126, 128, 131).

[Spa12] M. T. J. Spaan. '*Partially Observable Markov Decision Processes*'. *Reinforcement Learning*. Volume 12. Adaptation, Learning, and Optimization. Springer, 2012, pages 387–414. DOI: 10.1007/978-3-642-27645-3_12 (cited on page 199).

[Spe18] J. Spel. '*Monotonicity in Markov Chains*'. Master thesis. University of Twente, 2018, 115 Seiten (cited on pages 5, 178).

[Spr21] J. Sproston. '*Probabilistic Timed Automata with Clock-Dependent Probabilities*'. *Fundam. Informaticae* 178.1-2 (2021), pages 101–138. DOI: 10.3233/FI-2021-2000 (cited on pages 176, 179).

[SS04] T. Smith and R. G. Simmons. '*Heuristic Search Value Iteration for POMDPs*'. *UAI*. DOI: https://dl.acm.org/doi/10.5555/1036843.1036906. AUAI Press, 2004, pages 520–527 (cited on page 192).

[SSY18] T. Sun, Y. Sun and W. Yin. '*On Markov Chain Gradient Descent*'. *NIPS*. DOI: https://dl.acm.org/doi/10.5555/3327546.3327656. 2018, pages 9918–9927 (cited on pages 3, 126).

[SV05] M. T. J. Spaan and N. A. Vlassis. '*Perseus: Randomized Point-based Value Iteration for POMDPs*'. *J. Artif. Intell. Res.* 24.1 (2005). DOI: https://dl.acm.org/doi/10.5555/1622519.1622525, pages 195–220 (cited on page 127).

[SV10] D. Silver and J. Veness. '*Monte-Carlo Planning in Large POMDPs*'. *NIPS*. DOI: https://dl.acm.org/doi/10.5555/2997046.2997137. Curran Associates, Inc., 2010, pages 2164–2172 (cited on page 127).

[SV99] M. Stoelinga and F. W. Vaandrager. '*Root Contention in IEEE 1394*'. *ARTS*. Volume 1601. LNCS. Springer, 1999, pages 53–74. DOI: 10.1007/3-540-48778-6_4 (cited on pages 196, 197).

[TBF05] S. Thrun, W. Burgard and D. Fox. '*Probabilistic robotics*'. Intelligent robotics and autonomous agents. MIT Press, 2005 (cited on page 1).

[TH12] T. Tieleman and G. Hinton. '*Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude*'. COURSERA: Neural Networks for Machine Learning. 2012 (cited on pages 125, 126, 128, 134, 150).

[Van98] R. J. Vanderbei. '*Linear programming - foundations and extensions*'. Volume 14. ISOR. Kluwer, 1998, pages 277–280. DOI: 10.1007/978-1-4614-7630-6 (cited on pages 128, 138).

[WCK18] Y. Wang, S. Chaudhuri and L. E. Kavraki. '*Bounded Policy Synthesis for POMDPs with Safe-Reachability Objectives*'. *AAMAS*. DOI: https://dl.acm.org/doi/10.5555/3237383.3237424. ACM, 2018, pages 238–246 (cited on page 127).

[WJWJ+17] L. Winterer, S. Junges, R. Wimmer, N. Jansen, U. Topcu, J. Katoen and B. Becker. '*Motion planning under partial observability using game-based abstraction*'. *CDC*. IEEE, 2017, pages 2201–2208. DOI: 10.1109/CDC.2017.8263971 (cited on page 127).

[WJWJ+21] L. Winterer, S. Junges, R. Wimmer, N. Jansen, U. Topcu, J.-P. Katoen and B. Becker. '*Strategy Synthesis for POMDPs in Robot Planning via Game-Based Abstractions*'. *IEEE Trans. Autom. Control.* 66.3 (2021), pages 1040–1054. DOI: 10.1109/TAC.2020.2990140 (cited on page 127).

[WS17] E. Walraven and M. T. J. Spaan. '*Accelerated Vector Pruning for Optimal POMDP Solvers*'. *AAAI*. DOI: https://dl.acm.org/doi/10.5555/3298023.3298101. AAAI Press, 2017, pages 3672–3678 (cited on page 127).

[WWJB20] L. Winterer, R. Wimmer, N. Jansen and B. Becker. '*Strengthening Deterministic Policies for POMDPs*'. *NFM*. Volume 12229. LNCS. Springer, 2020, pages 115–132. DOI: 10.1007/978-3-030-55754-6_7 (cited on page 127).

[YMZ11] L. Yang, S. Murugesan and J. Zhang. '*Real-Time Scheduling over Markovian Channels: When Partial Observability Meets Hard Deadlines*'. *GLOBECOM*. IEEE, 2011, pages 1–5. DOI: 10.1109/GLOCOM.2011.6133855 (cited on page 192).

[Zei12] M. D. Zeiler. '*ADADELTA: An Adaptive Learning Rate Method*'. *CoRR* abs/1212.5701 (2012). DOI: 10.48550/arXiv.1212.5701 (cited on page 126).

Co-authored Publications

This section contains the papers published during my time as a PhD student.

[1] J. Spel, S. Junges and J. Katoen. '*Are Parametric Markov Chains Monotonic?*' *ATVA*. Volume 11781. LNCS. Springer, 2019, pages 479–496. DOI: 10.1007/978-3-030-31784-3_28 (cited on pages 5, 9, 29, 69, 100).

[2] A. Hartmanns, J. Katoen, B. Kohlen and J. Spel. '*Tweaking the Odds in Probabilistic Timed Automata*'. *QEST*. Volume 12846. LNCS. Springer, 2021, pages 39–58. DOI: 10.1007/978-3-030-85172-9_3 (cited on pages 5, 159, 163, 176).

[3] J. Spel, S. Junges and J. Katoen. '*Finding Provably Optimal Markov Chains*'. *TACAS*. Volume 12651. LNCS. Springer, 2021, pages 173–190. DOI: 10.1007/978-3-030-72016-2_10 (cited on pages 5, 9, 99, 105).

[4] L. Heck, J. Spel, S. Junges, J. Moerman and J. Katoen. '*Gradient-Descent for Randomized Controllers Under Partial Observability*'. *VMCAI*. Volume 13182. LNCS. Springer, 2022, pages 127–150. DOI: 10.1007/978-3-030-94583-1_7 (cited on pages 3, 5, 9, 125).

[5] J. Spel, S. Stein and J. Katoen. '*POMDP Controllers with Optimal Budget*'. *QEST*. Volume 13479. LNCS. Springer, 2022, pages 107–130. DOI: 10.1007/978-3-031-16336-4_6 (cited on pages 5, 9, 29, 69, 99, 100, 110).

List of Symbols

Basic Notation

$\mathbb{Q}[\vec{x}]$	multivariate polynomial over $\vec{x}$
V	set of real-valued variables
X	distribution domain
μ	distribution function
$\mathsf{Distr}(X)$	set of distributions over X
$\mathsf{Distr}_{\mathbb{Q}(V)}(X)$	set of parametric distributions over X
$\mathsf{QDistr}_{\mathbb{Q}(V)}(X)$	set of parametric quasi-distributions over X
$\vec{e}_i$	vector with $e_j = 1$ if $i = 0$ and $e_j = 0$ otherwise
$\vec{u}$	instantiation

Gradient Descent

η	learning rate
β_1	average decay factor
β_2	squared average decay factor

Markov Model

Act	set of actions
$Act(s)$	set of available actions at state s
Act_R^{min}	minimizing action set
V	set of parameters
$\mathbf{P}$	probabilistic transition function
$\mathbf{Q}$	transition function with a quasi-distribution
s_I	initial state
$\mathsf{occur}(s)$	set of variables occuring at state s
$\mathsf{occur}(s, \alpha)$	set of variables occuring at state s for action α
S	set of states
T	set of target states
U	set of safe states
$\mathsf{succ}(s)$	set of successor states of s
$\mathsf{succ}(s, \alpha)$	set of successor states of s w.r.t. action α
$\mathsf{succ}(s, \sigma)$	set of successor states of s w.r.t. independent scheduler σ
$\mathsf{succ}(s, \sigma(\vec{u}))$	set of successor states of s w.r.t. dependent scheduler $\sigma(\vec{u})$

σ	independent scheduler
$\sigma^{min,\mathsf{pr}}$	minimizing scheduler for reachability probability
$\sigma^{min,\mathsf{rew}}$	minimizing scheduler for expected total reward
$\sigma(\vec{u})$	dependent scheduler
$Dep(\sigma(\vec{u}))$	set of independent schedulers on which $\sigma(\vec{u})$ depends
$Sched(\mathcal{M})$	set of independent schedulers of $\mathcal{M}$
$Sched(\mathcal{M}, \vec{u})$	set of dependent schedulers of $\mathcal{M}$
$Sched(\mathcal{M})^{\mathcal{A}^{Act}}$	set of schedulers with action assumptions

Model Symbols

$\mathcal{A}$	PTA
$\mathcal{D}$	pMC
$\mathcal{M}$	pMDP
$\mathcal{P}$	pPTA
$\mathcal{W}$	WFA
$[\![\cdot]\!]_{\boldsymbol{br}}$	backward reachability semantics
$[\![\cdot]\!]_{\boldsymbol{dc}}$	digital clocks semantics
$[\![\cdot]\!]_{\boldsymbol{dense}}$	dense-time semantics
$[\![\cdot]\!]_{\boldsymbol{fr}}$	forward reachability semantics

Monotonicity

$\uparrow_p$	globally monotonic increasing in parameter p
$\uparrow_p^{\ell}$	locally monotonic increasing in parameter p
$\uparrow_p^{\ell,n}$	Locally monotonic increasing in parameter p for n steps
$\uparrow_p^{min}$	globally monotonic increasing in p under minimization
$\uparrow_{p,s}^{\ell,min}$	locally monotonic increasing in p at s under minimization
$\downarrow$	monotonic decreasing, superscripts and subscripts as for monotonic increasing
R	on region R, superscript, omitted if clear from context

Order

$\mathcal{A}^{Act}$	set of action assumptions
$\mathcal{A}_{\equiv}$	set of state assumptions w.r.t. $\equiv$
$\mathcal{A}_{\prec}$	set of state assumptions w.r.t. $\prec$
$\mathsf{lb}(X)$	downward closure of X w.r.t. the reward order
$\mathsf{lb}^{\mathbf{min}}(X)$	downward closure of X w.r.t. the reward order under minimization
$\mathsf{ub}(X)$	upward closure of X w.r.t. the reward order
$\mathsf{ub}^{\mathbf{min}}(X)$	upward closure of X w.r.t. the reward order under minimization
$\max(X)$	largest element of X w.r.t. the reward order
$\max^{\mathbf{min}}(X)$	largest element of X w.r.t. the reward order under minimization
$\min(X)$	smallest element of X w.r.t. the reward order

$\min^{\mathbf{min}}(X)$	smallest element of X w.r.t. the reward order under minimization
$\preceq^{\mathsf{rew}}$	non-strict reward order
$\preceq^{\mathsf{rew},\mathcal{A}}$	non-strict reward order with assumptions
$\preceq^{\mathsf{rew}}_{min}$	non-strict reward order under minimization
$\equiv^{\mathsf{rew}}$	equivalence reward order
$\equiv^{\mathsf{rew}}_{min}$	equivalence reward order under minimization
$\prec^{\mathsf{rew}}$	irreflexive variant of $\preceq^{\mathsf{rew}}$
$\prec^{\mathsf{rew}}_{min}$	irreflexive variant of $\preceq^{\mathsf{rew}}_{min}$
R, T	w.r.t. region R and target states T, subscript, omitted if clear from context

Parameter Lifting

$L_R(s)$	lower bound on the expected total reward of s on R
$U_R(s)$	upper bound on the expected total reward of s on R

Paths

$\hat{\pi}$	finite path
π_i	i^{th} state of π
π	infinite path
$Paths^{\mathcal{M}}_{inf}(s)$	set of infinite paths in $\mathcal{M}$ starting from state s
$Paths^{\mathcal{M}}_{fin}(s)$	set of finite paths in $\mathcal{M}$ starting from state s
$Paths^{\mathcal{M}}_{fin}$	set of all finite paths in $\mathcal{M}$
$Paths_n(s)$	set of all finite paths with length n starting from s
$pref(\pi)$	set of prefixes of path

Probabilities and Rewards

$Cyl(\hat{\pi})$	cylinder set of $\hat{\pi}$
$r(\hat{\pi})$	expected total reward for finite path $\hat{\pi}$
$r(\hat{\pi}, \Diamond T)$	expected total reward for $\hat{\pi}$ until reaching T
$r(\pi, \Diamond T)$	expected total reward for π until reaching T
r	state reward function
$\mathrm{ER}^s(\Diamond T)$	expected total reward of reaching T from s
$\mathrm{ER}^s_{min}(\Diamond T)$	minimal expected total reward of reaching T from s
$\mathsf{ER}^{s\to T}$	solution function for the expected total reward of reaching T from s
$\mathsf{ER}^{s\to T}_{min}$	minimal solution function for the expected total reward of reaching T from s
$\Pr(\hat{\pi})$	probability of a finite path
$\Pr^s(\Diamond T)$	probability of reaching T from s
$\Pr^s_{min}(\Diamond T)$	minimal probability of reaching T from s
$\Pr^{s\to T}$	solution function for the probability of reaching T from s

$\Pr_{min}^{s \to T}$	minimal solution function for the probability of reaching T from s
$\Pr^{s}(U \cup T)$	probability of reaching T from s while staying in U before
$\Pr_{min}^{s}(U \cup T)$	minimal probability of reaching T from s while staying in U before
$\Pr^{s \overset{U}{\to} T}$	solution function for the probability of reaching T from s while staying in U before
$\Pr_{min}^{s \overset{U}{\to} T}$	minimal solution function for the probability of reaching T from s while staying in U before
$\mathcal{D}$	in $\mathcal{D}$, subscript, omitted if clear from the context
$\mathcal{M}$	in $\mathcal{M}$, subscript, omitted if clear from the context

Problems

pMC-F	feasibility problem for pMC
pMC-Mon	decision problem whether a pMC is monotonic
pMC-RO	decision problem whether two states are ordered by the expected total reward
pMC-V	verification problem for pMC
pMDP-F	feasibility problem for pMDP
pMDP-Mon-Min	decision problem whether a pMDP is monotonic under minimization
pMDP-RO-Min	decision problem whether two states are ordered under minimization by the expected total reward
pMDP-Mon	decision problem whether a pMDP is monotonic
pMDP-RO	decision problem whether two states are ordered by the expected total reward
pMDP-V	verification problem for pMDP

Timed Automaton

Act	set of actions
$\mathbf{P}$	probabilistic transition function
$CC(\mathcal{X})$	set of clock constraints
$\mathcal{X}$	set of clocks
$\mathbf{k}_x$	maximum constant to which x is compared
inv	location invariant function
l_I	initial location
Loc	set of locations
(l, ζ)	symbolic state
$[\![true]\!]$	set of all possible symbolic states
$dpre$	discrete predecessor operation
$tpre_U$	time predecessor operation
v	clock valuation

$v[X := 0]$	clock valuation in which all clocks in X are reset to 0
$\mathbf{0}$	zero valuation
$\mathbb{R}_{\geq 0}^{\mathcal{X}}$	set of clock valuations
$\mathbb{N}^{\mathcal{X}}$	set of digital clocks valuations
$\zeta_{\mathcal{X}}$	zone
ζ_l	zone satisfying location invariant of l
$Zones(\mathcal{X})$	set of zones over $\mathcal{X}$
$[X := \mathbf{0}]\zeta$	zone that represents ζ after resetting clocks in X

List of Acronyms

cpTA constrained parameteric TA

CTL computation tree logic

GD gradient descent

iMC interval MC

iPTA interval PTA

LTL linear temporal logic

MC Markov chain

MDP Markov decision process

NWR never-worse relation

piPTA parametric interval PTA

pMC parametric MC

pMDP parametric MDP

pMM parametric Markov model

POMDP partially observable MDP

pPTA parametric PTA

PSO particle swarm optimization

PTA probabilistic TA

QCQP quadratically-constrained quadratic programming

SCP sequential convex programming

SMA simple movement average

SMC stochastical model checking

TA timed automaton

WFA weighted finite automaton

Index