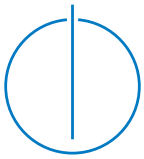
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Verified Solution Methods for Markov Decision Processes

Maximilian Schäffeler



DEPARTMENT OF INFORMATICS

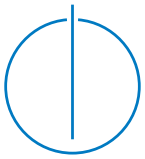
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Verified Solution Methods for Markov Decision Processes

Verifizierte Lösungsverfahren für Markov-Entscheidungsprozesse

Author:	Maximilian Schäffeler
Supervisor:	Prof. Tobias Nipkow
Advisor:	Mohammad Abdulaziz, PhD
Submission Date:	15.05.2021



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Garching, 13.05.2021 Maximilian Schäffeler

Acknowledgments

Tobias Nipkow was one of my first lecturers during the early semesters of my bachelor studies with his course "Perlen der Informatik". Later on, his lectures on logic, the semantics of programming languages, and functional data structures spurred my interest in formal verification, interactive theorem proving, and Isabelle/HOL. Thank you for allowing me to work on a thesis in your research group!

My advisor Mohammad Abdulaziz showed me how to use Isabelle/HOL beyond small homework assignments. He taught me how to do probability theory in Isabelle/HOL and showed me several tips and tricks which improve the everyday work with the proof assistant. He was also reassuring in moments when I was on bad terms with Isabelle and doubted my abilities to prove even simple theorems. Unfortunately, due to the ongoing pandemic, we could not yet meet in person. I am looking forward to working together with you in the future. Thank you, Mohammad!

I want to thank Benedikt Seidl, who provided valuable tips for code generation and other issues in life. I am grateful to Antonia Hartl and Jakob Schäffeler for their extensive feedback on draft versions of this thesis.

Finally, I would like to give special thanks to my parents Birgit and Robert Schäffeler, for their continued support over the last 24 years. Thank you for always being there for me!

Abstract

Markov decision processes (MDPs) allow modeling decision-making in systems that exhibit both random and deterministic behavior. In our work, we extend an existing formalization of MDPs in the interactive theorem prover Isabelle/HOL. First, we introduce policies that formalize the strategy of the decision-maker. To assign a value to policies, we add reward functions to the MDP formalization. The problem arises of how to find an optimal policy, one that maximizes the rewards accumulated over time. We solve the problem by formalizing its Bellman equation and give conditions under which an optimal policy exists.

Based on these developments, we verify the value iteration and policy iteration algorithms which compute (close to) optimal policies. This formalization may serve as a basis for future work on reinforcement learning algorithms, planning under uncertainty, and partially observable MDPs.

Markov-Entscheidungsprozesse (MDPs) erlauben es, Systeme zu modellieren, die sowohl stochastisches als auch deterministisches Verhalten aufweisen. Das Verhalten des Systems wird dabei durch von einem Agenten durchgeführte Aktionen beeinflusst. Diese Arbeit erweitert eine bereits existierende Formalisierung von MDPs für den interaktiven Theorembeweiser Isabelle/HOL. Zuerst werden Strategien, die das Verhalten des Agenten beschreiben, formal eingeführt. Um diese Strategien zu bewerten und zu vergleichen, wird die existierende Formalisierung um eine Belohnungsfunktion ergänzt. Es stellt sich nun das Problem eine Strategie zu finden, welche die Belohnungen, die im Verlauf der Zeit gesammelt werden, maximiert. Zur Lösung wird die Bellman-Gleichung für dieses Problem formal hergeleitet und Bedingungen formuliert, unter denen eine optimale Strategie existiert.

Basierend auf diesen Formalisierungen stellt die Arbeit verifizierte Versionen des Value-Iteration-Verfahren und des Policy-Iteration-Verfahren bereit, die (nahezu) optimale Strategien berechnen. Die hier entwickelte Formalisierung kann in Zukunft als Basis für Verifikationsprojekte im Zusammenhang mit bestärkendem Lernen und Planungsproblemen in der KI dienen.

Contents

1 Introduction

Markov decision processes (MDPs) are widely used models for probabilistic systems that are partly controlled by a decision-maker. They find applications in numerous research areas like autonomous vehicles, robotics and reinforcement learning. The relevance of MDPs stems from their important role in the very active research area of artificial intelligence, especially in reinforcement learning. As a result, the algorithms presented in this thesis are still in use in slightly modified variants, even though they were already dicovered in the 1950s [17, 3].

MDPs model scenarios where a decision-maker (or agent) finds itself in partial control over a system. The state of the system is influenced by the actions the agent undertakes, but also evolves due to outside events the agent cannot control. Over time, MDPs award the agent with rewards depending on the actions chosen and the current state of the modeled system. The goal of the agent is to discover a strategy for action selection that maximizes rewards in the long run. The study of MDPs has lead to algorithms that automatically find optimal or nearly optimal strategies for a large class of MDPs.

Hölzl [11] provides an extensive formalization of MDPs and Markov chains in the interactive theorem prover Isabelle/HOL. We build on these developments to define and analyse the expected total discounted reward criterion. Our formalization culminates in verified implementations of the value iteration and policy iteration algorithms in Isabelle/HOL[1]. These may serve as a basis for a future formal verification of more practical algorithms in the realm of reinforcement learning or planning under uncertainty.

An important characteristic of our work is that the formalization we develop builds on considerable background theory in the areas of probability theory, vector spaces, analysis and linear algebra. As a result, our work has to build on, combine and extend several prior verification efforts carried out in Isabelle/HOL. While this fact introduces challenges e.g. for automation in proofs, this formalization shows that integrating the different Isabelle/HOL libraries is very feasible.

1.1 Related Work

Vajja et al. developed a formalization of the value iteration algorithm on finite MDPs in the interactive theorem prover Coq [18]. They provide a convergence proof for the algorithm. Furthermore, they show optimality of value iteration in a context where only stationary, deterministic action choice is permitted. Our formalization differentiates

[1] The source code of the formalization is publicly available for inspection at `https://github.com/schaeffm/ma-mdp`

itself from their work in multiple aspects: we prove a more general notion of optimality with regard to randomized history-dependent policies. Furthermore, most of our formalization is not restricted to MDPs with finite state and action spaces.

Our work builds on the formalization of Markov (decision) processes and Markov chains in Isabelle/HOL by Hölzl [11, 12]. These developments formalize Markov processes over discrete and continuous time and analyse among other things stationary distributions, fairness, and reachability. Wimmer et al. [19] then use the resulting library to formalize probabilistic timed automata.

There exists a variety of (unverified) tools that can execute the value or policy iteration algorithm on MDPs [5, 2]. These projects also provide more efficient and sophisticated algoritms to find optimal policies.

1.2 Contributions

The thesis makes the following key contributions:

- As a first step, we extend the existing formalization of Markov decision processes in Isabelle/HOL with rewards to define the optimal value of an MDP.
- We give denotational semantics for MDPs to prove that markovian policies can simulate general policies.
- We define and prove correct the value iteration and policy iteration algorithms. In the process, we show how to compute the value of an MDP and derive an optimal policy using fixpoint methods.
- Finally, we extract executable code for both algorithms and experimentally evaluate their performance.

1.3 Outline

In Chapter 2, we begin with an introduction to probability theory and the theory of vector spaces as they are formalized in Isabelle/HOL. Next, we introduce Markov decision processes and their formal definition in Chapter 3. This includes a hierarchy of different types of decision rules and policies. We also give both an operational and a denotational view on the trace space of MDPs. In Chapter 4, we continue to define the expected discounted total reward criterion, which in turn allows us to assign an optimal value to MDPs. As a final step of the analysis of MDPs, we give conditions under which this value can be attained by a concrete policy and give a fixed-point iteration to approximate the optimal value. These observations directly lead to a formalization of the value iteration algorithm in Chapter 5, a method to find close to optimal policies. We also prove correctness of the policy iteration algorithm (Chapter 6), an alternative algorithm that often terminates in fewer iterations than value iteration. Subsequently, in Chapter 7,

we compare both algorithms based on the code extracted from the formalization. We close with an outlook on potential future formalizations that build on our work in Chapter 8.

2 Preliminaries

Our formalization of the value iteration and policy iteration algorithm is carried out in the interactive theorem prover Isabelle/HOL. Our work builds on the extensive Isabelle/HOL library and formalizations available in the AFP (Archive of Formal Proofs[1]). We give a short introduction to the core concepts of Isabelle/HOL and the libraries used in this thesis. Except for Theorem 2.2 and the paragraph on bounded functions, the developments presented in this chapter are not our own but are part of the Isabelle distribution.

2.1 Isabelle/HOL

Isabelle/HOL provides several concepts which differ from standard pen-and-paper mathematics to facilitate the efficient formalization of mathematics. We present the notation and core concepts of Isabelle/HOL [15] .

Notation The notation used throughout the thesis is based on Isabelle/HOL, with slight adaptations to improve readability. Function application is juxtaposition as common in functional programming, e.g. $f\ x\ y$ as opposed to the usual $f(x, y)$. Anonymous functions are declared with a leading λ followed by the parameters and the function body: $\lambda x.\ x + 1$.

We use the notation $\epsilon x.\ P\ x$ for the indefinite *Hilbert choice* operator. It is used as a placeholder for some arbitrary but fixed term that satisfies P. Only if P is satisfiable, the placeholder is well-defined: $\exists x.\ P\ x \longleftrightarrow P\ (\epsilon x.\ P\ x)$. Similarly, the definite description operator $\iota x.\ P\ x$ is a substitute for the unique term that satisfies P. A proof of $P\ (\iota x.\ P\ x)$ can be reduced to proving that there is a unique x such that $P\ x$: $\exists! x.\ P\ x \Longrightarrow P\ (\iota x.\ P\ x)$.

Finally, we introduce some more special notation. $\sum_i f\ i$ sums $f\ i$ over all natural numbers i. We write $\bigsqcup X$ for the supremum (the least upper bound) of the set X. UNIV is the set that includes every member of a type, whereas $-X$ is the set complement of X.

Types Terms in Isabelle/HOL are classified by assigning them a type. Types can be interpreted as sets of terms with common structure or properties. Each type has at least one inhabitant. The most basic type we use is bool, the type of truth values with the two members True and False. We can assert that a term has a certain type using type annotations such as $x :: \text{bool}$. Natural numbers are of type nat. We can use the principle

[1] `https://www.isa-afp.org`

of induction to prove theorems for all natural numbers n, where we distinguish the two cases $n = 0$ and $n = m + 1$ for some natural number m. Similarly, (recursive) function definitions can treat those two cases separately. A second numeric type important for our formalization is the type real of real numbers. The set of real numbers with infinity $\mathbb{R} \cup \{-\infty, +\infty\}$ corresponds to the type ereal.

The basic types can be systematically combined into more complex types like pairs or lists. These constructions work mostly independent of the concrete types. Type variables are written in greek letters $\alpha, \beta, \ldots$ and serve as placeholders for arbitrary concrete types. We use type variables to introduce the type of functions from α to β: $\alpha \Rightarrow \beta$. For functions $f :: \mathsf{nat} \Rightarrow \alpha$ we use a subscript notation, e.g. f_i for $f\ i$. The type α set is the type of sets over α. All pairs of the form (a, b) for $a :: \alpha$ and $b :: \beta$ are members of the type $\alpha \times \beta$. The projections $\mathsf{fst} :: \alpha \times \beta \Rightarrow \alpha$ and $\mathsf{snd} :: \alpha \times \beta \Rightarrow \beta$ provide access to the first and second element of a pair respectively. Tuples with more elements are just nested pairs, e.g. (a, b, c) means $(a, (b, c))$.

The type α list represents finite lists with members of type α. Its members are either the empty list [] or a single element prepended to another list, i.e. $x \cdot xs$ where $x :: \alpha$ and $xs :: \alpha$ list. Streams are infinite lists, the type α stream is isomorphic to sequences $\mathsf{nat} \Rightarrow \alpha$. Terms of type stream have the form $x \cdot xs$, so they can always be decomposed into a head x and another infinite stream xs. As with sequences, ω_n is the n-th element of the stream ω.

Locales Mathematics textbooks often introduce a context of constants associated with assumptions such as "For the remainder of the chapter, fix a c s.t. $0 \leq c < 1$.". *Locales* are the Isabelle/HOL way to provide such contexts. They allow us to fix constants and assumptions on these constants in a locale declaration. Within a locale, we may then carry out a formalization that uses the assumptions and constants. Finally, locales may be instantiated with definitions that satisfy the locale assumptions to obtain the theorems derived within the locale.

Type Classes Isabelle/HOL supports *type classes* to group types which provide common operations satisfying a set of axioms. For example, the type class group-add is the class of additive groups. Type classes are implemented as a special form of locales. Types can become members of a type class via *instantiations*, where we provide definitions for the operations of the type class and proofs of the corresponding axioms. Type variables may be annotated with type classes, e.g. $\alpha :: \mathsf{complete\text{-}space}$. In this case, α may only be instantiated to types that are instances of complete-space. Using type classes, theorems and notation can be shared among a range of concrete types, which reduces code duplication.

Type Definitions With the **typedef** command, a new type can be defined as a nonempty subset of an existing type. This way to define a subtype is called a type definition. The following example shows how to define the type of all even natural numbers:

typedef nat-even $= \{x :: \text{nat} \mid \text{even } x\}$

The **typedef** command generates coercions between the existing and the newly defined type. Isabelle/HOL provides a lifting mechanism that allows to define functions on the whole type and then automatically lift them to the subtype. The user has to provide proof that the range of the defined function satisfies the invariant of the subtype.

2.2 Probability Theory

The current version of the probability theory library for Isabelle/HOL was developed primarily by Hölzl [10]. It is based on an extensive formalization of measure and integration theory by Hölzl and Heller [13]. We give a short introduction to the measure-theoretic view on probability theory as formalized in Isabelle/HOL.

Measure Spaces Measure theory studies how to assign a meaningful measure (a non-negative real number) to sets. An important measure for euclidean spaces $\mathbb{R}^n$ is the *Lebesgue* measure, which assigns to each subset that forms a geometric object its volume. A general measure can be thought of as a map from sets to their mass or volume.

For finite spaces, the counting measure assigns to each subset its cardinality. With more general uncountable spaces, however, several issues arise that make it impossible to assign a meaningful measure to each subset. Thus we define measures only for selected subsets of a space.

A *measure space* over elements of type α is a triple $(\Omega, \mathcal{A}, \mu)$, where $\mathcal{A} :: \alpha$ set set is a σ-algebra on the space $\Omega :: \alpha$ set. A σ-algebra on a space Ω is a set of subsets of Ω which contains the empty set and is closed under complement and countable union. The members of $\mathcal{A}$ are called *measurable* sets. $\mu :: \alpha$ set $\Rightarrow$ ereal is required to be a measure, i.e. it is non-negative, $\mu\, \emptyset = 0$, and countably additive on measurable sets. Countable additivity is fulfilled if the measure of a countable union of disjoint measurable sets equals the sum of the individual sets.

The type α measure is defined using **typedef** as all such triples which satisfy the axioms of a measure space. It comes with the projections space, sets, and emeasure to extract Ω, $\mathcal{A}$, or μ respectively. If all measures are finite for a measure space, we use measure :: α measure $\Rightarrow$ α set $\Rightarrow$ real instead of emeasure.

For countable spaces Ω, we may use the measure count-space Ω, where every subset of Ω is measurable. The measure of a set equals its cardinality. For uncountable Ω, it is often impossible to assign a consistent measure to all subsets. On type real we use the borel-measure, based on the smallest σ-algebra that contains all open sets. For intervals, the borel-measure coincides with their length.

When measure M (space M) $= 1$, M is a probability space, denoted by prob-space M. In this context, measurable sets are also called *events*, the measure of an event is its *probability*.

Random Variables & Distributions In the measure-theoretic view on probability theory, a *random variable* f is a *measurable function* from a probability space to another measure space. A function $f \in M \to_M N$ is called measurable if the preimages of all measurable sets are again measurable. This property is desirable because it means that we can assign a probability measure $M\ (f^{-1}\ X)$ to all measurable sets X in N. A function whose domain is a count-space is trivially measurable.

If M is a probability space, a random variable $f \in M \to_M N$ gives rise to a probability space on N via distr $M\ N\ f$. distr $M\ N\ f$ returns a new measure space with the sets and space inherited from N, but replaces the measure with the *push-forward* measure of f:

$$\text{measure}\ (\text{distr}\ M\ N\ f)\ X = \text{measure}\ M\ \{x \mid f\ x \in X\}$$

Expected Values We use *Lebesgue integrals* to compute expected values. On each measure space M, the integral assigns a real number to *integrable* functions $f \in M \to_M$ borel, those for which the negative and positive parts of the function integrate to finite values. For probability spaces, in particular, all bounded and measurable functions are integrable. Notation for the integral of a function f over the space M is $\int f\ \partial M$ or $\int_x f\ x\ \partial M$. The integral is monotone and linear. For a counting space with finite Ω, the integral simplifies to a sum: $\int f\ \partial\text{count-space}\ \Omega = \sum_{x\in\Omega} f\ x$. Integration of distr follows a simple rule:

$$\textbf{assumes}\quad g \in M \to_M N \quad \textbf{and} \quad f \in N \to_M \text{borel}$$
$$\textbf{shows}\quad \int f\ \partial\text{distr}\ M\ N\ g = \int_x f\ (g\ x)\ \partial M$$

Almost Everywhere In the context of a measure space $M :: \alpha$ measure, a predicate $P :: \alpha \Rightarrow$ bool is said to hold almost everywhere in M (a.e.) if it holds for all elements of M except for a set with measure 0:

$$\text{AE}\ x\ \text{in}\ M.\ P\ x \longleftrightarrow \exists X \in \text{sets}\ M.\ \text{emeasure}\ M\ X = 0 \wedge (\forall x \in \text{space}\ M \cap -X.\ P\ x)$$

Concerning probability spaces, such an event that occurs with probability 1 happens "almost surely" (a.s.).

Product Spaces & Stream Spaces Given two measure spaces M and N, we may construct a binary product measure $M \bigotimes_M N$. Its space is the cartesian product of the individual spaces space $M \times$ space N. The sets are the smallest σ-algebra that includes the cartesian product of the sets of the components. The measure of a set $A \times B \in$ sets $M \times$ sets N is measure $M\ A \cdot$ measure $N\ B$.

The binary product can be generalized to a finite or infinite number of measure spaces M_i for $i \in I$. We write this product space as $\Pi_{i\in I}\ M_i$. For $I = \text{UNIV} ::$ nat set and M constant we obtain the space of infinite sequences $\Pi_{n::\text{nat}}\ M$. It can be lifted to a space

over streams:

$$\text{stream-space } M = \text{vsigma } (\text{streams } (\text{space } M)) \ (\lambda\omega\ i.\ \omega_i) \ (\Pi_{n::\text{nat}}\ M)$$

where vsigma $\Omega\ f\ M$ is the smallest σ-algebra on Ω that makes f measurable,
streams X is the set of all streams with elements in X.

Giry Monad The Giry monad [7] gives us a way to define complex probability spaces from a combination of simple spaces. It is a monad on probability spaces. Giry monads were formalized in Isabelle/HOL by Eberl et al. [6] who also give a more detailed exposition. For constructing and composing probability spaces, the Giry monad provides the two operations bind (with infix notation _ ⤜ _) and return.

$$\text{return} :: \alpha \text{ measure} \Rightarrow \alpha \Rightarrow \alpha \text{ measure}$$
$$\text{bind} \quad :: \alpha \text{ measure} \Rightarrow (\alpha \Rightarrow \beta \text{ measure}) \Rightarrow \beta \text{ measure}$$

Given a measurable space K, the Giry monad operates on the space prob-algebra K, a measurable space on the probability measures on K. The return operator lifts an element of K to a measure space in prob-algebra K. return $M\ x$ updates the measure of M to the *Dirac* measure, where x happens almost surely. Assume $A \in \text{sets } K$, then

$$\text{measure } (\text{return } M\ x)\ A = \textbf{if } x \in A \textbf{ then } 1 \textbf{ else } 0.$$

For $x \in \text{space } M$ and $f \in M \to_M \text{borel}$, integration of return $M\ x$ reduces to function application:

$$\int f\ \partial\text{return } M\ x = f\ x$$

The bind operator allows expressing the consecutive execution of two random experiments M and N, where the second experiment depends on the outcome of the first: $M \gg\!= (\lambda x.\ N\ x)$. Let M be a probability space, let N be a probability space dependent on the outcome of M, i.e. $N \in M \to_M \text{prob-algebra } K$. We obtain a rule to compute the measure with ⤜ and an integration rule for bounded functions.

assumes $X \in \text{sets } K$
shows $\text{measure } (M \gg\!= N)\ X = \int_x \text{measure } (N\ x)\ X\ \partial M$

assumes $f \in K \to_M \text{borel}$ **and** $\text{bounded } (\text{range } f)$
shows $\int f\ \partial M \gg\!= N = \int_x \int f\ \partial N\ x\ \partial M$

Computations involving bind and return are written in **do**-notation as established by the functional programming language Haskell. This helps to improve readability of terms

involving nested applications of bind. We show how this notation can be recursively desugared to return and bind:

$$\textbf{do}\{\ \mathsf{return}\ M\ x\ \} = \mathsf{return}\ M\ x$$
$$\textbf{do}\{\ x \leftarrow P;\ \langle Q\ x\rangle\ \} = P \gg\!= (\lambda x.\ \textbf{do}\{\ \langle Q\ x\rangle\ \})$$

Probability Mass Functions Most of the thesis is concerned with *discrete probability spaces*, where all sets are measurable and the measure of a set equals the sum of the measures of its members. A discrete probability space can thus be represented by probability mass functions which only assign probabilities to individual elements of the space. In Isabelle/HOL, discrete probability spaces form the type pmf of discrete probability distributions, defined as a subtype of measure:

$$\textbf{typedef}\ \alpha\ \mathsf{pmf} =$$
$$\{M \mid \mathsf{prob\text{-}space}\ M \wedge \mathsf{sets}\ M = \mathsf{UNIV} \wedge (\exists S.\ \mathsf{countable}\ S \wedge\ \mathsf{measure}\ M\ S = 1)\}$$

An element of α pmf can be coerced into a general measure with the function measure-pmf, which we omit for the most part. Thus the theory developed for ordinary measures carries over to the type pmf. Since functions out of a discrete probability space are trivially measurable, we can drop the measurability assumptions for many theorems. Let $M :: \alpha$ pmf, we present special notation for some concepts familiar from general probability spaces:

$$\begin{aligned}
\mathsf{pmf}\ M\ x &= \mathsf{measure}\ M\ \{x\}\\
\mathsf{set\text{-}pmf}\ M &= \{x \mid \mathsf{measure}\ M\ \{x\} \neq 0\}\\
\mathsf{map\text{-}pmf}\ f\ M &= \mathsf{distr}\ M\ (\mathsf{count\text{-}space}\ \mathsf{UNIV})\ f\\
\mathsf{return\text{-}pmf}\ x &= \mathsf{return}\ (\mathsf{count\text{-}space}\ \mathsf{UNIV})\ x
\end{aligned}$$

We further abbreviate $x \in \mathsf{set\text{-}pmf}\ M$ by $x \in M$.

2.3 Normed Vector Spaces

The Isabelle/HOL library uses type classes to build a hierarchy of vector spaces [14]. Real vector spaces are abelian additive groups that allow scaling by a real number. They form the type class real-vector. Real vectors can be added ($v + u$) or scaled by a real number: $c \cdot_R v$. If there is no possibility for confusion we omit the subscript and write just $c \cdot v$.

A type with an instance for real-vector that provides a norm (as a constant norm) can be made an instance of real-normed-vector. We require that the norm is compatible with the scaling of vectors. The norm of a vector is often induced by its distance to the origin in terms of the metric on the space. The following two properties relate the norm and the metric: $\mathsf{norm}\ x = \mathsf{dist}\ x\ 0$ and $\mathsf{dist}\ x\ y = \mathsf{norm}\ (x - y)$. Next, we specify what it means

for a sequence of real normed vectors $f :: (\mathsf{nat} \Rightarrow \alpha :: \mathsf{real\text{-}normed\text{-}vector})$ to converge to a limit $l :: \alpha$:

$$(f \longrightarrow l) \quad \longleftrightarrow \quad (\forall \epsilon > 0.\ \exists N.\ \forall n \geq N.\ \mathsf{dist}\ f_n\ l < \epsilon)$$
$$\mathsf{lim}\ f = (\iota l.\ f \longrightarrow l)$$

The type class complete-space is a subclass of real-normed-vector. It asserts that every Cauchy sequence converges to a limit. For this class of types, the *Banach fixed-point theorem* states that every contraction mapping has a unique fixed point that can be computed by repeated application of that mapping.

Theorem 2.1 (Banach Fixed-Point Theorem)**.**

$$\begin{aligned} \textbf{fixes} \quad & f :: (\alpha :: \mathsf{complete\text{-}space}) \Rightarrow \alpha \\ \textbf{assumes} \quad & 0 \leq c < 1 \quad \textbf{and} \quad \forall x\ y.\ \mathsf{dist}\ (f\ x)\ (f\ y) \leq c \cdot \mathsf{dist}\ x\ y \\ \textbf{shows} \quad & \exists! x.\ f\ x = x \quad \textbf{and} \quad f^n\ x \longrightarrow (\iota x.\ f\ x = x) \end{aligned}$$

Bounded Functions We call a function $f :: \alpha \Rightarrow (\beta :: \mathsf{real\text{-}normed\text{-}vector})$ a bounded function if the absolute value of all function values never exceeds a fixed constant. In Isabelle/HOL, boundedness of a set is expressed with the predicate bounded. The range of the function f is just range f. So f is a bounded function if and only if it satisfies bounded (range f). We use a type definition to construct the type of bounded functions $\alpha \Rightarrow_b \beta$ whose domain is of type α and range is of type β. This type is a small generalization of an existing type for bounded continuous functions in the Isabelle/HOL library.

$$\textbf{typedef}\ \alpha \Rightarrow_b (\beta :: \mathsf{real\text{-}normed\text{-}vector}) = \{f :: \alpha \Rightarrow \beta \mid \mathsf{bounded}\ (\mathsf{range}\ f)\}$$

Bounded functions are an instance of real-vector, as we can define addition and scaling of bounded functions pointwise: $(f + g)\ x = f\ x + g\ x$. We can also define the uniform or supremum metric on bounded functions:

$$\mathsf{dist}\ f\ g = \bigsqcup x.\ \mathsf{dist}\ (f\ x)\ (g\ x)$$

With this definition we can make bounded functions an instance of the type class real-normed-vector. Finally, for $\beta ::$ complete-space we show that $\alpha \Rightarrow_b \beta$ is a complete space as well.

Bounded Linear Functions Bounded linear functions are linear transformations between instances of real-normed-vector. Moreover, they only perform bounded scaling on the vectors. We use the existing definition from the Isabelle/HOL library which again employs a type definition to create the subtype of bounded linear functions from the ordinary function type.

$$\begin{aligned} &\textbf{typedef}\ (\alpha :: \mathsf{real\text{-}normed\text{-}vector}) \Rightarrow_L (\beta :: \mathsf{real\text{-}normed\text{-}vector}) = \\ &\quad \{f \mid \mathsf{linear}\ f \wedge \exists c.\ \forall x.\ \mathsf{norm}\ (f\ x) \leq c \cdot \mathsf{norm}\ x\} \end{aligned}$$

Linearity of a function f (written $\mathsf{linear}\ f$) requires that both $f\ (v + u) = f\ v + f\ u$ and $f\ (c \cdot_R v) = c \cdot_R f\ v$ are satisfied. The identity function is bounded linear and called id_L. Bounded linear functions are closed under composition, which is written $\_ \circ_L \_$. f^n denotes the n-fold application of f.

For discrete types, bounded linear functions correspond to matrices. Bounded linear functions are real normed vectors as well. As for bounded functions, addition and scaling are again defined pointwise. We define the norm of a bounded linear function as the largest ratio that an argument is stretched by:

$$\mathsf{norm}\ (f :: \alpha \Rightarrow_L \beta) = \bigsqcup x.\ \mathsf{norm}\ (f\ x) / \mathsf{norm}\ x$$

We finish the discussion of bounded linear transformations with a theorem that states a generalized version of the closed-form formula for geometric series [16, Corollary C.4]. It plays an important role in our upcoming analysis of MDPs.

Theorem 2.2 (Geometric Series of Bounded Linear Transformations)**.**

assumes $\mathsf{norm}\ Q < 1$

shows $(\mathsf{id}_L - Q) \circ_L \sum_i Q^i = \mathsf{id}_L$ **and** $\sum_i Q^i \circ_L (\mathsf{id}_L - Q) = \mathsf{id}_L$

3 Markov Decision Processes

Markov decision processes (MDPs) are a powerful tool to model systems that show both deterministic and stochastic behavior. In particular, they allow representing systems that consist of two components: an *outside world* and an *agent* (also called decision-maker). To the agent, the outside world appears to exhibit randomized behavior, which the agent may influence to a certain degree by choosing to perform specific actions. In that way MDPs are an extension of Markov chains, those can only model purely stochastic systems. MDPs were first studied by Bellman [4] who devised inventory control as a possible application. We provide a simple version of this problem as a motivating example.

Imagine a business selling goods to customers, which can restock its inventory only at the beginning of each month. Customer demand is considered to follow some random distribution while the process of restocking may be influenced deterministically by placing orders. The theory formalized in this thesis can be used to derive an optimal behavior for the purchasing department to maximize profits.

When modeling a system, its possible configurations are abstracted to a set of states in the MDP. For inventory control, the current state is the number of items in stock, so the natural numbers are a suitable state space. An MDP also specifies a set of actions for each state, a subset of the action space of the MDP. These are the decisions available to the agent. In our example, an action corresponds to the number of goods that should be bought from the supplier. Thus a fitting action space is again the set of all natural numbers. However, not all actions are available in each state. One reason might be the limited capacity of the warehouse, which could place a bound on the order volume depending on the current stock.

The MDPs we consider start from an initial state and then develop in discrete time steps. At each time step, the agent observes the current state, then chooses an available action. The strategy the agent applies for action selection is called a policy. In general, a policy can depend on past observations. Once an action is chosen, a randomized transition to a successor state takes place. The transition probabilities are dependent only on the current state and action, hence the name *Markov* decision process. In the inventory control example, the successor state is the number of items in stock at the beginning of the next month. The transition probabilities are determined based on the current stock, consumer demand, and the amount of restocking that takes place. The process as presented so far describes a single *decision epoch*. In this work we focus on infinite-horizon MDPs, where the agent goes through an infinite number of decision epochs, thereby generating an infinite walk or trace through the MDP. We analyze the distribution of those infinite traces.

Throughout this chapter, we first define MDPs and the agent formally, then determine the distribution of traces of an MDP. Finally, we show that markovian policies (which take only the current state into account) can closely simulate general policies. Section 3.1, 3.3, and 3.4 present an adaptation of a formalization by Hölzl [11, 12], the other sections are own developments based on an introduction to MDPs by Puterman [16].

3.1 Definition of MDPs

Based on the informal description above, we develop a formal definition of MDPs with discrete state and action spaces in Isabelle/HOL. MDPs are formalized as a locale MDP. The assumptions and fixed variables of the locale provide the context for the rest of the chapter. Our definition closely follows the existing formalization of MDPs by Hölzl [11, 12]. We cannot directly reuse their formalization as it does not allow for stochastic action choice in policies. It remains a future task to unify the different implementations of MDPs and Markov processes for Isabelle/HOL (see also Section 7.2, 8.1).

Definition 3.1 (Discrete Markov Decision Process)**.** A Markov decision process is defined as the locale MDP:

locale MDP =
  fixes σ :: countable
  and α :: countable
  and $A :: \sigma \Rightarrow \alpha$ set
  and $K :: \sigma \times \alpha \Rightarrow \sigma$ pmf
assumes A-ne: $\forall s.\ A\ s \neq \emptyset$

The discrete types σ and α represent the state and action spaces. The function A specifies the subset of available actions for every system state: $A\ s$ denotes the set of actions enabled in state s. Through A-ne, we assume that there exists a feasible or enabled action in each state. Finally, the Markov kernel K describes the probabilistic transition system corresponding to the MDP. For each state-action pair, it specifies the transition probabilities to successor states: pmf $(K\ (s, a))\ s'$ is the probability to move to state s' after choosing action a in state s. To obtain the results derived in this work for a concrete MDP, one has to provide definitions for A and K together with a proof of A-ne.

3.2 A Model of the Agent

We now proceed to precisely model the second component of an MDP: the agent. Abstracting from irrelevant details, the policy it uses for action selection characterizes the agent fully.

3.2.1 Decision Rules

The agent can influence the behavior of the MDP by virtue of its ability to select an action at each decision epoch. The decision is not necessarily deterministic. The agent may decide on a probability distribution over actions from which the actual action then gets sampled. Additionally, the action selection is based not only on the current state but also on the history of states visited and actions chosen in the past. In Isabelle/HOL, such a history is of type $(\sigma \times \alpha)$ list. It is a list of state-action pairs. For each possible history, the agent gives a *decision rule*, a mapping from the current state to a probability distribution over enabled actions. A decision rule has type (σ, α) dec, an abbreviation for $\sigma \Rightarrow \alpha$ pmf. Not all decision rules are feasible, as valid decision rules have to respect the enabled actions in each state.

Definition 3.2 (Decision Rule)**.** A (randomized) decision rule is a function $d :: (\sigma, \alpha)$ dec which satisfies the predicate is-dec. We collect all valid randomized decision rules in the set D_R.

$$\text{is-dec } d \longleftrightarrow \forall s.\ \text{set-pmf } (d\ s) \subseteq A\ s \qquad\qquad D_R = \{d \mid \text{is-dec } d\}$$

In many practical applications, we seek deterministic decision rules, as they are simpler to implement and analyze. Deterministic decision rules are functions from states to enabled actions. The conversion function mk-dec-det $d\ s = \text{return-pmf } (d\ s)$ transforms deterministic decision rules into randomized decision rules with probability 1 for a single action in each state. We use the compact notation d_D for mk-dec-det d.

Definition 3.3 (Deterministic Decision Rule)**.** A deterministic decision rule is a function $d :: \sigma \Rightarrow \alpha$, which satisfies is-dec-det. We collect all deterministic decision rules in the set D_D.

$$\text{is-dec-det } d \longleftrightarrow \forall s.\ d\ s \in A\ s \qquad\qquad D_D = \{d \mid \text{is-dec-det } d\}$$

We show that for any MDP, the agent always has at least one decision rule to choose from. Otherwise, our objective to optimize the strategy of the agent would be meaningless. Due to the assumption A-ne, it is clear that a deterministic decision rule always exists. It immediately follows that a randomized decision rule also exists.

Theorem 3.4 (Existence of Decision Rules)**.**

$$\exists d.\ \text{is-dec-det } d \qquad\qquad \exists d.\ \text{is-dec } d$$

3.2.2 Policies

The preferences of an agent may evolve over time, possibly influenced by its past decisions and observations. Thus a single decision rule is in general not sufficient to express the decision-making process over multiple epochs. Instead, a different decision rule can be specified for each history. Such a collection of decision rules is

called a *policy*. We introduce a type synonym for policies: (σ, α) pol is a shorthand for $(\sigma \times \alpha)$ list $\Rightarrow$ (σ, α) dec. As apparent from the types, a policy assigns a decision rule to each history.

Definition 3.5 (Policy)**.** A (history-dependent randomized) policy is a function $p ::$ (σ, α) pol from histories to decision rules. We call the set of all valid policies Π_{HR}.

$$\text{is-policy } p \longleftrightarrow \forall h.\ \text{is-dec } (p\ h) \qquad\qquad \Pi_{HR} = \{p \mid \text{is-policy } p\}$$

History-dependent policies model very general decision-making. Unfortunately, they are also hard to analyze. With history-dependent policies, the outcome of a single decision epoch possibly depends on all past epochs. In many applications, policies with a simpler structure are known to deliver optimal results. Following Puterman [16], we introduce *markovian* policies as a class of more restricted policies. Markovian policies base action selection only on the current state and decision epoch. They do not depend on the history, therefore such policies are also called *memoryless*. We represent markovian policies as sequences of decision rules. They come in a deterministic and randomized variant.

Definition 3.6 (Markovian Policy)**.** Markovian (randomized) policies form the set Π_{MR}. A markovian policy is a function $p ::$ nat $\Rightarrow$ (σ, α) dec which satisfies is-dec p_n for all $n ::$ nat.

$$\Pi_{MR} = \{p \mid \forall n.\ \text{is-dec } p_n\}$$

Definition 3.7 (Markovian Deterministic Policy)**.** Markovian deterministic policies form the set Π_{MD}. A markovian deterministic policy is a function $p ::$ nat $\Rightarrow \sigma \Rightarrow \alpha$ which satisfies is-dec-det p_n for all $n ::$ nat.

$$\Pi_{MD} = \{p \mid \forall n.\ \text{is-dec-det } p_n\}$$

An important subclass of markovian policies are *stationary policies*. They do not vary over time but apply the same decision rule at each epoch. As such, stationary policies can be represented by a single decision rule. Similarly to mk-dec-det, we introduce conversion functions to convert policies with a special structure to general policies, e.g. $d_{SR} = \lambda\_.\ d$ is the randomized decision rule d transformed into a policy. p_{MR} denotes a markovian policy in a context where a general policy is expected.

We later show that we can often restrict ourselves to markovian policies, as they can closely approximate history-dependent policies. The policy and value iteration algorithm that are presented and verified in this thesis may even restrict their search space to deterministic stationary policies.

3.3 Trace Space of the MDP

Fixing a policy and an initial state for the MDP induces a stochastic process. It completely determines the distribution of infinite walks or traces through the MDP the agent will

encounter. We call the probability space corresponding to this distribution the *trace space*. A single trace is a sequence of state-action pairs, at each decision epoch it records both the state of the MDP and the decision of the agent.

The trace space is constructed to model the following setting: the system starts in a random initial state sampled from a distribution S_0. The agent observes the initial state as s, then assigns probabilities to actions according to its policy. Next, an action a is sampled from this distribution. Finally, the system moves to a successor state s' according to the transition probabilities $K\ (s, a)$.

Here, another action is chosen. This time the agent can base the decision both on the current state s' and on the current history $[(s, a)]$, the state-action pair from the last epoch. Afterward, the agent transitions to another state where it bases the decision on the extended history $[(s, a), (s', a')]$. The process repeats infinitely often, once for each decision epoch, where the decision epochs are indexed by the natural numbers. In the limit, the histories that are built up become sequences or streams, i.e. infinite lists of state-action pairs. They are the traces or walks through the MDP. This section studies the probability space formed by the traces of the MDP, as constructed in Isabelle/HOL by Hölzl [12].

To obtain a probability distribution over the traces of an MDP, we fix a probability space of initial states $S_0 :: \sigma$ pmf and a policy $p :: (\sigma, \alpha)$ pol. We derive a stochastic process and show that it behaves like the informal explanation above. We start with a definition of the law of the stochastic process. It is a function that takes a history of length n and returns a probability space over the next state-action pair of the trace, which shows how to advance the MDP by a single decision epoch. This task can be subdivided into two steps: first, determine the state of the next epoch, then its action.

We define the state distribution in the next epoch as $\mathsf{S}_{\mathsf{step}}\ S_0\ h$: if the history is empty, it is just S_0. Otherwise, the state distribution is $K\ (s, a)$, where (s, a) is the last (most current) element of the history.

Definition 3.8.

$\mathsf{S}_{\mathsf{step}} :: \sigma$ pmf $\Rightarrow (\sigma \times \alpha)$ list $\Rightarrow \sigma$ pmf
$\mathsf{S}_{\mathsf{step}}\ S_0\ h =$ **case** reverse h **of** $sa \cdot \_ \Rightarrow K\ sa \mid [] \Rightarrow S_0$
where reverse :: β list $\Rightarrow \beta$ list reverses a list.

To extend $\mathsf{S}_{\mathsf{step}}$ to a state-action distribution, we sample a state s from $\mathsf{S}_{\mathsf{step}}\ S_0\ h$ and can then use the decision rule $p\ h$ to obtain the state-action distribution. Deriving a state-action distribution from a state distribution and a decision rule will be of importance later, so we introduce a proper definition P_{X}. The law of the stochastic process is the composition of P_{X} after $\mathsf{S}_{\mathsf{step}}$.

Definition 3.9.

$\mathsf{P}_{\mathsf{X}} :: (\sigma, \alpha)$ dec $\Rightarrow \sigma$ pmf $\Rightarrow (\sigma \times \alpha)$ pmf
$\mathsf{P}_{\mathsf{X}}\ d\ S_0 =$ **do**$\{\ s \leftarrow S_0;\ a \leftarrow d\ s;$ return-pmf $(s, a)\ \}$

Definition 3.10.

$$\mathsf{P}_{\mathsf{step}} :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma\ \mathsf{pmf} \Rightarrow (\sigma \times \alpha)\ \mathsf{list} \Rightarrow (\sigma \times \alpha)\ \mathsf{pmf}$$
$$\mathsf{P}_{\mathsf{step}}\ p\ S_0\ h = \mathsf{P}_{\mathsf{X}}\ (p\ h)\ (\mathsf{S}_{\mathsf{step}}\ S_0\ h)$$

Using this definition, we can already define the distribution of finite traces of arbitrary length by repeated iteration of $\mathsf{P}_{\mathsf{step}}$. However, we are interested in the distribution of infinite traces. The *Ionescu-Tulcea Extension theorem* gives us just that: It turns a sequence of Markov kernels into an infinite product space IT for which a projection onto prefixes agrees with the distribution of finite traces.

Ionescu-Tulcea Extension Theorem We use the formalization of the Ionescu-Tulcea Extension theorem by Hölzl [12]. The formalization assumes a context with a sequence of Markov kernels κ_i and a sequence of measures M_i. A Markov kernel is a function that generates a probability space depending on a history of events. Formally, we express this as follows: A function $\kappa_i :: (\mathsf{nat} \Rightarrow \beta) \Rightarrow \beta\ \mathsf{measure}$ is called a Markov kernel, if it only depends on the input sequence upto but excluding i. Furthermore, for each history, κ_i has to return a probability space. These properties are ensured by the measurability assumption $\kappa_i \in (\Pi_{j<i}\ M_j) \to_M \mathsf{prob\text{-}algebra}\ M_i$
Given a history of length n, we can use the markov kernels κ to obtain a probability space over histories of length $n + m$:

$$\mathsf{C} :: \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow (\mathsf{nat} \Rightarrow \beta) \Rightarrow (\mathsf{nat} \Rightarrow \beta)\ \mathsf{measure}$$
$$\mathsf{C}\ n\ 0\ \omega = \mathsf{return}\ (\Pi_{i<n}\ M_i)\ \omega$$
$$\mathsf{C}\ n\ (m+1)\ \omega = \mathbf{do}\{\ \omega' \leftarrow \mathsf{C}\ n\ m\ \omega;$$
$$x \leftarrow \kappa_{n+m}\ \omega';$$
$$\mathsf{return}\ (\Pi_{i<n+m+1}\ M_i)\ (\omega'(n+m := x))\ \}$$

Here $\omega(i := x)$ updates ω at position i with x. The Ionescu-Tulcea Extension theorem then shows the existence of an infinite product measure IT which coincides with C for projections onto finite index sets. We only state the special case for index sets of the form $\{0.. < n\} = \{i \mid 0 \le i < n\}$.

Theorem 3.11 (Ionescu-Tulcea)**.**

$$\mathsf{distr}\ (\mathsf{IT}\ M\ \kappa)\ (\Pi_{i<n}\ M_i)\ (\lambda\omega.\ \omega_{|\{0..<n\}}) = \mathsf{C}\ 0\ n\ \bot$$
where $f_{|I}\ x = \mathbf{if}\ x \in I\ \mathbf{then}\ f\ x\ \mathbf{else}\ \bot$.

For discrete MDPs we use the Ionescu-Tulcea Extension theorem with $\beta = \sigma \times \alpha$ and $M_i = \mathsf{count\text{-}space}\ \mathsf{UNIV}$. We instantiate κ with $\mathsf{P}'_{\mathsf{step}}$, which is $\mathsf{P}_{\mathsf{step}}$ adapted to sequences:

$$\mathsf{P}'_{\mathsf{step}}\ n\ \omega = \mathsf{P}_{\mathsf{step}}\ (\mathsf{take\text{-}seq}\ n\ \omega)$$
where take-seq converts the initial elements of a sequence to a list.

The trace space of the MDP is defined as a probability space over infinite sequences using IT on $\mathsf{P}'_{\mathsf{step}}$.

Definition 3.12.

$$\mathsf{T}_{\mathsf{seq}}\ p\ S_0 = \mathsf{IT}\ (\lambda\_.\ \mathsf{count\text{-}space})\ (\mathsf{P}'_{\mathsf{step}}\ p\ S_0)$$

We use C_{p,S_0} for C in the context of MDPs to indicate the policy and initial state. This allows us to restate Theorem 3.11 for MDPs:

Theorem 3.13 (Ionescu-Tulcea for $\mathsf{T}_{\mathsf{seq}}$)**.**

$$\mathsf{distr}\ (\mathsf{T}_{\mathsf{seq}}\ p\ S_0)\ (\Pi_{i<n}\ \mathsf{count\text{-}space})\ (\lambda\omega.\ \omega_{|\{0..<n\}}) = \mathsf{C}_{p,S_0}\ 0\ n\ \bot$$

We directly lift the infinite product space to an isomorphic probability space over streams.

Definition 3.14 (Trace Space)**.**

$$\mathsf{T} :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma\ \mathsf{pmf} \Rightarrow (\sigma \times \alpha)\ \mathsf{stream\ measure}$$
$$\mathsf{T}\ p\ S_0 = \mathsf{distr}\ (\mathsf{T}_{\mathsf{seq}}\ p\ S_0)\ (\mathsf{stream\text{-}space}\ (\mathsf{count\text{-}space}\ \mathsf{UNIV}))\ \mathsf{to\text{-}stream}$$
where $\mathsf{to\text{-}stream} :: (\mathsf{nat} \Rightarrow \beta) \Rightarrow \beta\ \mathsf{stream}$ transforms sequences into streams.

In Isabelle/HOL, we actually construct the trace space for MDPs with general state and action spaces. Here, we specialized all definitions to discrete spaces to simplify the exposition.

3.4 Iteration Rule for the Trace Space

After the execution of the first decision epoch is over, the resulting situation resembles the initial scenario, only the initial state is now different and the policy is modified slightly. This suggests that the trace space T of the MDP follows an iteration rule: first sample the initial state-action pair from $\mathsf{P}_{\mathsf{step}}$, then call T again to obtain the tail of the trace. We now develop an iteration rule that allows us to split off the initial decision epoch. Our proofs are again based on the formalization by Hölzl [12]. First, we observe that $\mathsf{P}_{\mathsf{step}}$ can be computed by recursion on the history:

Theorem 3.15 (Recursive Equations for $\mathsf{P}_{\mathsf{step}}$)**.**

$$\mathsf{P}_{\mathsf{step}}\ p\ S_0\ [] = \mathsf{P}_{\mathsf{X}}\ (p\ [])\ S_0$$
$$\mathsf{P}_{\mathsf{step}}\ p\ S_0\ (sa \cdot hs) = \mathsf{P}_{\mathsf{step}}\ (\pi\text{-}\mathsf{Suc}\ p\ sa)\ S_0\ hs$$
where $\pi\text{-}\mathsf{Suc}\ p\ sa\ h = p\ (sa \cdot h)$ is the successor policy.

The result of π-Suc is again a policy, i.e. it satisfies is-policy. For markovian policies, the successor policy is independent of the initial state-action pair, so we write p_{Suc} instead of $\pi\text{-}\mathsf{Suc}\ p\ sa$. Using these equations, we can derive an iteration rule for finite traces.

Theorem 3.16 (Iteration Rule for C)**.**

$$\begin{aligned}\mathsf{C}_{p,S_0}\ 0\ (n+1)\ \bot = \mathbf{do}\{\ & sa \leftarrow \mathsf{P}_\mathsf{X}\ (p\ [])\ S_0;\\ & \omega \leftarrow \mathsf{C}_{\pi\text{-}\mathsf{Suc}\ p\ sa,\ K\ sa}\ 0\ n\ \bot;\\ & \mathsf{return}\ (\Pi_{i<n+1}\ \mathsf{count\text{-}space})\ (sa \cdot \omega)\ \}\end{aligned}$$

Proof. First we use $\mathsf{C}\ 0\ (m+l)\ \bot = \mathsf{C}\ 0\ m\ \bot \gg\!= \mathsf{C}\ n\ l$ to deduce

$$\mathsf{C}_{p,x}\ 0\ (n+1)\ \bot = \mathsf{P}_\mathsf{X}\ (p\ [])\ x \gg\!= (\lambda sa.\ \mathsf{C}_{p,x}\ 1\ n\ (sa \cdot \bot))$$

To complete the proof, we show by induction on n that

$$\begin{aligned}&\mathsf{C}_{p,x}\ 1\ n\ (sa \cdot \bot) =\\ &\quad \mathsf{C}_{\pi\text{-}\mathsf{Suc}\ p\ sa,\ K\ sa}\ 0\ n\ \bot \gg\!= (\lambda\omega.\ \mathsf{return}\ (\Pi_{i<n+1}\ \mathsf{count\text{-}space})\ (sa \cdot \omega)).\end{aligned}$$

□

This reasoning can be generalized to infinite sequences and thus the iteration rule holds for $\mathsf{T}_{\mathsf{seq}}$. We can then lift the rule to T. For a detailed proof we again refer to Hölzl [12].

Theorem 3.17 (Iteration rule for T)**.**

$$\begin{aligned}\mathsf{T}\ p\ S_0 = \mathbf{do}\{\ & sa \leftarrow \mathsf{P}_\mathsf{X}\ (p\ [])\ S_0;\\ & \omega \leftarrow \mathsf{T}\ (\pi\text{-}\mathsf{Suc}\ p\ sa)\ (K\ sa);\\ & \mathsf{return}\ (\Pi_{i::\mathsf{nat}}\ \mathsf{count\text{-}space})\ (sa \cdot \omega)\ \}\end{aligned}$$

The iteration rule lets us decompose integrals (expected values) over functions on traces of the MDP. The boundedness and measurability assumptions on f ensure that the integral is well-defined and finite. The theorem then follows from the integration rules for bind and return.

Theorem 3.18 (Integral Decomposition for T)**.**

$$\begin{aligned}\textbf{assumes}\quad & \mathsf{bounded}\ (\mathsf{range}\ f)\\ \textbf{and}\quad & f \in (\mathsf{stream\text{-}space}\ (\mathsf{count\text{-}space}\ \mathsf{UNIV}) \rightarrow_M \mathsf{borel}\\ \textbf{shows}\quad & \int f\ \partial\mathsf{T}\ p\ S_0 = \int_{sa}\int_{\omega} f\ (sa \cdot \omega)\ \partial\mathsf{T}\ (\pi\text{-}\mathsf{Suc}\ p\ sa)\ (K\ sa)\ \partial\mathsf{P}_\mathsf{X}\ (p\ [])\ S_0\end{aligned}$$

3.5 A Denotational View on the Trace Space

The view on the trace space presented so far emphasizes individual traces. However, the definitions on MDPs we use in this thesis can do without such a detailed operational view on the trace space: They depend only on the distribution of states and actions at each epoch, not the likelihood of individual traces to occur. We define the state-action distribution P_n for each epoch n, it follows a simple recursive definition. Our formalization shares similarities with Vajjha et al. [18] but was developed independently.

Definition 3.19. Let p be a policy and S_0 be an initial distribution of states. The state-action distribution at epoch $n :: \mathsf{nat}$ is then defined as P_n.

$$\mathsf{P}_n :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma\ \mathsf{pmf} \Rightarrow (\sigma \times \alpha)\ \mathsf{pmf}$$
$$\mathsf{P}_0 \quad p\ S_0 = \mathsf{P}_\mathsf{X}\ (p\ [])\ S_0$$
$$\mathsf{P}_{n+1}\ p\ S_0 = \mathsf{P}_\mathsf{X}\ (p\ [])\ S_0 \gg\!= (\lambda sa.\ \mathsf{P}_n\ (\pi\text{-}\mathsf{Suc}\ p\ sa)\ (K\ sa))$$

P_n agrees with the definition of T. The proof is by induction on n using the iteration rule for T.

Theorem 3.20 (Equivalence of T and P_n)**.**

$$\mathsf{P}_n\ p\ S_0\ n = \mathsf{distr}\ (\mathsf{T}\ p\ S_0)\ (\mathsf{count\text{-}space}\ \mathsf{UNIV})\ (\lambda\omega.\ \omega_n)$$

Using P_n we may easily prove that only enabled actions can occur in the traces of the MDP. The proof again proceeds by induction on n, using the fact that for a decision rule d and $(s, a) \in \mathsf{P}_\mathsf{X}\ d\ S_0$ we can show $a \in A\ s$.

Theorem 3.21. Let $p \in \Pi_{HR}$, then

$$(s, a) \in \mathsf{P}_n\ p\ S_0 \Longrightarrow a \in A\ s$$
$$\mathsf{AE}\ \omega\ \mathsf{in}\ \mathsf{T}\ p\ S_0.\ \mathsf{snd}\ \omega_n \in A\ (\mathsf{fst}\ \omega_n).$$

Alongside P_n, we also define the state and action distributions X_n and Y_n as projections from P_n. $\mathsf{X}_n\ p\ S_0$ is the state distribution at epoch n.

Definition 3.22. For $n :: \mathsf{nat}$, we define

$$\mathsf{X}_n :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma\ \mathsf{pmf} \Rightarrow \sigma\ \mathsf{pmf}$$
$$\mathsf{X}_n\ p\ S_0 = \mathsf{map\text{-}pmf}\ \mathsf{fst}\ (\mathsf{P}_n\ p\ S_0).$$

We can show that the probability mass function associated with X_n behaves as expected.

Theorem 3.23.

$$\mathsf{pmf}\ (\mathsf{X}_n\ p\ S_0)\ s = \mathsf{measure}\ (\mathsf{P}_n\ p\ S_0)\ \{(s', a) \mid s = s'\}$$

The state distribution at epoch 0 is just S_0. For all other epochs, X_{n+1} may be split at both ends: we can decompose X_{n+1} into P_n followed by a transition in the MDP. Alternatively, we can also split off the initial epoch.

Theorem 3.24.

$$\mathsf{X}_0 \quad p\ S_0 = S_0$$
$$\mathsf{X}_{n+1}\ p\ S_0 = \mathsf{P}_n\ p\ S_0 \gg\!= K$$
$$\mathsf{X}_{n+1}\ p\ S_0 = \mathsf{P}_\mathsf{X}\ (p\ [])\ S_0 \gg\!= (\lambda sa.\ \mathsf{X}_n\ (\pi\text{-}\mathsf{Suc}\ p\ sa)\ (K\ sa))$$

The distribution of actions Y_n is defined analogously to X_n.

Definition 3.25.

$$\mathsf{Y}_n :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma\ \mathsf{pmf} \Rightarrow \alpha\ \mathsf{pmf}$$
$$\mathsf{Y}_n\ p\ S_0 = \mathsf{map\text{-}pmf}\ \mathsf{snd}\ (\mathsf{P}_n\ p\ S_0)$$

For markovian policies, the decision rules at each epoch do not depend on the history hence we may express Y_n solely in terms of X_n and the current decision rule. To use a markovian policy p in a context where a general policy is expected, we write p_{MR}.

Theorem 3.26.

$$\mathsf{Y}_n\ p_{MR}\ S_0 = \mathsf{X}_n\ p_{MR}\ S_0 \gg\!= p\ n$$

3.6 Restriction to Markovian Policies

For each history-dependent policy and fixed initial distribution, there exists a markovian policy which produces the same probability spaces P_n. The idea is to replace the original policy with a markovian policy that averages the action selection of the original policy over all histories for each epoch and state. To obtain the average action at state s in epoch n, we condition P_n on the state s, then we extract the action using map-pmf.

Definition 3.27 (Conditional Action Distribution)**.** For $n :: \mathsf{nat}$ we define

$$\mathsf{Y}^{\mathsf{X}}_n :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma\ \mathsf{pmf} \Rightarrow \sigma \Rightarrow \alpha\ \mathsf{pmf}$$
$$\mathsf{Y}^{\mathsf{X}}_n\ p\ S_0\ s = \mathsf{map\text{-}pmf}\ \mathsf{snd}\ (\mathsf{cond\text{-}pmf}\ (\mathsf{P}_n\ p\ S_0)\ \{(t,a) \mid s = t\})$$
where $\mathsf{cond\text{-}pmf}\ P\ X$ conditions P on the set or event X.

We show that the conditional distribution has the desired probability mass function.

Theorem 3.28.

assumes $s \in \mathsf{X}_n\ p\ S_0$
shows $\mathsf{pmf}\ (\mathsf{Y}^{\mathsf{X}}_n\ p\ S_0\ s)\ a = \mathsf{pmf}\ (\mathsf{P}_n\ p\ S_0)\ (s,a) / \mathsf{pmf}\ (\mathsf{X}_n\ p\ S_0)\ s$

Moreover, P_n can be expressed in terms of X_n and $\mathsf{Y}^{\mathsf{X}}_n$. This fact already suggests that $\mathsf{Y}^{\mathsf{X}}_n\ p\ S_0$ can serve as a markovian policy that simulates p with regard to P_n.

Theorem 3.29.

$$\mathsf{P}_n\ p\ S_0 = \mathbf{do}\{\ s \leftarrow \mathsf{X}_n\ p\ S_0;\ a \leftarrow \mathsf{Y}^{\mathsf{X}}_n\ p\ S_0;\ \mathsf{return\text{-}pmf}\ (s,a)\ \}$$

For markovian policies, the conditional probabilities match the policy as expected.

Theorem 3.30.

assumes $s \in \mathsf{X}_n\ p_{MR}\ S_0$
shows $\mathsf{Y}^{\mathsf{X}}_n\ p_{MR}\ S_0\ s = p\ n\ s$

Given a history-dependent policy p and initial states S_0 we can now define a markovian policy in the following way: For each epoch n, choose as the decision rule the conditional probability distribution $\mathsf{Y}^{\mathsf{X}}_n\ p\ S_0$. The resulting policy is markovian. We will see that it exhibits the same P_n as the original policy. However, the trace space T may be different. This is necessarily so, not all trace spaces that can be achieved by history-dependent policies can also be constructed using markovian policies. Note that the resulting policy also depends on the initial distribution, it is in general impossible to find a corresponding markovian policy that works for all initial distributions simultaneously.

Definition 3.31.

$$\begin{aligned}\mathsf{as\text{-}markovian}\ p\ S_0\ n\ s = {} & \textbf{if}\ s \in \mathsf{X}_n\ p\ S_0 \\ & \textbf{then}\ \mathsf{Y}^{\mathsf{X}}_n\ p\ S_0\ s \\ & \textbf{else}\ \mathsf{return\text{-}pmf}\ (\epsilon a.\ a \in A\ s)\end{aligned}$$

For states which do not occur in the trace space, we choose an arbitrary enabled action with Hilbert choice. In this case $\mathsf{Y}^{\mathsf{X}}_n$ is not well-defined (a distribution conditioned on an event with probability 0). We first show that as-markovian transforms a valid policy into a valid markovian policy.

Theorem 3.32.

assumes $p \in \Pi_{HR}$
shows $\mathsf{as\text{-}markovian}\ p\ S_0 \in \Pi_{MR}$

Proof sketch. We have to show that an enabled action is always selected in each state. Assume $s \in \mathsf{X}_n\ p\ S_0$, the other case is trivial. This means that $\mathsf{Y}^{\mathsf{X}}_n\ p\ S_0\ s$ is defined. As p only selects enabled actions, $\mathsf{set\text{-}pmf}\ (\mathsf{P}_n\ p\ S_0)$ contains only enabled actions by Theorem 3.21. Therefore using Theorem 3.28, $\mathsf{Y}^{\mathsf{X}}_n$ and thus as-markovian select an enabled action with probability 1. □

We are now ready to prove the central theorem of this section: For each policy and initial distribution there exists a markovian policy with identical P_n.

Theorem 3.33.

$$\mathsf{P}_n\ (\mathsf{as\text{-}markovian}\ p\ S_0)_{MR}\ S_0 = \mathsf{P}_n\ p\ S_0$$

Proof sketch. The proof is by induction on the decision epoch n. In the induction step, we rewrite P_n on both sides using Theorem 3.29. It now suffices to prove that

$$\begin{aligned}
&\textbf{do}\{\ s \leftarrow \mathsf{X}_n\ (\text{as-markovian}\ p\ S_0)\ S_0;\\
&\quad\ a \leftarrow \mathsf{Y}_n^{\mathsf{X}}\ (\text{as-markovian}\ p\ S_0)\ S_0\ s;\\
&\quad\ \text{return-pmf}\ (s,a)\ \}\\
=\ &\textbf{do}\{\ s \leftarrow \mathsf{X}_n\ p\ S_0;\\
&\quad\ a \leftarrow \mathsf{Y}_n^{\mathsf{X}}\ p\ S_0\ s;\\
&\quad\ \text{return-pmf}\ (s,a)\ \}
\end{aligned}$$

We simplify $\mathsf{Y}_n^{\mathsf{X}}\ (\text{as-markovian}\ p\ S_0)_{MR}\ S_0\ s$ to match the right-hand side using Theorem 3.30 and the definition of as-markovian. Finally, we use the induction hypothesis and Theorem 3.24 to simplify $\mathsf{X}_n\ (\text{as-markovian}\ p\ S_0)_{MR}\ S_0$ and finish the proof. □

3.7 MDPs with Deterministic Initial States

From now on, we assume that the initial state of an MDP is deterministic, we do no longer allow probability distributions. All results from the previous discussion carry over, as we are now in the special case where the initial state is of the form return-pmf s.

Definition 3.34. The trace space with deterministic initial state is defined as

$$\begin{aligned}
&\mathcal{T} :: (\sigma,\alpha)\ \text{pol} \Rightarrow \sigma \Rightarrow (\sigma \times \alpha)\ \text{stream measure}\\
&\mathcal{T}\ p\ s = \mathsf{T}\ p\ (\text{return-pmf}\ s)
\end{aligned}$$

The statement of the iteration equation for the trace space changes slightly: We can no longer pass a state distribution to $\mathcal{T}$, so the state of the next epoch is determined before the recursive call already.

Theorem 3.35 (Iteration rule for $\mathcal{T}$)**.**

$$\begin{aligned}
\mathcal{T}\ p\ s = \textbf{do}\{\ a\ &\leftarrow p\ []\ s\\
s'\ &\leftarrow K\ (s,a)\\
\omega\ &\leftarrow \mathcal{T}\ (\pi\text{-Suc}\ p\ (s,a))\ s'\\
&\text{return}\ ((s,a)\cdot\omega)\ \}
\end{aligned}$$

Theorem 3.36 (Integration rule for $\mathcal{T}$)**.**

$$\textbf{assumes}\quad \text{bounded}\ (\text{range}\ f)\quad \textbf{and}\quad f \in S \to_M \text{borel}$$
$$\textbf{shows}\quad \int f\ \partial\mathcal{T}\ p\ s = \int_a \int_t \int_\omega f\ ((s,a)\cdot\omega)\ \partial\mathcal{T}\ (\pi\text{-Suc}\ p\ (s,a))\ t\ \partial K\ (s,a)\ \partial p\ []\ s$$

We transfer the most important results and the definitions of P_n, X_n, Y_n, and P_{X}. The new definitions shadow the old ones.

Definition 3.37.

$$\mathsf{P}_n\ p\ s := \mathsf{P}_n\ p\ (\mathsf{return\text{-}pmf}\ s) \qquad \mathsf{X}_n\ p\ s := \mathsf{X}_n\ p\ (\mathsf{return\text{-}pmf}\ s)$$
$$\mathsf{Y}_n\ p\ s := \mathsf{Y}_n\ p\ (\mathsf{return\text{-}pmf}\ s) \qquad \mathsf{P}_\mathsf{X}\ d\ s := \mathsf{P}_\mathsf{X}\ d\ (\mathsf{return\text{-}pmf}\ s)$$
$$\mathsf{as\text{-}markovian}\ p\ s := \mathsf{as\text{-}markovian}\ p\ (\mathsf{return\text{-}pmf}\ s)$$

Theorem 3.38 (Equivalence of $\mathcal{T}$ and P_n)**.**

$$\mathsf{P}_n\ p\ s = \mathsf{distr}\ (\mathcal{T}\ p\ s)\ (\mathsf{count\text{-}space\ UNIV})\ (\lambda\omega.\ \omega_n)$$

4 Expected Total Discounted Reward

Consider again the problem of inventory control for a business. The goal for a business owner is to maximize profits by optimizing the tradeoff between the cost of keeping inventory and the risk of not being able to serve the demand. To model this problem with MDPs, we can label each action in every state with the expected profit over the next month. The owner can now try to maximize profits by optimizing the amount of restocking for each state.

In general, we extend MDPs as follows: each state-action pair is assigned a (possibly negative) real-valued reward. Rewards generalize the notion of profits from the business example. We can make certain state-action pairs more desirable by assigning them a higher reward.

In this chapter, we introduce and analyze MDPs with rewards towards the goal of finding optimal policies. For this extended model of MDPs, we show how to assign a reward to the trace space generated from a policy on an MDP. This allows us to compare and optimize policies to achieve the maximal reward. The definitions and theorems in this chapter closely follow a treatment of MDPs by Puterman [16], specifically we base our formalization on the sections 5.3, 5.4, 6.1, and 6.2.

4.1 Markov Decision Processes with Rewards

Markov decision processes with rewards are formalized as a new locale MDP-reward which extends the locale MDP. In addition to the kernel K and enabled actions A, it fixes two more components. First, we fix a reward function from state-action pairs to the real numbers. We make the standard assumption that the absolute value of rewards is bounded, which simplifies the analysis. Furthermore, we fix a nonnegative discounting factor l. It will be used later on in the definition of the reward of a policy, where it serves as a way to value rewards higher the sooner they are accrued. For the remainder of this chapter, we implicitly assume the context of MDP-reward, i.e. functions l and r which satisfy the assumptions of the locale.

Definition 4.1 (MDP with Rewards)**.**

locale MDP-reward = MDP +
fixes $r :: \sigma \times \alpha \Rightarrow$ real
and l :: real
assumes bounded (range r)
and $0 \leq l < 1$

As the reward function r is bounded, we may find an upper bound on its absolute values. We use r_M to denote the least upper bound or supremum of all absolute values in the image of r.

Definition 4.2 (Maximum reward)**.**

$$\mathsf{r}_\mathsf{M} = \bigsqcup sa.\ |r\ sa|$$

4.2 Expected Total Discounted Reward

This section extends the reward function r to assign a reward to policies, based on which they can be compared and optimized. Let us first consider the problem of assigning a reward or value to an individual trace of the MDP. One obvious definition for the value would be a simple infinite sum over all rewards incurred at the decision epochs of the trace. However, even with bounded rewards, these might not sum to a finite value over the whole trace. The sum of all rewards might be infinite or it could diverge in other ways.

We can circumvent this problem with the introduction of a discounting factor $0 \leq l < 1$. The reward in epoch t is discounted by a factor of l^t. This makes sure the sum always converges and has a nice practical interpretation: the discounting factor represents the shortsightedness of the agent. A myopic agent (l close to 0) values rewards in the near future exceedingly higher than rewards in the far future.

Let p be an arbitrary policy and s be the initial state. We first define the *expected total discounted reward* for a finite horizon N as etr_N. For each trace in the trace space $\mathcal{T}\ p\ s$, we sum the rewards up to epoch N scaled by the discounting factor. Then we take the expected value of this sum over all traces. Note that we only include rewards up to but excluding epoch N, so etr_N denotes the reward accrued at the beginning of decision epoch N.

Definition 4.3 (Discounted Reward over a Finite Horizon)**.** For $N :: \mathsf{nat}$ we define

$$\mathsf{etr}_N :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma \Rightarrow_b \mathsf{real}$$
$$\mathsf{etr}_N\ p\ s = \int_\omega \sum_{i<N} l^i \cdot r\ \omega_i\ \partial\mathcal{T}\ p\ s.$$

We now let the horizon N tend towards infinity in etr_N and define the expected total discounted reward etr as the limit of this process.

Definition 4.4 (Expected Total Discounted Reward)**.**

$$\mathsf{etr} :: (\sigma, \alpha)\ \mathsf{pol} \Rightarrow \sigma \Rightarrow_b \mathsf{real}$$
$$\mathsf{etr}\ p\ s = \mathsf{lim}\ (\lambda N.\ \mathsf{etr}_N\ p\ s)$$

Since r_M is an upper bound for the absolute value of all rewards, the value of each trace is bounded by the geometric series $\sum_i l^i \cdot \mathsf{r}_\mathsf{M} = \mathsf{r}_\mathsf{M}/(1-l)$. This is also a bound for the range of both etr_N and etr.

Theorem 4.5 (Boundedness of etr)**.**

$$\begin{aligned} \mathsf{etr}_N\ p\ s &\leq \mathsf{r_M}/(1-l) & \mathsf{bounded}\ (\mathsf{range}\ (\mathsf{etr}_N\ p)) \\ \mathsf{etr}\ p\ s &\leq \mathsf{r_M}/(1-l) & \mathsf{bounded}\ (\mathsf{range}\ (\mathsf{etr}\ p)) \end{aligned}$$

Hence, both etr and etr_N are bounded functions, so we can lift them to the type bfun. We omit the coercions needed to transfer the bounded functions to normal functions, e.g. for function application. As a result of the lifting, $\mathsf{etr}\ p :: \sigma \Rightarrow_b \mathsf{real}$ is a real-normed-vector, thus in its analysis we can leverage results from the theory of vector spaces.

The limit in the definition of etr and the integral can be swapped, resulting in an infinite sum within the integral. This alternative definition of etr is easier to manipulate, as now the integral decomposition for $\mathcal{T}$ is applicable.

Theorem 4.6.

$$\mathsf{etr}\ p\ s = \int_\omega \sum_i l^i \cdot r\ \omega_i\ \partial\mathcal{T}\ p\ s$$

Proof sketch. The fact follows from Lebesgue's Dominated Convergence Theorem, which in particular states that for a sequence of bounded functions f_n converging pointwise to f and a probability space M

$$(\lambda i.\ \int_x f_i\ x\ \partial M) \longrightarrow \int_x f\ x\ \partial M.$$

□

The goal of the agent is to find a policy that maximizes the expected total discounted reward etr for a specific or all initial states. We define the (optimal) value etr* of an MDP with an initial state as the supremum of etr over all valid policies. etr* can be lifted to bfun since etr p is bounded for all p. As the number of policies may be infinite, a policy that attains the optimal value need not exist in general. Note that we can optimize policies independently for each initial state. If an optimal policy exists for each initial state, there exists a policy which is optimal for all initial states simultaneously.

Definition 4.7 (Value of the MDP)**.**

$$\begin{aligned} &\mathsf{etr}^* :: \sigma \Rightarrow_b \mathsf{real} \\ &\mathsf{etr}^*\ s = \bigsqcup p \in \Pi_{HR}.\ \mathsf{etr}\ p\ s \end{aligned}$$

We call a policy p *optimal* if its value equals etr* for all states: $\mathsf{etr}\ p = \mathsf{etr}^*$. A policy p is called *ϵ-optimal*, if its value is within a distance of ϵ to the optimal value for all states, i.e. if $\mathsf{dist}\ (\mathsf{etr}\ p)\ \mathsf{etr}^* \leq \epsilon$. Due to the definition of the supremum, an ϵ-optimal policy exists for any $\epsilon > 0$.

4.3 Optimality of Markovian Policies

The expected total discounted reward etr is *separable*: it can be expressed as an infinite sum, where each term depends only on a single decision epoch. Specifically, the contribution of each epoch i to etr is the expected value of r over the state-action distribution $\mathsf{P}_i\ p\ s$ of the epoch scaled by the discounting factor.

Theorem 4.8.

$$\mathsf{etr}_N\ p\ s = \sum_{i<N} l^i \cdot \int r\ \partial\mathsf{P}_i\ p\ s \qquad\qquad \mathsf{etr}\ p\ s = \sum_i l^i \cdot \int r\ \partial\mathsf{P}_i\ p\ s$$

Proof. We prove the first theorem by induction on N, the base case is trivial:

$$\begin{aligned}
\mathsf{etr}_{N+1}\ p\ s &= \int_\omega \sum_{i<N+1} l^i \cdot r\ \omega_i\ \partial\mathcal{T}\ p\ s && \text{(Definition of } \mathsf{etr}_N\text{)}\\
&= \mathsf{etr}_N\ p\ s + l^N \cdot \int_\omega r\ \omega_N\ \partial\mathcal{T}\ p\ s && \text{(Split sum)}\\
&= \mathsf{etr}_N\ p\ s + l^N \cdot \int r\ \partial\mathsf{distr}\ (\mathcal{T}\ p\ s)\ (\lambda\omega.\ \omega_N) && \text{(Integration rule for distr)}\\
&= \mathsf{etr}_N\ p\ s + l^N \cdot \int r\ \partial\mathsf{P}_N\ p\ s && \text{(Theorem 3.38)}\\
&= \sum_{i<N+1} l^i \cdot \int r\ \partial\mathsf{P}_i\ p\ s && \text{(I.H.)}
\end{aligned}$$

The second fact follows from the definition of etr as the limit of etr_N. □

This fact allows us to restrict the search space for optimal policies to markovian policies. etr can be expressed in terms of P, but in Section 3.6, we proved that the state-action distribution P of any history-dependent policy can be achieved by a markovian policy. Thus markovian policies can achieve the same rewards as general policies.

Theorem 4.9 (Optimality of Markovian Policies)**.**

$$\begin{aligned}
\mathsf{etr}\ p\ s &= \mathsf{etr}\ (\mathsf{as\text{-}markovian}\ p\ s)_{MR}\ s\\
\mathsf{etr}^*\ s &= \bigsqcup p \in \Pi_{MR}.\ \mathsf{etr}\ p_{MR}\ s
\end{aligned}$$

Proof. The first fact follows from a combination of Theorem 4.8 and Theorem 3.33. Since markovian policies are a subset of history-dependent policies, the second fact is a consequence of the first. □

4.4 Bellman's Principle of Optimality

In the last section, we reduced the search space for optimal policies to markovian policies. A markovian policy is a sequence of decision rules, each of which can now be optimized

individually for an optimal reward. This observation will finally lead to the finding that the search space can be reduced to stationary policies.

As a first step, we show how to decompose the computation of etr into a computation for the initial epoch and another computation of etr for the tail. We obtain this decomposition as a consequence of the iteration rule for $\mathcal{T}$.

Theorem 4.10 (Decomposition of etr)**.**

$$\mathsf{etr}\ p\ s = \mathsf{r_{dec}}\ (p\ [])\ s + l \cdot \int_a \int \mathsf{etr}\ (\pi\text{-}\mathsf{Suc}\ p\ (s,a))\ \partial K\ (s,a)\ \partial p\ []\ s$$

$$\text{where } \mathsf{r_{dec}}\ d\ s = \int_a r\ (s,a)\ \partial d\ s.$$

The first addend involving $\mathsf{r_{dec}}$ is the expected reward for the initial decision rule. We have $\mathsf{bounded}\ (\mathsf{range}\ \mathsf{r_{dec}})$, i.e. $\mathsf{r_{dec}}$ is a bounded function. For markovian policies, Theorem 4.10 simplifies to

$$\mathsf{etr}\ p_{MR}\ s = \mathsf{r_{dec}}\ p_0\ s + l \cdot \int \mathsf{etr}\ (p_{\mathsf{Suc}})_{MR}\ \partial \mathsf{X}_1\ p\ s.$$

Thus to maximize etr, we can independently search for an optimal decision rule $p\ 0$ and an optimal tail of the policy. This fact motivates Bellman's *Principle of Optimality* [4]:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Suppose that p is an optimal policy, i.e. $\forall s.\ \mathsf{etr}\ p\ s = \mathsf{etr}^{\star}\ s$. If the initial decision rule was suboptimal, we could replace it with a better one, so p would not be optimal. If the tail of the policy was suboptimal, we could also replace it, again resulting in a contradiction. This also suggests that it suffices to search for a stationary policy, in which case the equation simplifies further. We proceed with a formalization of these observations.

$$\mathsf{etr}\ d_{SR}\ s = \mathsf{r_{dec}}\ d\ s + l \cdot \int \mathsf{etr}\ d_{SR}\ \partial \mathsf{K_{st}}\ d\ s$$

$$\text{where } \mathsf{K_{st}}\ d\ s = \mathsf{P_X}\ d\ s \gg\!= K \text{ is the distribution of successor states.}$$

The following analysis of etr draws on facts from the theory of real normed vectors. We show how to write etr in a compact vector notation. First we introduce a different view on expectations over bounded functions.

A function $P :: \beta \Rightarrow \gamma\ \mathsf{pmf}$ may also be seen as a stochastic matrix with real entries, rows β, and columns γ where all rows and columns add up to one. Let $i :: \beta$, $j :: \gamma$, then the entry $P_{ij} = \mathsf{pmf}\ (P\ i)\ j$. An example for such a matrix is K, which in this context is called the transition probability matrix of the MDP. For a bounded function $v :: \gamma \Rightarrow_b \mathsf{real}$, we can then define the matrix-vector product $P \cdot_E v$ as follows: the entry $b :: \beta$ of the resulting vector equals $\int v\ \partial P\ b$. We define $\_ \cdot_E \_$ as a bounded linear function.

Definition 4.11.

$$(\_ \cdot_E \_) :: (\beta \Rightarrow \gamma \text{ pmf}) \Rightarrow (\gamma \Rightarrow_b \text{real}) \Rightarrow_L (\beta \Rightarrow_b \text{real})$$
$$(P \cdot_E v)\ s = \int v\ \partial P\ s$$

For markovian policies, we define the n-step transition probability matrix $\mathcal{X}_n$ as the transformation X_n performs on bounded functions.

Definition 4.12. For $n :: \text{nat}$ we define

$$\mathcal{X}_n :: (\text{nat} \Rightarrow (\sigma, \alpha)\ \text{dec}) \Rightarrow (\sigma \Rightarrow_b \text{real}) \Rightarrow_L (\sigma \Rightarrow_b \text{real})$$
$$\mathcal{X}_n\ p\ v = \mathsf{X}_n\ p_{MR} \cdot_E v.$$

If we equip an agent that follows policy p from state s with a quantity $v\ s$, then $\mathcal{X}_n\ p\ v\ s'$ expresses the expected quantity an agent at state s' carries after n epochs. $\mathcal{X}_n$ satisfies the following alternative recursive definition:

Theorem 4.13.

$$\mathcal{X}_0\quad p\ v = v$$
$$\mathcal{X}_{n+1}\ p\ v = \mathsf{K}_{\mathsf{st}}\ p_0 \cdot_E \mathcal{X}_n\ p_{\mathsf{Suc}}\ v$$

Finally, we can write the expected total reward for markovian policies in a compact vector notation. Note that we sum a sequence of bounded linear functions, which is possible as they are elements of a real vector space.

Theorem 4.14.

$$\text{etr}\ p_{MR} = \sum_i l^i \cdot \mathcal{X}_i\ p\ (\mathsf{r}_{\mathsf{dec}}\ p_i)$$

Proof. A short calculation shows

$$\begin{aligned}
\text{etr}\ p_{MR}\ s &= \sum_i l^i \cdot \int r\ \partial \mathsf{P}_i\ p_{MR}\ s && \text{(Theorem 4.3)}\\
&= \sum_i l^i \cdot \int r\ \partial \mathsf{X}_i\ p_{MR}\ s \gg\!= \mathsf{P}_{\mathsf{X}}\ p_i && (\mathsf{P}_i \text{ for markovian policies})\\
&= \sum_i l^i \cdot \mathcal{X}_i\ p\ (\mathsf{r}_{\mathsf{dec}}\ p_i)\ s && \text{(Def. } \mathcal{X}_i \text{ and } \mathsf{r}_{\mathsf{dec}}\text{, integration of bind).}
\end{aligned}$$

Since all $l^i \cdot \mathcal{X}_i\ p\ (\mathsf{r}_{\mathsf{dec}}\ p_i)$ are linear transformations, we can pull s out of the infinite sum and obtain the desired statement. □

For stationary policies, etr may be simplified further. Let $\mathcal{K}_{\mathsf{st}}\ d\ v = \mathsf{K}_{\mathsf{st}}\ d \cdot_E v$ be the single-step transformation, such that $\mathcal{K}_{\mathsf{st}}\ p_0 = \mathcal{X}_1\ p$. In case of a stationary policy, the transformation $\mathcal{X}_n\ d_{SR}$ can be simplified using Theorem 4.13. We obtain that $\mathcal{X}_n\ d_{SR}$ is just the n-fold application of $\mathcal{K}_{\mathsf{st}}\ d$.

Theorem 4.15.

$$\mathsf{etr}\ d_{SR} = (\sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i)\ (\mathsf{r}_{\mathsf{dec}}\ d)$$

Since $\mathsf{r}_{\mathsf{dec}}\ d$ is the same at each epoch, we can pull it out of the infinite sum. Finally, we obtain a very simple decomposition of etr for stationary policies which relates back to the beginning of the section. The decomposition can be derived from Theorem 4.15 by splitting off the initial sum term.

Theorem 4.16.

$$\mathsf{etr}\ d_{SR} = \mathsf{r}_{\mathsf{dec}}\ d + l \cdot \mathcal{K}_{\mathsf{st}}\ d\ (\mathsf{etr}\ d_{SR})$$

We introduce the abbreviation L for the right-hand side. L is a bounded linear operator.

Definition 4.17.

$$\mathsf{L} :: (\sigma, \alpha)\ \mathsf{dec} \Rightarrow (\sigma \Rightarrow_b \mathsf{real}) \Rightarrow_L (\sigma \Rightarrow_b \mathsf{real})$$
$$\mathsf{L}\ d\ v = \mathsf{r}_{\mathsf{dec}}\ d + l \cdot \mathcal{K}_{\mathsf{st}}\ d\ v$$

From Theorem 4.16 it follows that etr d_{SR} is a fixed point of L. Moreover it is the unique fixed point. To establish this fact, we first prove an auxiliary lemma.

Lemma 4.18.

$$(\sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i) \circ_L (\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ d) = \mathsf{id}_L$$

Proof. The proof is by Theorem 2.2, we need to show that $\mathsf{norm}\ (l \cdot \mathcal{K}_{\mathsf{st}}\ d) < 1$. Any probability matrix such as $\mathcal{K}_{\mathsf{st}}\ d$ has a norm of 1. Because the norm scales with its argument and $l < 1$, we are done. □

Theorem 4.19 (Fixed point of L)**.**

$$\mathsf{L}\ d\ v = v \longleftrightarrow \mathsf{etr}\ d_{SR} = v$$

Proof. We consider both directions separately:
"$\Longrightarrow$" Unfolding L in the assumption and rearranging terms gives $(\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ d)\ v = \mathsf{r}_{\mathsf{dec}}\ d$. After composing from the left with $\sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i$ the facts 4.15 and 4.18 complete the proof.
"$\Longleftarrow$" Follows directly from Theorem 4.16.

□

4.5 Iterative Approximation of the Optimal Value

To find the optimal value of the MDP, we first optimize the rewards accrued in the initial decision epoch. We introduce the nonlinear operator $\mathcal{L}$, which takes as an argument an estimate v for the value of the MDP. It then maximizes $\mathsf{L}\ d\ v$ over all decision rules d. For all bounded functions v, $\mathcal{L}\ v$ is again a bounded function.

Definition 4.20.

$$\mathcal{L} :: (\sigma \Rightarrow_b \mathsf{real}) \Rightarrow (\sigma \Rightarrow_b \mathsf{real})$$
$$\mathcal{L}\ v\ s = \bigsqcup d \in D_R.\ \mathsf{L}\ d\ v\ s$$

We will show that $\mathcal{L}$ returns a better estimate for the value of the MDP than the estimate it receives as an argument. Moreover, we will see that etr^* is the unique fixed point of $\mathcal{L}$. First we show that if a fixed point of $\mathcal{L}$ exists, it is equal to etr^*.

Theorem 4.21 (Uniqueness of the fixed point of $\mathcal{L}$)**.**

assumes $v = \mathcal{L}\ v$
shows $v = \mathsf{etr}^*$

Proof. We treat the cases $\mathcal{L}\ v \leq v$ and $v \leq \mathcal{L}\ v$ separately:

- Assume $\mathcal{L}\ v \leq v$, we show $\mathsf{etr}^* \leq v$: It suffices to show $\mathsf{etr}\ p_{MR} \leq v$ for arbitrary $p \in \Pi_{MR}$. For all n, we show by induction the following fact about the total reward over a finite horizon: if we sum the expected reward upto epoch $n+1$ and the estimate of the discounted reward at epoch $n+1$ derived from v, the result is at most v:

  $$\mathsf{etr}_{n+1}\ p_{MR} + l^{n+1} \cdot \mathcal{X}_{n+1}\ p\ v \leq v$$

  Since the right addend tends towards 0 for large n, at the limit we obtain the desired statement.

- Now assume $v \leq \mathcal{L}\ v$, we need to prove $v \leq \mathsf{etr}^*$. It suffices to show $v \leq \mathsf{etr}^* + \epsilon/(1-l) \cdot_R 1_b$ for arbitrary $\epsilon > 0$. Here, 1_b is a constant function with value 1 everywhere. By assumption, we obtain a $d \in D_R$ where $v \leq \mathsf{L}\ d\ v + \epsilon \cdot 1_b$. Rearranging terms yields

  $$(\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ d)\ v \leq \mathsf{r}_{\mathsf{dec}}\ d + \epsilon \cdot 1_b.$$

Since $\sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i$ is a monotone function (proof omitted), we can calculate

$$
\begin{aligned}
v &= \sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i\ ((\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ d)\ v) && \text{(Lemma 4.18)} \\
&\leq \sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i\ (\mathsf{r}_{\mathsf{dec}}\ d + \epsilon \cdot 1_b) && \text{(monotonicity)} \\
&= \mathsf{etr}\ d_{SR} + \epsilon \cdot \sum_i (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^i\ 1_b && \text{(Theorem 4.15)} \\
&= \mathsf{etr}\ d_{SR} + \epsilon \cdot \sum_i l^i \cdot 1_b && (\mathsf{K}_{\mathsf{st}} \text{ is a probability matrix}) \\
&= \mathsf{etr}\ d_{SR} + \epsilon/(1-l) \cdot 1_b && \text{(limit of geometric series).}
\end{aligned}
$$

As $\mathsf{etr}\ d_{SR} \leq \mathsf{etr}^*$, this completes the proof of the second case.

The whole theorem is a consequence of both proofs above. □

So far, we have only proved uniqueness but not existence of a fixed point. To show the existence of a fixed point, we prove that $\mathcal{L}$ is a contraction mapping with factor l.

Theorem 4.22 ($\mathcal{L}$ is a contraction mapping)**.**

$$\mathsf{dist}\ (\mathcal{L}\ v)\ (\mathcal{L}\ u) \leq l \cdot \mathsf{dist}\ v\ u$$

Proof. Let $s :: \sigma$ be an arbitrary state. Without loss of generality it suffices to consider the case $\mathcal{L}\ u\ s \leq \mathcal{L}\ v\ s$ and show $\mathsf{dist}\ (\mathcal{L}\ v\ s)\ (\mathcal{L}\ u\ s) \leq l \cdot \mathsf{dist}\ v\ u$. We calculate

$$
\begin{aligned}
&\mathsf{dist}\ (\mathcal{L}\ v\ s)\ (\mathcal{L}\ u\ s) \\
&\leq \mathcal{L}\ v\ s - \mathcal{L}\ u\ s && \text{(Definition of dist)} \\
&= (\bigsqcup d \in D_R.\ \mathsf{L}\ d\ v\ s) - (\bigsqcup d \in D_R.\ \mathsf{L}\ d\ u\ s) && \text{(Definition of } \mathcal{L}) \\
&\leq \bigsqcup d \in D_R.\ \mathsf{L}\ d\ v\ s - \mathsf{L}\ d\ u\ s && \text{(Property of subtraction and } \textstyle\bigsqcup) \\
&= \bigsqcup d \in D_R.\ l \cdot \mathcal{K}_{\mathsf{st}}\ d\ v\ s - l \cdot \mathcal{K}_{\mathsf{st}}\ d\ u\ s && \text{(Definition of L)} \\
&= l \cdot \bigsqcup d \in D_R.\ \mathcal{K}_{\mathsf{st}}\ d\ (v-u)\ s && (\text{Linearity of } \mathsf{K}_{\mathsf{st}}) \\
&\leq l \cdot \bigsqcup d \in D_R.\ \mathsf{norm}\ (\mathcal{K}_{\mathsf{st}}\ d\ (v-u)) && \text{(norm of bounded functions)} \\
&\leq l \cdot \mathsf{norm}\ (v-u) && (\mathsf{K}_{\mathsf{st}}\ d \text{ is a probability matrix}) \\
&= l \cdot \mathsf{dist}\ v\ u.
\end{aligned}
$$

□

For contraction mappings on complete spaces (which the space of bounded functions into the reals is), the Banach fixed-point theorem tells us that they have a unique fixed point (Theorem 2.1). Thus etr* is the unique fixed point of $\mathcal{L}$.

Theorem 4.23. (Fixed point of $\mathcal{L}$)

$$\mathcal{L}\ v = v \longleftrightarrow v = \mathsf{etr}^*$$

Moreover, the Banach fixed-point theorem tells us how to compute the fixed point: starting with any bounded function, repeated application of the contraction leads to the fixed point.

Theorem 4.24.

$$(\lambda n.\ \mathcal{L}^n\ v) \longrightarrow \mathsf{etr}^*$$

Repeated application of $\mathcal{L}$ gives ever better approximations of etr^*. The calculation of $\mathcal{L}$ can still be simplified. In the definition of $\mathcal{L}$, we take the supremum over all decision rules. However, it suffices to consider only the set of deterministic decision rules D_D. Intuitively, the next fact tells us that since there is an optimal action for each state, it is wasteful to give any weight to suboptimal actions.

Theorem 4.25.

$$\mathcal{L}\ v\ s = \bigsqcup d \in D_D.\ \mathsf{L}\ d_D\ v\ s$$

Proof. The direction "$\geq$" is clear. For the other direction, we use a fact from probability theory: let $p :: \beta\ \mathsf{pmf}$. Let $f :: \beta \Rightarrow_b \mathsf{real}$ and $\mathsf{set\text{-}pmf}\ p \subseteq X$. Then

$$\int f\ \partial p \leq \bigsqcup \{f\ x \mid x \in X\}.$$

If we unfold the definition of $\mathcal{L}$, after some simplifications we can apply the above fact. □

4.6 Existence of Optimal Policies

So far we studied how to determine the optimal value etr^* of an MDP. As a next step we establish a condition under which this value is attained by a concrete policy, i.e. under which assumption an optimal policy exists.

We first define what it means for a decision rule to be optimal for a given estimate v of the value of the MDP. A v-improving decision rule is a decision rule d for which $\mathsf{L}\ d\ v = \mathcal{L}\ v$.

Definition 4.26 (Improving decision rule)**.** We call a decision rule $d \in D_R$ v-improving if it maximizes L:

$$\mathsf{improving}\ v\ d \longleftrightarrow d \in D_R \wedge (\forall d' \in D_R.\ \forall s.\ \mathsf{L}\ d'\ v\ s \leq \mathsf{L}\ d\ v\ s)$$

Decision rules which are etr^*-improving are called *conserving*.

Definition 4.27 (Conserving decision rule)**.**

$$\mathsf{conserving}\ d = \mathsf{improving}\ \mathsf{etr}^*\ d$$

Conserving decision rules are optimal. Thus the existence of a conserving deterministic decision rule is enough to ensure existence of an optimal (deterministic stationary) policy.

Theorem 4.28 (Optimality of conserving decision rules)**.**

assumes conserving d_D
shows etr $d_{SD} =$ etr*

Proof. If d is conserving, then $\mathsf{L}\ d$ etr* $= \mathcal{L}$ etr* $=$ etr*. Hence etr* is a fixed point of $\mathsf{L}\ d$. However, we already established in Theorem 4.19 that etr d_{SD} is the unique fixed point of $\mathsf{L}\ d$. So etr $d_{SD} =$ etr* and therefore d is optimal. □

Existence of a conserving decision rule is a sufficient condition for the existence of an optimal policy. We can also give a weaker sufficient condition: if the supremum in $\mathcal{L}\ v$ is attained for all bounded vectors v then an optimal policy exists. This condition is equivalent to the existence of a v-improving decision rule for all bounded v.

Finally we can show that if an optimal policy exists, an optimal deterministic stationary policy always exists. The corresponding theorem by Puterman [16, Theorem 6.2.9] claims to show the same result, but in fact the proof only shows existence of a randomized stationary policy. We repair this small inaccuracy with an additional proof similar to Theorem 4.25. It shows that a randomized stationary policy can always be replaced by a deterministic decision rule with identical reward.

Theorem 4.29 (Optimality of deterministic decision rules)**.**

assumes $p \in \Pi_{HR}$ **and** etr $p =$ etr*
shows $\exists d \in D_D.$ etr $d_{SD} =$ etr*

Proof. We have for all $s :: \sigma$

$$\begin{aligned}
\text{etr*}\ s &= \text{etr}\ p\ s \\
&= \text{etr}\ (\text{as-markovian}\ p\ s)\ s \\
&= \mathsf{L}\ (\text{as-markovian}\ p\ s)_0\ (\text{etr}\ (\lambda n.\ \text{as-markovian}\ p\ s\ (n+1))_{MR})\ s \\
&= \mathsf{L}\ (p\ [])\ (\text{etr}\ (\lambda n.\ \text{as-markovian}\ p\ s\ (n+1))_{MR})\ s \\
&\leq \mathsf{L}\ (p\ [])\ \text{etr*}\ s \\
&\leq \mathcal{L}\ \text{etr*}\ s \\
&= \text{etr*}\ s.
\end{aligned}$$

So $\mathsf{L}\ (p\ [])$ etr* $s =$ etr* s, $p\ []$ is conserving and thus optimal. An argument similar to Theorem 4.25 shows that there exists an equivalent deterministic decision rule. □

5 Value Iteration

The *value iteration* algorithm computes an approximation of the optimal value of the MDP by fixed-point iteration. From this computation, it then derives a policy that attains the approximation. Value iteration uses the computation rule given by the Banach fixed-point theorem to compute the fixed point of $\mathcal{L}$, which equals the value of the MDP. The first phase of the algorithm computes the value of the MDP. It proceeds as follows: given an initial bounded estimate v_0 for the value of the MDP and an error $\epsilon > 0$, compute $v_{n+1} = \mathcal{L}\ v_n$ until two successive iterates are less than $\frac{\epsilon(1-l)}{2l}$ apart.

Definition 5.1 (Value Iteration)**.**

$$\begin{aligned}
&\mathsf{value\text{-}iteration} :: \mathsf{real} \Rightarrow (\sigma \Rightarrow_b \mathsf{real}) \Rightarrow (\sigma \Rightarrow_b \mathsf{real})\\
&\mathsf{value\text{-}iteration}\ \epsilon\ v = \textbf{if}\ 2l \cdot \mathsf{dist}\ v\ (\mathcal{L}\ v) < \epsilon(1-l) \vee \epsilon \leq 0\\
&\qquad\qquad\qquad\qquad\ \textbf{then}\ \mathcal{L}\ v\\
&\qquad\qquad\qquad\qquad\ \textbf{else}\ \ \mathsf{value\text{-}iteration}\ \epsilon\ (\mathcal{L}\ v)
\end{aligned}$$

In the formal definition, we multiply the termination condition on both sides by $2l$ to avoid division by 0 in the case $l = 0$. An argument for the termination of the algorithm follows. We assume $l > 0$, the case $l = 0$ is trivial. If $\epsilon \leq 0$, we abort the iteration immediately to ensure termination. If $\epsilon > 0$, we prove termination of the algorithm by providing a measure that decreases in every iteration. Here, a measure measures the size of the arguments to a function, not to be confused with the same term in a measure-theoretic context.

$$\lambda(\epsilon, v).\ \textbf{if}\ v = \mathsf{etr}^*\ \textbf{then}\ 0\ \textbf{else}\ \lceil \mathsf{log}\ (1/l)\ (\mathsf{dist}\ v\ \mathsf{etr}^*) - \mathsf{log}\ (1/l)\ (\epsilon(1-l)/8l) \rceil$$

The distance between the iterates and etr* decreases by a factor of at least l in each iteration, so the measure decreases in each iteration. Due to the termination condition, we can show that the measure is always nonnegative. Such a measure, which is decreasing and bounded from below ensures termination.

We now prove the correctness of the first phase of the algorithm. Specifically, we show that value-iteration computes an approximation of the optimal value that deviates less than $\epsilon/2$ from etr*. To prove correctness, we require the following result on contraction mappings.

Theorem 5.2. Let β be a complete-space, let $C :: \beta \Rightarrow \beta$ be a contraction mapping with factor c. The distance from an arbitrary point x to the fixed point of C is bounded in terms of the distance that C moves x.

assumes $\forall v\ u.\ \mathsf{dist}\ (C\ v)\ (C\ u) \leq c \cdot \mathsf{dist}\ v\ u$ **and** $0 \leq c < 1$
shows $\mathsf{dist}\ v\ (\mathsf{fix}\ C) \leq \mathsf{dist}\ v\ (C\ v)/(1-c)$
where $\mathsf{fix}\ C = (\iota v.\ C\ v = v)$

Instantiating the theorem with $\mathcal{L}$ provides us with a bound for the error of an approximation for etr*: $\mathsf{dist}\ v\ \mathsf{etr}^* \leq \mathsf{dist}\ v\ (\mathcal{L}\ v)/(1-l)$. It then directly follows from the termination condition that value-iteration has an error of less than $\epsilon/2$.

Theorem 5.3 (ϵ-optimality of value-iteration)**.**

assumes $0 < \epsilon$
shows $\mathsf{dist}\ (\mathsf{value\text{-}iteration}\ \epsilon\ v)\ \mathsf{etr}^* < \epsilon/2$

After the value iteration terminates, one can easily obtain an ϵ-optimal stationary deterministic policy. We perform another value iteration step, but this time we record which decision rule maximizes $\mathcal{L}$. This is equivalent to considering each state individually and choosing the enabled action which maximizes

$$\mathsf{L}_{\mathsf{act}}\ a\ v\ s = r\ (s,a) + l \cdot \int v\ \partial K\ (s,a).$$

We call this action act*, where $\mathsf{act}^*\ v\ s = \mathsf{arg\text{-}max}\ a \in A\ s.\ \mathsf{L}_{\mathsf{act}}\ a\ v\ s$. We introduce the abbreviation $\mathsf{arg\text{-}max}\ x \in X.\ f\ x = \epsilon x.\ x \in X \wedge (\forall y \in X.\ f\ y \leq f\ x)$. Ultimately, we compose act* and value-iteration in the definition of vi-policy. vi-policy takes as arguments an error $\epsilon > 0$ and an initial value estimate, it returns a deterministic decision rule.

Definition 5.4.

$\mathsf{vi\text{-}policy} :: \mathsf{real} \Rightarrow (\sigma \Rightarrow_b \mathsf{real}) \Rightarrow (\sigma, \alpha)\ \mathsf{dec}$
$\mathsf{vi\text{-}policy}\ \epsilon\ v\ s = \mathsf{act}^*\ (\mathsf{value\text{-}iteration}\ \epsilon\ v)\ s$

Note that vi-policy is not well-defined in general, as the attainment of the supremum in $\mathcal{L}$ is not guaranteed in general. However, to show that vi-policy finds an ϵ-optimal policy, we require that the supremum in $\mathcal{L}$ is attained for all states and value functions. In particular, this assumption is fulfilled for finite action spaces. For the remainder of this chapter we make the following (equivalent) assumption:

$$\forall v\ s.\ \exists a.\ a = \mathsf{act}^*\ v\ s$$

Due to the triangle inequality, the error of the resulting policy $\mathsf{dist}\ (\mathsf{etr}\ (\mathsf{vi\text{-}policy}\ \epsilon\ v)_{SD})\ \mathsf{etr}^*$ is bounded by $\mathsf{dist}\ (\mathsf{etr}\ (\mathsf{vi\text{-}policy}\ \epsilon\ v)_{SD})\ (\mathsf{value\text{-}iteration}\ \epsilon\ v) + \mathsf{dist}\ (\mathsf{value\text{-}iteration}\ \epsilon\ v)\ \mathsf{etr}^*$. We show that both addends are less than $\epsilon/2$.

Theorem 5.5 (ϵ-optimality of vi-policy)**.**

assumes $0 < \epsilon$ **and** $\forall v\ s.\ \exists a.\ a = \mathsf{act}^*\ v\ s$
shows $\mathsf{dist}\ (\mathsf{etr}\ (\text{vi-policy}\ \epsilon\ v)_{SD})\ \mathsf{etr}^* < \epsilon$

Proof. Let u be arbitrary such that $2l \cdot \mathsf{dist}\ u\ (\mathcal{L}\ u) < \epsilon(1-l)$. We abbreviate $d = (\mathsf{act}^*\ (\mathcal{L}\ u))$ and calculate

$$
\begin{aligned}
& \mathsf{dist}\ (\mathsf{etr}\ d_{SD})\ (\mathcal{L}\ u) \\
&= \mathsf{dist}\ (\mathsf{L}\ d\ (\mathsf{etr}\ d_{SD}))\ (\mathcal{L}\ u) && \text{(Theorem 4.19)} \\
&\leq \mathsf{dist}\ (\mathsf{L}\ d\ (\mathsf{etr}\ d_{SD}))\ (\mathcal{L}\ (\mathcal{L}\ u)) + \mathsf{dist}\ (\mathcal{L}\ (\mathcal{L}\ u))\ (\mathcal{L}\ u) && \text{(Triangle ineq.)} \\
&\leq \mathsf{dist}\ (\mathsf{L}\ d\ (\mathsf{etr}\ d_{SD}))\ (\mathcal{L}\ (\mathcal{L}\ u)) + l \cdot \mathsf{dist}\ (\mathcal{L}\ u)\ u && \text{(Theorem 4.22)} \\
&= \mathsf{dist}\ (\mathsf{L}\ d\ (\mathsf{etr}\ d_{SD}))\ (\mathsf{L}\ d\ (\mathcal{L}\ u)) + l \cdot \mathsf{dist}\ (\mathcal{L}\ u)\ u && \text{(Definition of } d\text{)} \\
&\leq l \cdot \mathsf{dist}\ (\mathsf{etr}\ d_{SD})\ (\mathcal{L}\ u) + l \cdot \mathsf{dist}\ (\mathcal{L}\ u)\ u && \text{(Contraction mapping L)} \\
&< l \cdot \mathsf{dist}\ (\mathsf{etr}\ d_{SD})\ (\mathcal{L}\ u) + \epsilon(1-l)/2 && \text{(assumption).}
\end{aligned}
$$

We omit the proof that L is a contraction mapping. Rearranging terms and division by $(1-l)$ results in the inequality

$$\mathsf{dist}\ (\mathsf{etr}\ (\mathsf{act}^*\ (\mathcal{L}\ u))_{SD})\ (\mathcal{L}\ u) < \epsilon/2.$$

From the definition of vi-policy and value-iteration we obtain

$$\mathsf{dist}\ (\mathsf{etr}\ (\text{vi-policy}\ \epsilon\ v)_{SD})\ (\text{value-iteration}\ \epsilon\ v) < \epsilon/2.$$

Together with Theorem 5.3, ϵ-optimality of vi-policy follows from the triangle inequality. □

Code Generation If we restrict attention to MDPs with finite state and action spaces, our definitions for the value iteration algorithm can be turned into executable code. To extract executable code from the mathematical definitions, we use the code generation framework available in Isabelle/HOL [9]. It allows us to specify rewriting rules that refine our definitions to executable code.

We only generate code for finite MDPs, where integrals reduce to finite sums and the supremum in $\mathcal{L}$ can be found in finite time. In particular, we assume that both types σ and ω are instances of mod-type. This type class also gives a bijection to a subset $\{0.. < n\}$ of the natural numbers, hence the type can be used as an index type for vectors. This will be useful as the executable code stores decision rules and functions over finite types as vectors. Moreover, we obtain an order on the elements. This allows us to make act* executable, we always choose the least action that maximizes $\mathsf{L}_{\mathsf{act}}$.

The code generation for value iteration proceeds as follows: using the **typedef** command, define a new type mdp of tuples (A, K, r, l) that satisfy all assumptions of the locale MDP-reward. The function to-MDP lifts a tuple to the type mdp. This allows us to

state all theorems previously discussed outside of a locale context. An extra argument of type (σ, α) mdp needs to be passed to each executable function definition outside of the locale. Using a type definition has particular relevance for generating code for recursive functions which only terminate due to locale assumptions. As we lift these assumptions to the type-level, we can prove termination outside the locale without any assumptions on the term-level. For the executable version of the value iteration algorithm, we end up with the following theorem:

assumes $0 < \epsilon$ **and** MDP-reward $A\ K\ r\ l$
let $m =$ to-MDP $A\ K\ r\ l$
shows dist $(\text{etr}\ m\ (\text{vi-policy}\ m\ \epsilon\ v)_{SD})\ (\text{etr*}\ m) < \epsilon$

When finite MDPs are considered, the predicate MDP-reward is executable as well. Thus it is possible to derive theorems about the generated code that hold without any assumptions on A, K, r, and l.

6 Policy Iteration

The *policy iteration* algorithm provides an alternative solution to the problem of determining optimal policies under the expected total discounted reward criterion. Similar to value iteration, it uses an iterative fixed-point computation to find an optimal deterministic policy. Value iteration iteratively improves its estimates of the value of the MDP and only afterwards derives a policy. Policy Iteration however directly searches the space of deterministic decision rules for an optimal policy. It continuously improves on an initial guess for an optimal decision rule and terminates as soon as the decision rule cannot be improved further.

Execution of the policy iteration algorithm can be subdivided into two alternating steps: *policy evaluation* and *policy improvement*. Policy evaluation calculates the value of the current decision rule d, i.e. $\mathsf{etr}\ d_{SD}$. During the improvement phase, we update the action selection for each state s individually. We choose the action with the maximum value for $\mathsf{L_{act}}\ a\ (\mathsf{etr}\ d)\ s$, while we prefer to keep the old action selection in case of ties. The definition policy-step describes a single execution of both policy evaluation and policy improvement:

Definition 6.1.

$$
\begin{aligned}
&\mathsf{policy\text{-}step} :: (\sigma \Rightarrow \alpha) \Rightarrow (\sigma \Rightarrow \alpha)\\
&\mathsf{policy\text{-}step}\ d\ s = \textbf{if}\ \mathsf{L_{act}}\ (d\ s)\ s = \mathsf{L_{act}}\ (\mathsf{act^*}(\mathsf{etr}\ d_{SD})\ s)\ s\\
&\qquad\qquad\qquad\quad \textbf{then}\ d\ s\\
&\qquad\qquad\qquad\quad \textbf{else}\ \ \mathsf{act^*}\ (\mathsf{etr}\ d_{SD})\ s
\end{aligned}
$$

To prove the desired properties about policy-step, we have to make sure the supremum in $\mathcal{L}$ is always attained. For the rest of the chapter, we assume that

$$\forall v\ s.\ \exists a.\ a = \mathsf{act^*}\ v\ s.$$

We can now define the policy iteration algorithm as a simple fixed-point iteration of policy-step. In case an invalid decision rule is passed, we immediately abort the algorithm.

Definition 6.2 (Policy Iteration)**.**

$$\begin{aligned}
&\mathsf{policy\text{-}iteration} :: (\sigma \Rightarrow \alpha) \Rightarrow (\sigma \Rightarrow \alpha)\\
&\mathsf{policy\text{-}iteration}\ d =\\
&\quad \textbf{let}\ d' = \mathsf{policy\text{-}step}\ d\ \textbf{in}\\
&\qquad \textbf{if}\ d = d' \vee \neg \mathsf{is\text{-}dec\text{-}det}\ d\\
&\qquad \textbf{then}\ d\\
&\qquad \textbf{else}\ \ \mathsf{policy\text{-}iteration}\ d'
\end{aligned}$$

It is easy to see that if policy iteration terminates, i.e. if $d = \mathsf{policy\text{-}step}\ d$, then d is an optimal decision rule.

Theorem 6.3.

$$\begin{aligned}
&\textbf{assumes} \quad d \in D_D \quad \textbf{and} \quad d = \mathsf{policy\text{-}step}\ d\\
&\quad \textbf{shows} \quad \mathsf{etr}\ d_{SD} = \mathsf{etr}^*
\end{aligned}$$

Proof. For all s, we have

$$\begin{aligned}
\mathsf{etr}\ d_{SD}\ s &= \mathsf{L}\ d_D\ (\mathsf{etr}\ d_{SD})\ s && \text{(Theorem 4.19)}\\
&= \mathsf{L}\ (\mathsf{policy\text{-}step}\ d)_D\ (\mathsf{etr}\ d_{SD})\ s && \text{(By assumption)}\\
&= \mathsf{L}_{\mathsf{act}}\ (\mathsf{policy\text{-}step}\ d\ s)\ (\mathsf{etr}\ d_{SD})\ s\\
&= \bigsqcup a \in A\ s.\ \mathsf{L}_{\mathsf{act}}\ (d\ s)\ (\mathsf{etr}\ d_{SD})\ s && \text{(Def. policy-step, act*)}\\
&= \mathcal{L}\ (\mathsf{etr}\ d_{SD})\ s
\end{aligned}$$

So $\mathsf{etr}\ d_{SD}$ is a fixed point of $\mathcal{L}$ and thus $\mathsf{etr}\ d_{SD} = \mathsf{etr}^*$. □

Next, we will see that for finite state and action spaces, policy iteration is guaranteed to terminate. The following fact is crucial for termination: the sequence of policies generated by policy iteration has monotonically increasing expected reward.

Theorem 6.4. Let $d \in D_D$, then

$$\mathsf{etr}\ d_{SD} \leq \mathsf{etr}\ (\mathsf{policy\text{-}step}\ d)_{SD}.$$

Proof. From Theorem 4.19 and the definition of policy-step, we observe that

$$\mathsf{etr}\ d_{SD} = \mathsf{L}\ d_D(\mathsf{etr}\ d_{SD}) \leq \mathsf{L}\ (\mathsf{policy\text{-}step}\ d)_D\ (\mathsf{etr}\ d_{SD}).$$

Unfolding the definition of L and rewriting yields

$$(\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ (\mathsf{policy\text{-}step}\ d)_D)\ (\mathsf{etr}\ d_{SD}) \leq \mathsf{r}_{\mathsf{b}}\ (\mathsf{policy\text{-}step}\ d)_D.$$

We abbreviate $\mathcal{K}_{\mathsf{st}}\ (\mathsf{policy\text{-}step}\ d)_D$ as $\mathcal{K}'_{\mathsf{st}}$. Since the linear function $\sum_i\ (l \cdot \mathcal{K}'_{\mathsf{st}})^i$ is monotone (without proof) we get

$$(\sum_i\ (l \cdot \mathcal{K}'_{\mathsf{st}})^i)\ ((\mathsf{id}_L - l \cdot \mathcal{K}'_{\mathsf{st}})\ (\mathsf{etr}\ d_{SD})) \leq \mathsf{etr}\ (\mathsf{policy\text{-}step}\ d)_{SD}.$$

Here we have simplified the right-hand side using Theorem 4.15. The transformations on the left-hand side cancel each other by Lemma 4.18, so we are done. □

We only show termination for the case that the state and action spaces are finite types. In this context, the set of all deterministic decision rules is finite (finite D_D). Thus for $d \in D_D$, there is also only a finite number of values that etr d_{SD} can take on. The value of the policy is strictly increasing for each recursive call in policy-iteration, as policy-step only alters the policy if an improvement can be made. A combination of these observations shows that the recursion in policy-iteration is only of finite depth and the algorithm terminates for finite MDPs. The termination proof gives us access to the simplification lemmas associated with the policy-iteration definition. Thus we can show that the algorithm finds an optimal policy as a direct consequence of Theorem 6.3.

Theorem 6.5 (Correctness of Policy Iteration for finite MDPs)**.** Assume σ :: finite, α :: finite. Fix $d \in D_D$, then

$$\mathsf{etr}\ (\mathsf{policy\text{-}iteration}\ d)_{SD} = \mathsf{etr}^{\star}.$$

Code Generation Code generation for the policy iteration algorithm proceeds similar to value iteration. We again generate code only for finite MDPs using the type mdp, which ensures the MDP axioms. The main challenge for policy iteration is the policy evaluation phase, where seemingly an infinite sum has to be computed:

$$\mathsf{etr}\ d_{SR} = (\sum_t (l \cdot \mathcal{K}_{\mathsf{st}}\ d)^t)\ \mathsf{r}_{\mathsf{dec}}$$

However, by Lemma 4.18, the inverse function $(\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ d)$ of the infinite sum can be computed in finite time. For finite types, bounded linear transformations correspond to matrices, inverse transformations can be determined by matrix inversion. Hence for code generation, we treat $\mathsf{id}_L - l \cdot \mathcal{K}_{\mathsf{st}}\ d$ as a matrix. We then use the verified and executable Gauss-Jordan algorithm by Aransay et al. [1] to perform matrix inversion. Finally, a simple matrix-vector multiplication with $\mathsf{r}_{\mathsf{dec}}$ yields the expected total discounted reward.

7 Evaluation

After the theoretical discussion and verification of the algorithms, we now illustrate their functionality in action. Therefore, we execute the algorithms on a concrete MDP and discuss their performance. Finally, we evaluate our formalization with regard to the design decisions made.

7.1 Gridworld Example

The so-called Gridworld is a standard example to exemplify MDPs and their solution methods. It well-suited to visually display the gradually improving policies discovered by the algorithms.

Scenario A robot is placed on a four-by-three grid, surrounded by walls. The scenario is depicted in Figure 7.1. The cells are named from $(0,0)$ in the bottom left to $(3,2)$ in the top right. Cell $(1,1)$ is an obstacle the robot cannot step onto. At discrete time points, the robot has choose the direction of its next move: up, right, down, or left. The direction picked is subject to interference: with a chance of ten percent each, the action the robot chooses is altered by 90 degrees clockwise or counterclockwise. Then the robot moves in this direction by a single square. If the movement is impossible due to an obstacle or a wall, the robot does not move. The objective of the robot is to get to the cell in the top right, which generates a reward of 1. However, care is required as stepping on cell $(3,1)$ leads to a negative reward of -1. After either cell $(3,1)$ or $(3,2)$ is reached, the robot is removed from the scenario and remains forever in a special trap state.

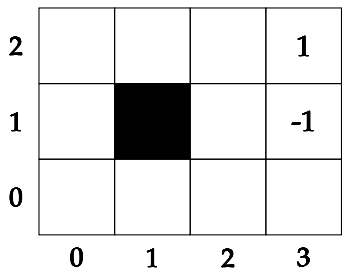


Figure 7.1: The Gridworld scenario

Gridworld in Isabelle/HOL We model the scenario described above as an MDP in Isabelle/HOL. The action space is given by the type action, whose members are the four directions the robot may move in. All actions are enabled everywhere ($\forall s.\ A\ s = \mathsf{UNIV}$). The state of the robot is either a position on the grid or special trap state, after the robot has reached either $(3,2)$ or $(3,1)$. We represent the coordinates of the cells as a combination of the types 4 and 3, which correspond to the sets $\{0,1,2,3\}$ and $\{0,1,2\}$ respectively.

$$\begin{aligned}&\textbf{datatype}\ \mathsf{action} = \mathsf{Up} \mid \mathsf{Right} \mid \mathsf{Down} \mid \mathsf{Left}\\ &\textbf{datatype}\ \mathsf{state}\ \ = \mathsf{Pos}\ 4\ 3 \mid \mathsf{Trap}\end{aligned}$$

The reward function is zero for all states except for $(3,2)$ and $(3,1)$, where a reward of $+1$ or -1 is accrued. The definitions are simplified slightly compared to the formalization, as pattern matching this way on the types 4 and 3 is not possible in Isabelle/HOL.

Definition 7.1 (Gridworld reward function)**.**

$$\begin{aligned}r\ ((\mathsf{Pos}\ 3\ 2), \_) &= +1\\ r\ ((\mathsf{Pos}\ 3\ 1), \_) &= -1\\ r\ \_ &= 0\end{aligned}$$

The kernel K of the MDP describes the movement of the robot. From the cells $(3,2)$, $(3,1)$ and Trap, it moves to Trap deterministically. In all other situations $(\mathsf{Pos}\ x\ y, a)$ we first add noise to the selected direction a using the function disturb-dir (we omit its definition). Afterward, an update of the position occurs according to this direction. If the robot would move out of bounds or onto $(1,1)$, its position remains unchanged.

Definition 7.2 (Gridworld MDP kernel)**.**

$$\begin{aligned}K\ (\mathsf{Trap}, \_) &= \mathsf{return\text{-}pmf}\ \mathsf{Trap}\\ K\ (\mathsf{Pos}\ 3\ 2, \_) &= \mathsf{return\text{-}pmf}\ \mathsf{Trap}\\ K\ (\mathsf{Pos}\ 3\ 1, \_) &= \mathsf{return\text{-}pmf}\ \mathsf{Trap}\\ K\ (\mathsf{Pos}\ x\ y, a) &= \textbf{do}\{\ a \leftarrow \mathsf{disturb\text{-}dir}\ a;\\ &\qquad \textbf{let}\ (x', y') = (\textbf{case}\ a\ \textbf{of}\\ &\qquad\quad \mathsf{Up}\ \ \Rightarrow (x, \textbf{if}\ y = 2\ \textbf{then}\ y\ \textbf{else}\ y + 1)\ \mid\\ &\qquad\quad \mathsf{Down} \Rightarrow (x, \textbf{if}\ y = 0\ \textbf{then}\ y\ \textbf{else}\ y - 1)\ \mid\\ &\qquad\quad \mathsf{Right} \Rightarrow (\textbf{if}\ x = 3\ \textbf{then}\ x\ \textbf{else}\ x + 1, y)\ \mid\\ &\qquad\quad \mathsf{Left}\ \ \Rightarrow (\textbf{if}\ x = 0\ \textbf{then}\ x\ \textbf{else}\ x - 1, y));\\ &\qquad \textbf{let}\ (x', y') = \textbf{if}\ (x', y') = (1,1)\ \textbf{then}\ (x, y)\ \textbf{else}\ (x', y');\\ &\qquad \mathsf{return\text{-}pmf}\ (\mathsf{Pos}\ x'\ y')\ \}\end{aligned}$$

To incentivize the robot to reach the target quickly, we set the discounting factor $l = 0.9$. We execute both value iteration and policy iteration to find an optimal policy for the scenario.

Policy Iteration First, we discuss the performance of the policy iteration algorithm on the Gridworld MDP. We initialize the algorithm with a decision rule that chooses the direction `Up` for each grid cell. The algorithm terminates with an optimal policy after only two iterations.

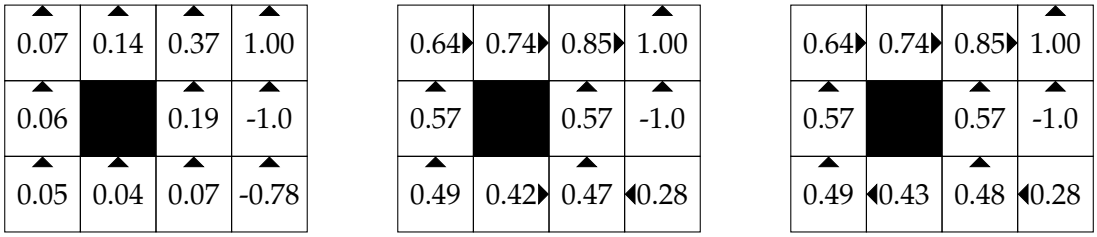
We visualize the current decision rule at each iteration on the cell grid in Figure 7.2. The small triangles in each cell denote the chosen action (direction) of the current policy. The number in each cell is the expected total discounted reward for the cell before the policy improvement phase has taken place. It is rounded to two decimal places.

Value Iteration As a first estimate for the value of the MDP we use the zero vector. We utilize it to initialize the value iteration algorithm on the Gridworld example. Visualization of the iterates is similar to policy iteration. For each iteration, we show the current approximation of the value function in the corresponding grid cell. Small triangles again indicate the optimal action choice for the estimate. The evolution of the iterates is shown in Figure 7.3. Our visualization shows how the value approximations propagate through the grid step by step. It takes six iterations until every state is assigned a nonzero value. The optimal policy is only found after ten iterations, after 20 iterations the values are correct to two decimal places.

Comparison On our laptop hardware it takes about 30 ms to perform ten iterations of value iteration, while policy iteration terminates in 22 ms. These numbers only serve as a demonstration that the generated code exhibits reasonable performance on small inputs. Due to a lack of optimization and a more extensive benchmarking suite, we cannot draw detailed conclusions on how the algorithms fare in comparison.

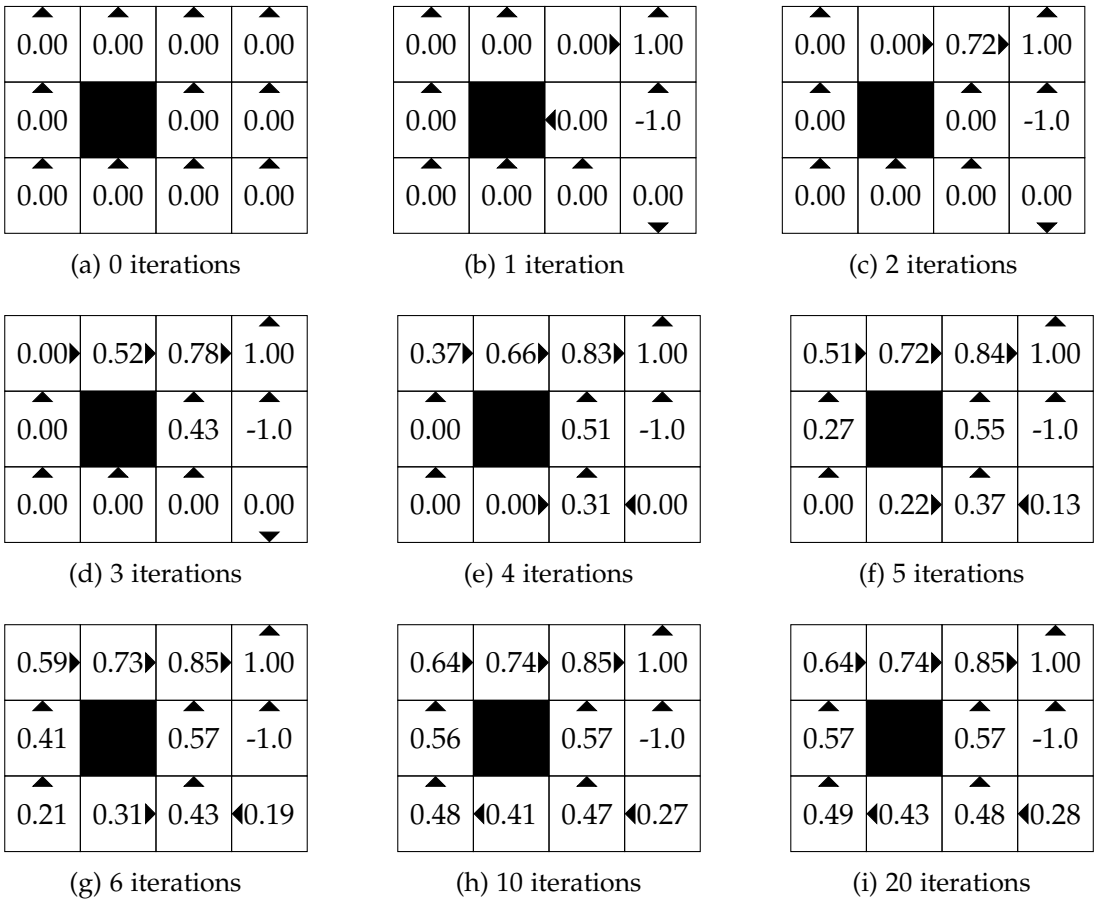
However, we do observe the typical tradeoff between value iteration and policy iteration: policy iteration converges in fewer iterations, but a single iteration is more computationally expensive. This is mostly due to the expensive policy evaluation, which in our case is done using a matrix inversion operation. These observations are also backed up by an analysis of the convergence rates of the algorithms. In many scenarios, policy iteration converges quadratically to the optimal value while value iteration converges only linearly at rate l [16]. We did not formally verify this analysis.

The policy iteration algorithm always terminates with an optimal policy. The same cannot be said for value iteration. We only obtain a guarantee of ϵ-optimality. On the other hand, after each value iteration step the algorithm may be aborted and an estimate on its optimality can be made. Similar but weaker guarantees can be obtained for the policy iteration algorithm as well, however only under certain assumptions on the concrete MDP [16].



(a) 0 iterations (b) 1 iteration (c) 2 iterations

Figure 7.2: Policy Iteration on Gridworld



(a) 0 iterations (b) 1 iteration (c) 2 iterations

(d) 3 iterations (e) 4 iterations (f) 5 iterations

(g) 6 iterations (h) 10 iterations (i) 20 iterations

Figure 7.3: Value Iteration on Gridworld

7.2 Formalization Effort

This section gives insight into the design decisions and effort that went into the formalization as it is part of this thesis.

Formalization Effort The formalization presented was carried out over the course of four months. An additional two months were required to gain the necessary background knowledge on MDPs and Isabelle/HOL libraries. In total, our formalization encompasses approximately 5000 lines, around 1000 of which are a small adaptation of existing libraries. The formalization effort can be split up as follows:

Component	Lines of Code
Utilities	600
Construction of the trace space (small adaptations)	1000
Denotational semantics	400
MDPs with rewards	1800
Value Iteration	500
Policy Iteration	700
Gridworld Example	250

Design Decisions and Challenges In our formalization, we represent special classes of functions with designated types. Bounded functions use the type $\_ \Rightarrow_b \_$, bounded linear functions are represented using the type $\_ \Rightarrow_L \_$. This allows us to make these types instances of type classes such as real-vector. Hence we get to use the familiar notation for addition, infinite sums and scaling of vectors also for bounded (linear) functions. Furthermore, we can use all of the theory developed for vector spaces directly.

Alternatively, we could use the ordinary function type and add assumptions on boundedness and linearity of the operators to our theorems. However, in this case we need to introduce our own notation and transfer proofs from vector spaces to functions. On the positive side, we would no longer need to insert coercions to convert between the specialized and general function types.

We faced a similar choice when defining the different classes of policies. Here we could have used predicates to classify policies are deterministic or markovian. Again, we settled on using special types for special classes of policies, with similar tradeoffs as above. As most of the formalization uses deterministic and markovian policies, using a simpler type has many advantages.The type $\sigma \Rightarrow \alpha$ encapsulates most of the assumptions of a deterministic decision rule on the type level. A function of this type is more ergonomic to use than a function of type $\sigma \Rightarrow \alpha$ pmf together with an assertion that it represents a Dirac measure.

We redefine MDPs and construct our own trace space instead of directly building on the formalization by Hölzl [11]. This was necessary, as their formalization does not allow for randomized action choice in decision rules, which we need to prove optimality of markovian policies. A problematic consequence is the considerable amount of duplicated

code that results from this decision. Future work should be focused on a unification of the different ways to generate the trace spaces for Markov chains, Markov processes and MDPs.

Our formalization defines the trace space for MDPs with general measurable action and trace spaces. Originally, the plan was to formalize value iteration and policy iteration in this generality. However, in this general setting we were not able to prove that markovian policies are as expressive as history dependent policies with regard to denotational semantics. A generalization of the formalizations carried out in thesis could be part of future verification projects (see Section 8.1).

8 Conclusion

The formalization developed in this work is a demonstration of the whole framework for verification that Isabelle/HOL provides. Our formalization of Markov decision processes with rewards builds on the rich probability theory library that comes with the Isabelle/HOL library. Our analysis of the expected total discounted reward draws on formalizations of vector spaces, order theory, fixed point theorems and limits. Finally, we use the code generation framework to generate executable code for value iteration and policy iteration on finite MDPs. For this part of the verification project, we profit from an AFP entry that provides an executable implementation of the Gauss-Jordan Algorithm. To summarize, the formalization shows how the different Isabelle/HOL libraries can be integrated and serve as powerful tools for the analysis of algorithms.

Furthermore, we made great use of the proof automation that Isabelle provides, in particular the dedicated measurability proof method and the proof method *auto*. Building on extensive libraries is a challenge in interactive theorem proving, as it often requires considerable effort to understand the underlying theories and discover the facts relevant to prove the current goal. The *sledgehammer* command facilitated the development in this regard with its ability to automatically collect facts from the Isabelle/HOL library and combine them into proofs.

This thesis provides a successful implementation of two verified solution algorithms for MDPs. The gridworld example instantiates our formalizations for a concrete MDP. We show that the generated code is able to find optimal policies with reasonable performance on small MDPs.

8.1 Future Work

The current formalization of MDPs is restricted to discrete state and action spaces. However, most of the theory presented in this thesis may be generalized to a larger class of state-action spaces. A formalization of Markov chains over more general state-action spaces exists in Isabelle/HOL [12]. Similarly, termination of the policy iteration algorithm is currently only shown for finite MDPs, Puterman [16] gives a proof for infinite action spaces.

There is much potential for optimizations in the code generation to achieve better performance. Ultimately however, to compete with the unverified algorithms in practical use, more sophisticated variants of policy and value iteration need to be verified. Notable variants include modified policy iteration, value iteration can be improved by using splitting methods [16]. Changing the input format to factored MDPs also allows for

more efficient solution algorithms [8].

In the context of reinforcement learning, the actor often has only incomplete knowledge of its current state. Such scenarios can be modeled by partially observable MDPs (POMDPs). Extending the formalization to POMDPs is required for the verification of a wide range of practical reinforcement learning algorithms.

Bibliography

[1] J. Aransay and J. Divasón. "Formalisation in higher-order logic and code generation to functional languages of the Gauss-Jordan algorithm." In: *J. Funct. Program.* 25 (2015). DOI: 10.1017/S0956796815000155.

[2] E. Bargiacchi, D. M. Roijers, and A. Nowé. "AI-Toolbox: A C++ library for Reinforcement Learning and Planning (with Python Bindings)." In: *Journal of Machine Learning Research* 21.102 (2020), pp. 1–12.

[3] R. Bellman, I. Glicksberg, and O. Gross. "On the Optimal Inventory Equation." In: *Management Science* 2 (1955), pp. 83–104.

[4] R. Bellman. "A Markovian Decision Process." In: *Indiana Univ. Math. J.* 6 (4 1957), pp. 679–684. ISSN: 0022-2518.

[5] I. Chadès, G. Chapron, M.-J. Cros, F. Garcia, and R. Sabbadin. "MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems." In: *Ecography* 37.9 (2014), pp. 916–920.

[6] M. Eberl, J. Hölzl, and T. Nipkow. "A Verified Compiler for Probability Density Functions." In: *CoRR* abs/1707.06901 (2017). arXiv: 1707.06901.

[7] M. Giry. "A categorical approach to probability theory." In: *Categorical Aspects of Topology and Analysis*. Ed. by B. Banaschewski. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 68–85. ISBN: 978-3-540-39041-1.

[8] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. "Efficient Solution Algorithms for Factored MDPs." In: *J. Artif. Intell. Res.* 19 (2003), pp. 399–468. DOI: 10.1613/jair.1000.

[9] F. Haftmann and L. Bulwahn. "Code generation from Isabelle/HOL theories." In: *Part of the Isabelle documentation: http://isabelle. in. tum. de/dist/Isabelle2017/doc/codegen. pdf* (2013).

[10] J. Hölzl. "Construction and stochastic applications of measure spaces in higher-order logic." PhD thesis. Technical University Munich, 2013.

[11] J. Hölzl. "Markov Chains and Markov Decision Processes in Isabelle/HOL." In: *J. Autom. Reason.* 59.3 (2017), pp. 345–387. DOI: 10.1007/s10817-016-9401-5.

[12] J. Hölzl. "Markov processes in Isabelle/HOL." In: *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*. Ed. by Y. Bertot and V. Vafeiadis. ACM, 2017, pp. 100–111. DOI: 10.1145/3018610.3018628.

[13] J. Hölzl and A. Heller. "Three Chapters of Measure Theory in Isabelle/HOL." In: *Interactive Theorem Proving - Second International Conference, ITP 2011, Berg en Dal, The Netherlands, August 22-25, 2011. Proceedings*. Ed. by M. C. J. D. van Eekelen, H. Geuvers, J. Schmaltz, and F. Wiedijk. Vol. 6898. Lecture Notes in Computer Science. Springer, 2011, pp. 135–151. DOI: 10.1007/978-3-642-22863-6_12.

[14] J. Hölzl, F. Immler, and B. Huffman. "Type Classes and Filters for Mathematical Analysis in Isabelle/HOL." In: *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*. Ed. by S. Blazy, C. Paulin-Mohring, and D. Pichardie. Vol. 7998. Lecture Notes in Computer Science. Springer, 2013, pp. 279–294. DOI: 10.1007/978-3-642-39634-2_21.

[15] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media, 2002.

[16] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. ISBN: 978-0-47161977-2. DOI: 10.1002/9780470316887.

[17] L. S. Shapley. "Stochastic games." In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[18] K. Vajjha, A. Shinnar, B. M. Trager, V. Pestun, and N. Fulton. "CertRL: formalizing convergence proofs for value and policy iteration in Coq." In: *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*. Ed. by C. Hritcu and A. Popescu. ACM, 2021, pp. 18–31. DOI: 10.1145/3437992.3439927.

[19] S. Wimmer and J. Hölzl. "MDP + TA = PTA: Probabilistic Timed Automata, Formalized (Short Paper)." In: *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*. Ed. by J. Avigad and A. Mahboubi. Vol. 10895. Lecture Notes in Computer Science. Springer, 2018, pp. 597–603. DOI: 10.1007/978-3-319-94821-8_35.