Towards Lifelong Reinforcement Learning through Temporal Logics and Zero-Shot Composition

School of Computer Science & Applied Mathematics
University of the Witwatersrand

GERAUD NANGUE TASSE
2291200

Supervised by: Benjamin Rosman & Steven James

October 1, 2024

A thesis submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

This thesis addresses the fundamental challenge of creating agents capable of solving a wide range of tasks in their environments, akin to human capabilities. For such agents to be truly useful and be capable of assisting humans in our day-to-day lives, we identify three key abilities that general purpose agents should have: *Flexibility*, *Instructability*, and *Reliability* (FIRe). Flexibility refers to the ability of agents to adapt to various tasks with minimal learning; instructability involves the capacity for agents to understand and execute task specifications provided by humans in a comprehensible manner; and reliability entails agents' ability to solve tasks safely and effectively with theoretical guarantees on their behavior.

To build such agents, reinforcement learning (RL) is the framework of choice given that it is the only one that models the agent-environment interaction. It is also particularly promising since it has shown remarkable success in recent years in various domains—including gaming, scientific research, and robotic control. However, prevailing RL methods often fall short of the FIRe desiderata. They typically exhibit poor *sample efficiency*, demanding millions of environment interactions to learn optimal behaviors. *Task specification* relies heavily on hand-designed reward functions, posing challenges for non-experts in defining tasks. Moreover, these methods tend to specialize in single tasks, lacking guarantees on the broader adaptability and behavior robustness desired for lifelong agents that need solve *multiple tasks*.

Clearly, the regular RL framework is not enough, and does not capture important aspects of what makes humans so general—such as the use of language to specify and understand tasks. To address these shortcomings, we propose a principled framework for the logical composition of arbitrary tasks in an environment, and introduce a novel knowledge representation called World Value Functions (WVFs) that will enable agents to solve arbitrary tasks specified using language. The use of logical composition is inspired by the fact that all formal languages are built upon the rules of propositional logics. Hence, if we want agents that understand tasks specified in any formal language, we must define what it means to apply the usual logic operators (conjunction, disjunction, and negation) over tasks. The introduction of WVFs is inspired by the fact that humans seem to always seek general knowledge about how to achieve a variety of goals in their environment, irrespective of the specific task they are learning.

Our main contributions include: (i) **Instructable agents**: We formalize the logical composition of arbitrary tasks in potentially stochastic environments, and ensure that task compositions lead to rewards minimising undesired behaviors. (ii) **Flexible agents**: We introduce WVFs as a new objective for RL agents, enabling them to solve a variety of tasks in their environment. Additionally, we demonstrate zero-shot skill composition and lifelong sample efficiency. (iii) **Reliable agents**: We develop methods for agents to understand and execute both natural and formal language instructions, ensuring correctness and safety in task execution, particularly in real-world scenarios. By addressing these challenges, our framework represents a significant step towards achieving the FIRe desiderata in AI agents, thereby enhancing their utility and safety in a lifelong learning setting like the real world.

Declaration

I, Geraud Nangue Tasse, hereby declare the contents of this research thesis to be my own work. This thesis is submitted for the degree of Doctor of Philosophy in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

[illegible]

01/10/2024

Signature

Date

Acknowledgements

From being a below-average primary school student who barely made the passing threshold to a soon-to-be Doctor of Philosophy, I still sometimes find it hard to believe that I have reached this point. This achievement is a testament to the amazing people who have knowingly or unknowingly supported me throughout this journey. In particular, I owe an enormous debt of gratitude to my supervisors, Benji and Steve—without whom this thesis would never have been completed.

To Benji: I still remember our first conversation back in 2018, when I reached out to you as a potential master's supervisor. It was a video call interview, and before that day, I had only a vague idea of what reinforcement learning was. I was always fascinated by human intelligence and had dreamed about creating artificial intelligence as general as humans, but I was lost about how to break it down into meaningful research milestones. Your brief yet illuminating explanation of reinforcement learning and the early work on skill composition by members of RAIL lab gave me direction. That conversation became the foundation of my master's research and ultimately this PhD thesis. Since then, your guidance has been unwavering, from recommending books and lectures to guiding me through research challenges. Your influence has been immeasurable.

To Steve: Words cannot fully express the impact you have had on my journey as a researcher. The effort you put into helping me write my first paper was phenomenal, from breaking down your skill composition paper when I was struggling with it to guiding me through my own work. Always helping me write papers often up until the last moments before the deadline. That first paper based on my masters research became my first NeurIPS publication. I am incredibly thankful for the countless hours you spent mentoring me and look forward to collaborating with you in the years to come.

I am deeply grateful to both Benji and Steve for their guidance over the past five years. Without Benji, I likely wouldn't have received the IBM PhD Fellowship, which supported me financially and led to an invaluable internship at IBM Watson with Tim Klinger and Matt Reimer. Without Steve, that first NeurIPS paper, and perhaps this whole journey, would not have been possible. I apologize for the countless late-night proofreadings that no doubt accelerated your time-to-gray-hairs, especially the moments just before conference deadlines! Although, I hear gray hairs make wise men. May we have more of that in the future.

My gratitude also extends to the brilliant friends, lecturers, and mentors I have had during my academic journey:

At Wits and in the RAIL lab: Brandon, Devon, Shahil, Geethen, Molefe, Kale-Ab, Kwanda, Tamlin, Abdel, Arnold, Thokozile, Gift, and the rest of my Wits friends and acquaintances—thank you for making my postgraduate years memorable. From random lab discussions to off-campus activities and collaborative research, each moment contributed to my growth. A special thank you to Pravesh for your constant support, not only in research but also financially during delays in payments, and for mentoring me during the design of the postgraduate diploma math course—my first experience in course design. I learned a great deal from your guidance during that one-block course. Finally, thank you to all my other research colleagues outside of Wits like Vanya and Nakul initially from Brown university and Mark from Duke university.

At Rhodes University: Charles, thank you for the endless discussions on physics. Vonghai and Cynthia, I will always remember our all-nighters working on our CS honors projects. Rodrigue, those coding project conversations were priceless. James, thank you for being an amazing BSc honors supervisor. And to the rest of my Rhodes friends and acquaintances, thank you for making my undergraduate experience unforgettable. Special thanks to the Dean of science Tony for always being there for advice and for allowing me to take four majors during my BSc—pure math, applied math, physics and electronics, and computer science—which gave me a rich foundation for everything I do today.

I also want to extend a special thanks to my high school friend Ekotto, my first and best friend, who also happened to be my rival in both academics and drawing. We pushed each other to be better, and without that motivation, I would probably still be an average student today. Though we lost contact after I moved to South Africa, I hope we reconnect someday.

Finally, I am eternally grateful to my family. To my parents, Josephine and Etienne, for their unwavering support and encouragement throughout my journey. To my eldest sister Flora, thank you for showing us the way, being the first in our family to earn a PhD (also in machine learning! and I, the last born am the second! the irony), and for setting an inspiring example. To Ghislain, thank you for single-handily supporting my undergraduate education financially, often at the expense of your own life plans. You and Flora have always been ready to spend centuries arguing with me before I end up converging to the fact that your career and life advices were right all along (not always, but definitely for the biggest inflection points of my life). And to my immediate older brother Cyrille, who is the smartest and most hard working person I know, thank you for being my inspiration. Given my complete failure in primary school, I learned alot from just watching you when I was young and also from our endless conversations to this day.

Without the collective effort of my family funding my journey to South Africa, I wouldn't have been able to pursue my undergraduate degree at Rhodes University, meet Benji, Steve, and the amazing people at RAIL lab, and embark on this life-changing journey. To every single person who played a part in making this PhD possible, thank you, truly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

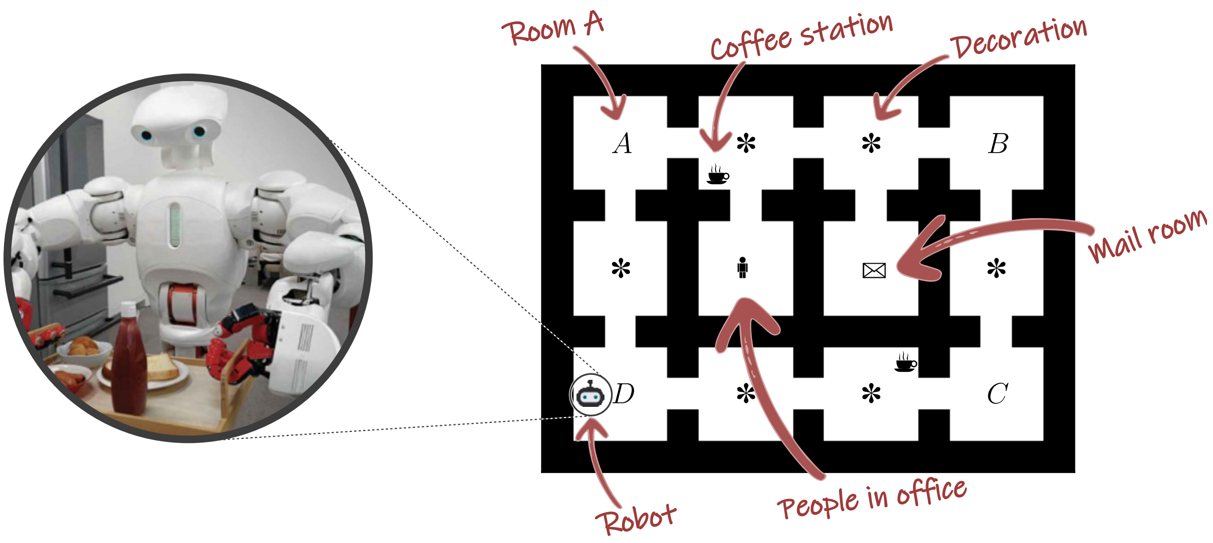


Figure 1.1: Illustration of a robot in an office environment.

Imagine you are managing a bustling office space with the assistance of a robot (illustrated in Figure 1.1). There are multiple tasks you may want the robot to help you with, such as: *Deliver important mails to the office* ***until*** *there is no mails left,* ***then*** *deliver coffee to the office* ***while*** *there are people craving coffee in the office,* ***then*** *patrol the Archive-room, Break-room, Conference-room, and Desk-area for security and to see if anyone else needs help, while* ***never*** *breaking a decoration*. This illustrates one of the ultimate goals in artificial intelligence (AI): Beyond the philosophical and scientific study of *intelligence*, we want to build agents that are capable of solving a variety of cognitive and physical tasks similarly to humans [Russell and Norvig 2016].

For such agents to be maximally useful and assist humans in their day-to-day lives solving many real life problems—for example domestic robots in our homes, industrial robots in factories, and service robots in hospitals and public spaces—we identify three main desiderata: agents should have ***F****lexibility*, ***I****nstructability*, and ***Re****liability* (FIRe).

(i) **Flexibility :** Similarly to humans, there is not one specific task we want an agent to specialise in. Instead, agents should be able to solve a variety of tasks in their environment. Preferably, this should also require minimal amount of learning. For example, the office robot should be able to navigate to various locations like the coffee and decoration ones, and should also be able to avoid the decorations while getting coffee.

(ii) **Instructability :** We should be able to specify what task we want the agent to solve, and the agent should be able to understand said specification to correctly achieve it. Preferably, this should be done in a human understandable way, such that it is not just AI experts that can specify tasks. For example, the office robot should be able to understand and solve the natural language specified task where it must deliver mail, then coffee then patrol the rooms.

(iii) **Reliability :** Agents should be able to solve any task given to them with theoretical guarantees on their behaviour. This is particularly important if we truly want to ultimately have agents that can safely act alongside humans in the real world and be helpful. For example, the office robot should be able to solve any task in its environment safely, such as delivering mails to the office and patrolling rooms A-B-C-D without breaking decorations.

1.1 Reinforcement learning

One way humans learn to make decisions and adapt behaviours is based on feedback from the environment. From the moment we are born, we initially explore different *policies*—what actions to take in different states of the environment—to understand their consequences. When an action leads to a favorable outcome or reward, the we are more likely to repeat that action in similar situations in the future. Conversely, if an action results in a negative outcome or punishment, the we are less likely to repeat that action. Over time, through repeated experiences, we develop associations between actions and cumulative rewards (a *value function*), allowing us to make more informed decisions to maximise rewards. The underlying neural mechanisms involve the dopamine system, which plays a crucial role in encoding reward prediction errors and updating behavioural strategies accordingly [Schultz *et al.* 1997].

This process motivates the sub-field of reinforcement learning (RL) in AI. Here, an agent receives a scalar reward signal as it acts in its environment, and learns by trial and error what actions maximises its returns [Sutton and Barto 1998]. RL is a promising framework for building general agents like humans, given its similarity to how humans learn and the numerous successes it has had within the last decade in tasks such as: playing games at super-human levels [Mnih *et al.* 2015; Fu 2016; Schrittwieser *et al.* 2020; Niu *et al.* 2024], helping with scientific discoveries [Degrave *et al.* 2022; Fawzi *et al.* 2022], and robot control in simulation and reality [Mirowski *et al.* 2017; Peng *et al.* 2018; Zhao *et al.* 2020; Zhou *et al.* 2023].

Despite these achievements, prevailing RL methods often fall short of the FIRe desiderata:

(i) **Sample efficiency ():** They typically exhibit poor *sample efficiency*, demanding millions of environment interactions to learn optimal behaviours. While this is similar to humans in that learning complex tasks is often slow, unlike humans, RL agents usually must learn each new task independently. This lack of flexibility is clearly undesirable for robots that will need to assist us in the real world— which are often heavily constrained in terms of time, energy, and resources. Additionally, wear and tear from repeated trial and error could lead to faulty robots with safety risks and pricey repairs.

(ii) **Task specification** (**)**:** They rely heavily on hand-designed reward functions, posing challenges for non-experts in defining tasks. The reward function often requires multiple iterations of fine-tuning (*reward engineering*) by an expert to capture the intended task, and often leads to unintended side effects or undesirable behaviours that exploit loopholes in the reward system (*reward hacking*) when misspecified [Amodei *et al.* 2016]. This lack of instructability is clearly undesirable as it compromise safety and efficiency in real-world applications, potentially causing harm to humans or the environment.

(iii) **Generalisation** (**)**:** They often specialise in a specific task and lack guarantees on the broader adaptability and behaviour robustness desired for lifelong agents that need to solve multiple tasks. This lack of reliability is clearly undesirable as it undermines trust and confidence in using RL agents for various tasks, hindering their widespread adoption and integration into society and industry.

A major challenge is thus in designing sample-efficient agents that are easily instructable and that can transfer their existing knowledge to solve new tasks quickly [Taylor and Stone 2009].

1.2 Lifelong learning

The FIRe desiderada is particularly important in the lifelong setting, where just like humans in the real word, an agent is presented with a number of tasks during its lifetime and should leverage knowledge learned from previous tasks to solve new ones [Thrun 1996]. However, given the challenges outlined in Section 1.1, it is clear that the standard RL framework does not capture important features that make humans such good lifelong learners. For example, language plays and important role in the way humans understand tasks and their solutions.

Language, whether it be natural, such as English as spoken by humans, or formal, like linear temporal logics (LTL) [Pnueli 1977] as studied in AI, provides an intuitive way for people to specify tasks and instructions to satisfy goals. This makes humans instructable (). However, instruction following is a difficult problem for agents because they need to simultaneously learn (a) the meaning of the instructions to solve the task, (b) the representation of the world in which the task is to be solved, and (c) a sequence of actions that will lead to task completion. This is even more challenging in the lifelong learning setting, where an agent must also acquire knowledge that will allow it to efficiently solve new unseen tasks ().

One of the most common approaches to this problem is to encode a language command into a real-valued vector embedding. The language embedding, together with the agent's current state, is then used to parametrize the agent's policy [Blukis *et al.* 2020; Chaplot *et al.* 2018; Tambwekar *et al.* 2021]. These end-to-end methods succeed in the presence of a simulator, but require large amounts of data. These methods also suffer from generalization issues as the agents require a large number of samples from the environment for every novel task presented [Lake and Baroni 2018], and they provide no guarantees on the resulting behaviours ().

1.3 The need for composition

To overcome these issues simultaneously, one promising approach is to leverage the principle of *compositionality*, which states that *a complex expression's meaning is defined by the meanings of its constituent expressions and the rules used to combine the meanings of these expressions* [Szabó 2020]. Here, an agent first learns individual skills and then combines them for novel behaviours.

There are several notions of compositionality in the literature, such as *spatial* composition [Todorov 2009; van Niekerk *et al.* 2019], where skills are combined to produce a new single behaviour to be executed to achieve sets of high-level goals "pick up an object that is both blue and a box"), and *temporal* composition [Sutton *et al.* 1999; Jothimurugan *et al.* 2021], where sub-skills are invoked one after the other to achieve sequences of high-level goals (for example, "pick up a blue object and then a box").

1.3.1 Spatial composition

Spatial composition is commonly achieved through a weighted combination of skills [Barreto *et al.* 2018; Peng *et al.* 2019; Alver and Precup 2022b]. Notably, work by Nangue Tasse *et al.* [2020b] has demonstrated spatial composition using logic operators. They introduced a formal framework for specifying tasks using conjunction (AND), disjunction (OR), and negation operators (NOT), and how to similarly apply them on learned skills to generate provably optimal solutions to new tasks (). Not only does this allow for safer and interpretable reward specification—since any new task reward function can be composed of well-understood components ()—it also leads to a combinatorial explosion in an agent's ability—producing semantically meaningful behaviours without further learning ( ).

Is this enough? While these agents seem to be FIRe, they suffer from a number of shortcomings. Mainly, notice that spatial composition is not enough for solving long-horizon tasks—such as the example we gave for the office environment in Figure 1.1. In these instances, it is often near impossible for the agent to learn, owing to the large sequence of actions that must be executed before a learning signal is received [Arjona-Medina *et al.* 2019]. Additionally, Nangue Tasse *et al.* [2020b] restricts the specification of tasks and corresponding skills learned to only goal-reaching ones in deterministic environments, making tasks with arbitrary rewards in stochastic environments unspecifiable within the framework (,). Finally, despite the guarantee that task compositions will have the correct semantics, they also provide no guarantee that the tasks being composed have the correct reward functions—such that maximising those rewards indeed leads to desired behaviours ().

1.3.2 Temporal composition

One of the most common approaches to temporal composition is to learn options [Sutton *et al.* 1999]—sub-skills that can be executed sequentially ( ). These are then used for achieving the sub-goals present in tasks specified using formal languages—such linear temporal logics [Pnueli 1977] ()—while learning a high-level policy over the options to provably solve the task, and then reusing the learned options in new tasks [Araki *et al.* 2021; Icarte *et al.* 2022] ().
However, other works like Vaezipoor *et al.* [2021] have proposed end-to-end neural network

architectures for learning sub-skills from a training set which are then demonstrated emperically to enable better sample efficiency and generalisation to similar new tasks.

Is this enough? While these agents also seem to be FIRe, they also suffer from a number of shortcomings. Mainly, Liu *et al.* [2022] observe that for all these prior works, some of the sub-skills (e.g., options) learned from previous tasks can not be transferred satisfactorily to new tasks and provide a method to determine when this is the case (). For example, if the agent has previously learned an option for "getting blue objects" and another for "getting boxes", it can reuse them to "pickup a blue object and then a box", but it cannot reuse them to "pickup a blue object that is not a box, and then a box that is not blue". We can observe that this problem is because all the compositions in prior works are *either strictly temporal or strictly spatial*.

1.4 Aims and contributions

The main aim of this work is to build a principled framework that leverages the principle of composition to simultaneously address the problem of task specification, sample efficiency, and generalisation, thereby bringing us a step closer towards the FIRe desiderata. Precisely, we (i) extend the logical composition framework of Nangue Tasse *et al.* [2020b] to stochastic tasks with arbitrary provably safe rewards; (ii) introduce a new knowledge representation, namely *world value functions* (WVFs), which defines a new objective for RL agents to be provably flexible; (iii) leverage world value functions and logical composition to produce agents that can provably solve a variety of tasks in their environment specified using natural or formal language.

We also focus our attention on model-free reinforcement learning——where an agent only has access to the rewards and observations for each environment interaction when learning. This constraint ensures our framework is applicable even when the reward and transition functions are not known to the agent. We briefly describe our main contributions:

(i) **Instructable agents [Part I]:** In Chapter 3, we formally define the logical composition of tasks and explore the semantic meaning of such compositions. Given that we are now able to specify tasks as a composition of other tasks, Chapter 4 ensures that the rewards of tasks in general minimise the probability of leading to optimal but undesired behaviours.

(ii) **Flexible agents [Part II]:** In Chapter 5, we propose a new goal for RL agents to learn *world value functions* which provably lead to the ability to solve any given goal-reaching task in the environment. We then show in Chapter 6 that zero-shot skill composition now holds for goal-reaching tasks with potentially stochastic dynamics. Finally, we show in Chapter 7 that this leads to agents that are provably sample efficient in lifelong RL.

(iii) **Reliable agents [Part III]:** In Chapter 8, we demonstrate how this ability to understand task compositions and solve them without further learning can then be leveraged by agents to reliably follow natural language instructions. However, the ambiguity in natural language makes it hard to guarantee that the task specification and resulting compositional behaviours are correct. One solution to this is to use formal languages, such as LTL, which can be verified. In Chapter 9, we propose *skill machines*, a method for agents to provably solve formal languages that are regular (like fragments of LTL). We then show in Chapter 10 that this compositional ability can be leveraged by robots to safely act in the real world.

Chapter 2

Background

As described in the introduction, there are three main areas of importance to us for designing general purpose agents capable of solving a variety of tasks in an environment: Reinforcement learning, logical composition, and temporal logic composition. In this chapter, we will give a sufficient description of them as they pertain to the rest of this thesis.

For example, what actually is an environment, a task in the environment, rewards and the various other key terms we have used so far? In Section 2.1, we describe what some of these terms mean formally in the context of RL, and refer the reader to Sutton and Barto [2018] for a more detailed introduction to RL. We then describe lattice theory in Section 2.2.1, since it is the algebraic framework that formalises the notions of conjunction, disjunction, and negation. This is also the framework used by Nangue Tasse *et al.* [2020b] to formalise the logical composition of tasks and skills in RL—under various assumptions on the environment dynamics and task rewards. Since we aim to extend this framework to more general tasks, we also describe it in detail in this section. Finally, we describe formal languages and reward machines in Section 2.3, which are essential for specifying temporal logic tasks.

2.1 Reinforcement learning

Reinforcement learning is a framework representing agents making sequential decisions in order to maximise the sum of a scalar reward. At its core, the process involves the agent interacting with an environment, learning from feedback (rewards) received for its actions, and adjusting its decision-making strategy accordingly (Figure 2.1). This framework is particularly suited for problems where decisions are made over time, and the outcomes of those decisions are influenced by the state of the environment.

The foundation of mordern reinforcement learning lies in Markov decision processes (MDPs) [Puterman 2014], which formalize decision problems by defining states, actions, transition dynamics, and rewards. States represent possible situations in the environment, actions are the choices available to the agent, transition dynamics dictate how the environment changes in response to actions, and rewards quantify the desirability of outcomes.

The objective in reinforcement learning is to develop a policy, a function that maps states to actions, in a way that maximises the cumulative reward the agent expects to receive over time.

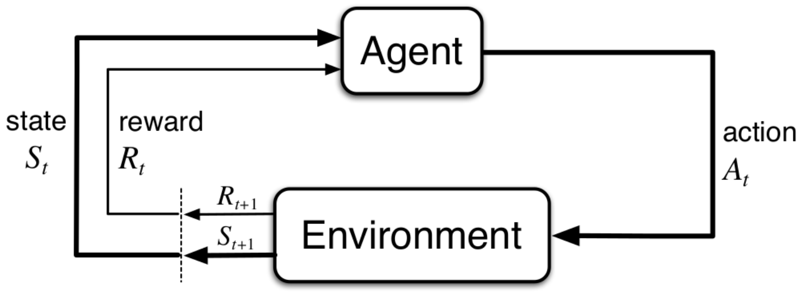


Figure 2.1: Illustration of agent-environment interaction [Sutton and Barto 1998]

Additionally, the value function plays a crucial role, estimating the total reward an agent can expect to accumulate from a given state following a particular policy. We now define these formally in the following sub-sections.

2.1.1 Markov decision processes

Markov decision processes (MDPs) provide a formal framework for modeling decision-making problems in reinforcement learning, where an agent interacts with an environment to achieve specific goals.

A state s of the environment at time t is considered Markov if the probability of transitioning to state s' at time $t+1$ depends only on the current state s and action a, and not on the previous history of states and actions. This property is expressed mathematically as:

$$\mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = \mathbb{P}[S_{t+1} = s' | S_0 = s, A_0 = a, ..., S_t = s, A_t = a]$$

A decision-making problem that satisfies the Markov property for all states is termed a Markov Decision Process (MDP). Formally, an MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where,

- $\mathcal{S}$ is the state space, representing all possible states the environment can be in;
- $\mathcal{A}$ is the action space, denoting all possible actions the agent can take;
- $P\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the transition function, defining the probability of transitioning from one state to another given a particular action;
- $R\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the bounded reward function, defining the immediate reward the agent receives upon transitioning from one state to another given a particular action;
- $\gamma \in [0,1]$ is the discount factor, representing the importance of future rewards relative to immediate rewards.

For clarity, we will focus on MDPs with finite $\mathcal{S}$ and $\mathcal{A}$, but note that the theory also hold for continuous $\mathcal{S}$ and $\mathcal{A}$. The set of tasks can then be naturally defined as follows:

Definition 2.1. *A set of tasks $\mathcal{M}$ is a set of MDPs in the same environment $\langle \mathcal{S}, \mathcal{A}, P, \gamma \rangle$ that differ only in their reward functions.*

2.1.2 Goals and returns

In reinforcement learning the agent's goal is informally to maximise its cumulative future reward. This can be stated as the reward hypothesis:

All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward). [Sutton and Barto 1998]

This notion of goal may at first appear limiting but has proven to be sufficient for a wide range of problems. Formally, the cumulative sum of rewards is defined as the *return*:

$$G_t \coloneqq R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

To ensure that G_t is finite, tasks with $\gamma = 1$ are assumed to be episodic—they terminate at absorbing states. These are states g in an absorbing space $\mathcal{G}$ such that $P(g, a, g) = 1$ and $R(g, a, g) = 0$ for all actions. That is, the agent always receives zero rewards in $g \in \mathcal{G}$ and can never transition out of it once reached.

2.1.3 Policies and value functions

The goal of the agent is to compute a policy that optimally solves a given task. Formally, a policy π is defined as the probability of selecting each action in a given state. That is,

$$\begin{aligned} \pi \colon \mathcal{S} \times \mathcal{A} &\to [0,1] \\ (s,a) &\mapsto \mathbb{P}[A_t = a | S_t = s] \end{aligned}$$

A given policy π is characterised by a state-value function V^π, which is the expected return of starting at a given state and following π thereafter. That is,

$$\begin{aligned} V^\pi \colon \mathcal{S} &\to \mathbb{R} \\ s &\mapsto \mathbb{E}^\pi[G_t | S_t = s] \end{aligned}$$

Similarly, the action-value function Q^π under a policy π is defined as,

$$\begin{aligned} Q^\pi \colon \mathcal{S} \times \mathcal{A} &\to \mathbb{R} \\ (s,a) &\mapsto \mathbb{E}^\pi[G_t | S_t = s, A_t = a] \end{aligned}$$

The action-value function is particularly useful in model-free reinforcement since it enables agents to learn optimal actions for each state without the need for transition dynamics.

Using the recursive property of G_t, the value functions can be shown to satisfy the following Bellman equations,

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s'|s,a) \left(R(s,a,s') + \gamma V^\pi(s') \right) \tag{2.2}$$

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)\left(R(s,a,s') + \gamma V^\pi(s')\right) \tag{2.3}$$

By defining a partial ordering over policies, it can be shown that $\exists \pi^* \geq \pi, \forall \pi$, which leads to the optimal value functions,

$$V^{\pi^*}(s) = V^*(s) \text{ and } Q^{\pi^*}(s,a) = Q^*(s,a) \ \ \forall \pi^*$$

These give rise to the following Bellman optimality equations for value functions,

$$V^*(s) = \max_{a \in \mathcal{A}} \left(\sum_{s' \in \mathcal{S}} P(s,a,s')\left(R(s,a) + \gamma V^*(s')\right) \right) \tag{2.4}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')) \tag{2.5}$$

which are written more succinctly using the Bellman optimality operator as follows,

$$\left[\mathcal{T}Q^*\right](s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left(R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \right] . \tag{2.6}$$

An optimal policy π^* can also be obtained by acting greedily on the optimal action-value function Q^*. That is,

$$\pi^*(s,a) = \begin{cases} 1 & a = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Hence, there always exists a deterministic optimal policy. Given this, we will henceforth only focus on deterministic policies $\pi\colon \mathcal{S} \to \mathcal{A}$ in our theory.

2.1.4 Learning optimal policies and value functions

There are several algorithms for learning optimal policies and value functions in RL [Sutton and Barto 1998], ranging from: (i) **Model-based** ones which assume that the dynamics of the environment are known (or learned), with foundational examples being *policy iteration* and *value iteration* (ii) **Model-free** ones learn solely by trial and error interactions with the environment, with a foundational example being *Q-learning*.

Since we are interested in ultimately having agents that can solve tasks in complex environments with unknown transition probabilities and reward functions, we will constrain ourselves to *model-free* RL. However, we still give a brief description of all these algorithms here, as they will be relevant through out this thesis.

Policy iteration [Sutton and Barto 1998]: Policy iteration is a dynamic programming approach that learns an optimal policy π^* for a given finite MDP by iteratively improving a candidate policy until it converges to the optimal one (Algorithm 1). It alternates between two steps: (i) **Policy evaluation:** Evaluate the value function V^π for the current policy π by iteratively solving the Bellman equation (Equation 2.2) until convergence. (ii) **Policy improvement:** Improve the policy by greedily selecting actions that maximise the expected cumulative reward based on the current value function.

Algorithm 1: Policy iteration

```
Input      : $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$
Initialise : Policy $\pi(s) = a \in \mathcal{A}$, State-value function $V(s) = 0$, value iteration error $\Delta = 1$
  /* Policy evaluation */
  while $\Delta > 0$ do
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ do
      $v' \leftarrow \sum_{s'} P(s'|s,\pi(s))(R(s,\pi(s),s') + \gamma V(s'))$
      $\Delta = \max\{\Delta, |V(s) - v'|\}$
      $V(s) \leftarrow v'$
  /* Policy improvement */
  for $s \in \mathcal{S}$ do
    $\pi(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V(s'))$
```

Value iteration [Sutton and Barto 1998]: Value iteration learns the optimal value function V^* (or Q^*) for a given finite MDP by iteratively improving the value function until it converges to the optimal one. It directly computes the optimal value function by iteratively applying the Bellman optimality equation (Equation 2.5) until it converges (Algorithm 2).

Algorithm 2: Value iteration

```
Input      : $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$
Initialise : Action-value function $Q(s,a) = 0$, value iteration error $\Delta = 1$
  while $\Delta > 0$ do
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ do
      for $a \in \mathcal{A}$ do
        $q' \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q(s',a'))$
        $\Delta = \max\{\Delta, |Q(s,a) - q'|\}$
        $Q(s,a) \leftarrow q'$
```

Q-learning [Watkins 1989]: Q-learning computes the optimal action-value function Q^* for a finite MDP directly from observed transitions without requiring a model of the environment. Here, the agent acts in the environment using random or greedy actions with some probability, and simultaneously uses that experience to update its action-values (Algorithm 3).

2.2 Logical composition

In the previous section, we described the regular RL setting which only pertains to an agent trying to solve a single task. However, as discussed in Chapter 1, we want the agent to be instructable such that it can solve multiple tasks specified as compositions of other tasks. Similarly, we want the agent to be flexible by composing its skills—defined as policies or value functions—to solve said task compositions. How should we then define the RL problem for such agents?

Since language is a natural way of specifying such compositions, and logics is foundational to all such languages, we seek to formalise the logical composition of tasks and skills. To

Algorithm 3: Q-learning

```
Input     : Discount factor $\gamma$, learning rate $\alpha$, exploration rate $\epsilon$
Initialise : Action-value function $Q(s, a) = 0$
foreach episode do
    Observe initial state $s \in \mathcal{S}$
    while episode is not done do
        $a \leftarrow \begin{cases} \arg\max_{a \in \mathcal{A}} Q(s, a) & \text{w.p. } 1 - \varepsilon \\ \text{a random action} & \text{w.p. } \varepsilon \end{cases}$
        Execute $a$, observe reward $r$ and next state $s'$
        $Q(s, a) \xleftarrow{\alpha} \left(r + \max_{a'} Q(s', a')\right) - Q(s, a)$
        $s \leftarrow s'$
```

precisely define these, we can leverage the mathematical framework of lattice theory, since it is the structure that abstracts the rules of propositional logic and set theory.

2.2.1 Lattice theory

Lattice theory extends the study of Boolean algebras and can be approached from either an order theory or abstract algebra perspective [Grätzer 2002]. In the order theoretic view, fundamental concepts include least upper bounds and greatest lower bounds, corresponding to the algebraic operations of join (disjunction) and meet (conjunction). As our emphasis lies on conjunction, disjunction, and negation operators, we primarily adopt the algebraic interpretation, aiming towards constructing a Boolean algebra. Nevertheless, we also incorporate the order theoretic perspective when appropriate.

In order theory, a lattice is a partial order $(\mathcal{L}, \leq)$ in which every pair of elements $a, b \in \mathcal{L}$ has a *least upper bound (their supremum)* and a *greatest lower bound (their infimum)* [Grätzer 2011]. The equivalent algebraic definition is as follows:

Definition 2.2. *A Lattice algebra $(\mathcal{L}, \vee, \wedge)$ is a set $\mathcal{L}$ equipped with the binary operators $\vee$ and $\wedge$ which satisfies the following lattice axioms for a, b, c in $\mathcal{L}$:*

(i) Idempotence: $a \wedge a = a \vee a = a$.

(ii) Commutativity: $a \wedge b = b \wedge a$ and $a \vee b = b \vee a$.

(iii) Associativity: $a \wedge (b \wedge c) = (a \wedge b) \wedge c$ and $a \wedge (b \vee c) = (a \vee b) \vee c$.

(iv) Absorption: $a \wedge (a \vee b) = a \vee (a \wedge b) = a$.

A lattice order $(\mathcal{L}, \leq)$ induces a lattice algebra $(\mathcal{L}, \vee, \wedge)$ with $\vee$ and $\wedge$ given by $a \vee b := \sup\{a, b\}$ and $a \wedge b := \inf\{a, b\}$. We can then call a set $\mathcal{B} \subseteq \mathcal{L}$ a *basis* of $\mathcal{L}$ if it is the smallest set that can generate all elements of $\mathcal{L}$ by applying the logical operators of $\mathcal{L}$.[1]

While the lattice algebra defines conjunction and disjunction, it says nothing of the negation operator. The *De Morgan algebra*, on the other hand, provides an intuitive notion of negation.

Definition 2.3. *A De Morgan algebra $(\mathcal{L}, \vee, \wedge, \neg)$ is a set $\mathcal{L}$ equipped with the binary operators $\vee$ and $\wedge$, and the unary operator $\neg$ (involution, negation), which satisfies the following De Morgan algebra axioms for a, b, c in $\mathcal{L}$:*

(i) All the lattice axioms.

(ii) Distributivity: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ and $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

(iii) Identity: there exists $\mathbf{0}, \mathbf{1}$ in $\mathcal{L}$ such that $\mathbf{0} \wedge a = \mathbf{0}$, $\mathbf{0} \vee a = a$, $\mathbf{1} \wedge a = a, \mathbf{1} \vee a = \mathbf{1}$.

(iv) De Morgan involution: $\neg\neg a = a$ and $\neg(a \vee b) = \neg a \wedge \neg b$.

In other words, a De Morgan algebra is a bounded distributive lattice equipped with a De Morgan involution (a negation operator that satisfies the De Morgan laws).

Finally, notice that the De Morgan involution does not necessarily satisfy the law of the excluded middle ($a \vee \neg a = \mathbf{1}$) and the law of non-contradiction ($a \wedge \neg a = \mathbf{0}$), which are neccessary to have the familiar semantics of propositional logic and set theory [Grätzer 2011]. A Boolean algebra gives us those semantics by enforcing said laws:

Definition 2.4. *A Boolean algebra $(\mathcal{L}, \vee, \wedge, \neg)$ is a set $\mathcal{L}$ equipped with the binary operators $\vee$ (disjunction) and $\wedge$ (conjunction), and the unary operator $\neg$ (negation), which satisfies the following Boolean algebra axioms for a, b, c in $\mathcal{L}$:*

(i) All the De Morgan algebra axioms.

(ii) Complements: $\forall a \in \mathcal{L}$, there exists $\neg a = a' \in \mathcal{L}$ such that $a \wedge a' = \mathbf{0}$ and $a \vee a' = \mathbf{1}$.

2.2.2 Task composition

By leveraging the lattice theory framework, Nangue Tasse *et al.* [2020b] formalise the disjunction, conjunction, and negation of tasks as operators acting on a set of tasks in lattice structures. They consider a family of related tasks $\mathcal{M}$ with an absorbing set $\mathcal{G} \subseteq \mathcal{S}$ and restricted by the following assumption:

Assumption 2.1 (Nangue Tasse *et al.* [2020b]). *Given a set of tasks $\mathcal{M}$ in an environment $\langle \mathcal{S}, \mathcal{A}, P, \gamma \rangle$ with absorbing states $\mathcal{G} \subseteq \mathcal{S}$,*

(i) the transition dynamics P are deterministic, and

[1]For simplicity, we adopt the convention of Grätzer [2011] of referring to an algebraic structure (e.g., the lattice $(\mathcal{L}, \vee, \wedge)$) by the set on which it is defined ($\mathcal{L}$), when the distinction is clear given the context.

(ii) the reward functions differ between tasks only in the boundary set $\mathcal{G}$. That is, for all $M_1, M_2 \in \mathcal{M}$ with reward functions r_{M_1} and r_{M_2} respectively, we have that $r_{M_1}(s,a) = r_{M_2}(s,a) = R_{s,a} \in \mathbb{R}$ for all $s \in \mathcal{S} \setminus \mathcal{G}$ and $a \in \mathcal{A}$.

The logic operators over tasks are then defined as follows:

Definition 2.5. *Let $\mathcal{M}$ be a set of tasks with bounds $\mathcal{M}_{INF}, \mathcal{M}_{SUP} \in \mathcal{M}$ such that,*

$$R_{\mathcal{M}_{SUP}}(s,a) := \max_{M \in \mathcal{M}} r_M(s,a) \qquad R_{\mathcal{M}_{INF}}(s,a) := \min_{M \in \mathcal{M}} r_M(s,a)$$

Define the $\neg$, $\vee$, and $\wedge$ operators over $\mathcal{M}$ as

$$\neg(M) := \langle \mathcal{S}, \mathcal{A}, P, R_{\neg M}, \gamma \rangle, \text{ where } R_{\neg M}(s,a) := (R_{\mathcal{M}_{SUP}}(s,a) + R_{\mathcal{M}_{INF}}(s,a)) - R_M(s,a)$$

$$\vee(M_1, M_2) := \langle \mathcal{S}, \mathcal{A}, P, R_{M_1 \vee M_2}, \gamma \rangle, \text{ where } R_{M_1 \vee M_2}(s,a) := \max\{r_{M_1}(s,a), r_{M_2}(s,a)\}$$

$$\wedge(M_1, M_2) := \langle \mathcal{S}, \mathcal{A}, P, R_{M_1 \wedge M_2}, \gamma \rangle, \text{ where } R_{M_1 \wedge M_2}(s,a) := \min\{r_{M_1}(s,a), r_{M_2}(s,a)\}$$

Nangue Tasse *et al.* [2020b] then show that these logical operators over tasks form a lattice, and under increasing restrictions on the reward functions they form a De Morgan and finally a Boolean algebra. Precisely, to obtain a Boolean algebra, they assume that for all tasks in a bounded set of tasks $\mathcal{M}$, the set of possible terminal rewards consists of only two values:

Assumption 2.2 (Nangue Tasse *et al.* [2020b]). *Consider a set of tasks $\mathcal{M}$ in an environment $\langle \mathcal{S}, \mathcal{A}, P, \gamma \rangle$ with absorbing states $\mathcal{G} \subseteq \mathcal{S}$. For all (g,a) in $\mathcal{G} \times \mathcal{A}$, we have that $R(g,a) \in \{R_{MIN}, R_{MAX}\} \subset \mathbb{R}$ with $R_{MIN} \leq R_{MAX}$.*

While restrictive, the Boolean algebra formalism allows for goal-reaching tasks with sparse rewards to be specified, and also gives us a notion of basis tasks—a minimal set of tasks that can be composed to specify the largest number of composite tasks.

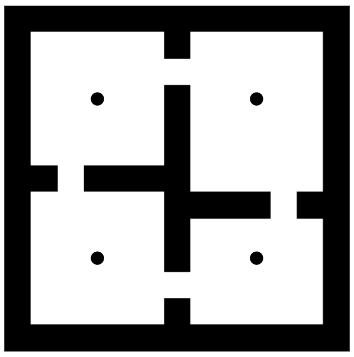


Figure 2.2: The layout of the Four Rooms domain. The circles indicate goals the agent must reach. We refer to the goals as `top-left`, `top-right`, `bottom-left`, and `bottom-right`.

Example 2.1. *Consider the Four Rooms domain [Sutton* et al. *1999], where an agent must navigate a gridworld to particular rooms. The agent can move in any of the four cardinal directions at each timestep, but colliding with a wall leaves the agent in the same location. There*

Goals	M_D	M_R
top-left	0	0
top-right	0	1
bottom-left	1	0
bottom-right	1	1

(a) Goals labelled by the well order $\leq$ given by: `top-left` $\leq$ `top-right` $\leq$ `bottom-left` $\leq$ `bottom-right`.

Goals	M_T	M_L
bottom-right	0	0
bottom-left	0	1
top-right	1	0
top-left	1	1

(b) Goals labelled by the well order $\leq$ given by: `bottom-right` $\leq$ `bottom-left` $\leq$ `top-right` $\leq$ `top-left`.

Table 2.1: Basis tasks induced by various well orders on $\mathcal{G}$. Each column represents a basis task, where **0** or **1** for goal g on task M means respectively reward of $R_M(g,a) = R_{\text{MIN}}$ or $R_M(g,a) = R_{\text{MAX}}$ $\forall a \in \mathcal{A}$. For example, M_D is the basis task requiring the agent to navigate to the bottom-left or bottom-right goals. Note that the goal rewards are sufficient to specify the tasks since the internal rewards are the same across all tasks.

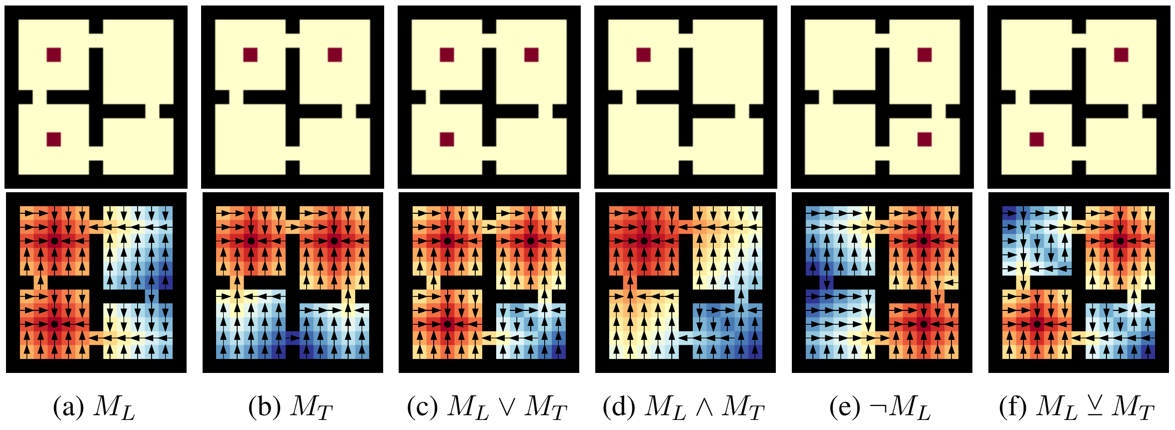
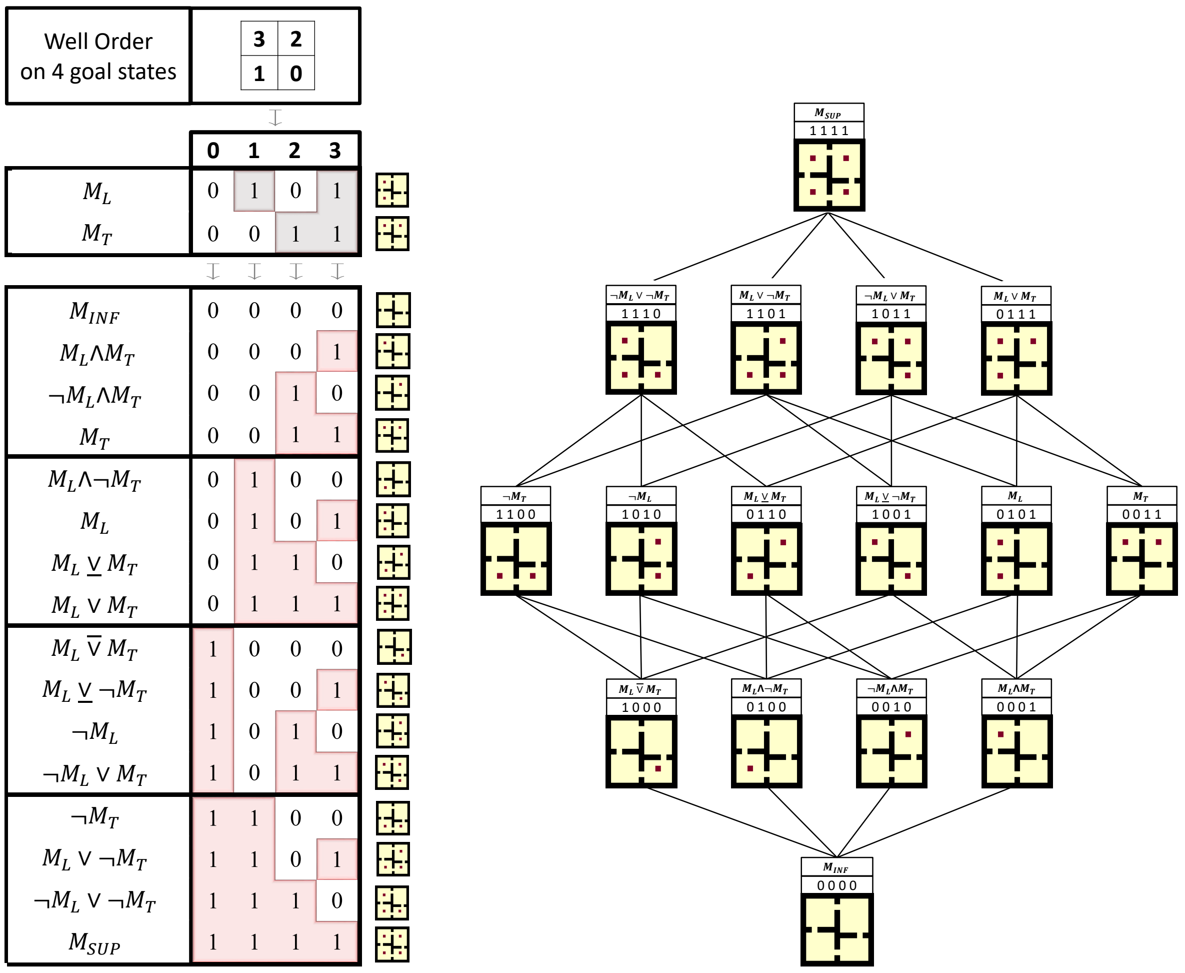


Figure 2.3: Task composition of basis tasks in the Four Rooms domain where $\veebar$ represents exclusive disjunction. (a–b) Rewards for the basis tasks. (c–f) Rewards for the composed tasks. The optimal policies (bottom row arrows) and their corresponding value functions (bottom row heat maps) are obtained using Q-learning.

is a fifth action for "done" that the agent chooses to achieve goals $\mathcal{G}$ (centre of a room). A goal position only becomes terminal if the agent chooses the done action in it. The non-terminals rewards are $R_{MIN} = -0.1$ and the goal rewards (rewards on the terminal set) are binary ($R_{MIN} = -0.1$ or $R_{MAX} = 2$). The discount factor used is $\gamma = 1$. Figure 2.2 illustrates the layout of the environment and the goals the agent must reach.

We can select a minimal set of generator tasks (basis tasks) by assigning each goal a binary number, and then using the columns of the table to select the tasks. Since the set of achievable goals (the terminal set) is finite, we this assignment can be done using a Boolean table. We first assign labels to the individual goals by defining a well order over the set $\mathcal{G}$. Since there are four goals, the number of basis tasks induced by this well order is $\lceil |\log_2 \mathcal{G}| \rceil = 2$. Table 2.1 illustrates how different well orders on $\mathcal{G}$ leads to different choices of basis tasks.



	0	1	2	3
M_L	0	1	0	1
M_T	0	0	1	1
M_{INF}	0	0	0	0
$M_L \wedge M_T$	0	0	0	1
$\neg M_L \wedge M_T$	0	0	1	0
M_T	0	0	1	1
$M_L \wedge \neg M_T$	0	1	0	0
M_L	0	1	0	1
$M_L \veebar M_T$	0	1	1	0
$M_L \vee M_T$	0	1	1	1
$M_L \bar{\vee} M_T$	1	0	0	0
$M_L \veebar \neg M_T$	1	0	0	1
$\neg M_L$	1	0	1	0
$\neg M_L \vee M_T$	1	0	1	1
$\neg M_T$	1	1	0	0
$M_L \vee \neg M_T$	1	1	0	1
$\neg M_L \vee \neg M_T$	1	1	1	0
M_{SUP}	1	1	1	1

(a) Boolean table of basis and composed tasks.

(b) Hasse diagram of the Boolean task algebra.

Figure 2.4: Boolean table and Hasse diagram for the Four Rooms domain. (a) A well order on $\mathcal{G}$ that labels the goals, the induced basis tasks, the logical expressions for all 16 compositions of the basis tasks, and their Boolean values and rewards. (b) The Boolean task algebra showing the rewards for all 16 tasks in $\mathcal{M}$, together with the Boolean values and logical expressions that generate them from the basis tasks.

Consider, for example, the well order on $\mathcal{G}$ shown in Table 2.1b. The basis tasks induced are M_L and M_T, in which an agent must navigate to the two left rooms and the two top rooms respectively. Figure 2.3 shows the rewards and optimal policies of the tasks specified by some of their logical compositions. Figure 2.4 shows the Boolean table and Hasse diagram for all $2^{2^k} = 16$ tasks generated by the $k = 2$ basis tasks, which spans the whole set of tasks, $|\mathcal{M}| = 2^4$. A Hasse diagram illustrates a lattice $\mathcal{M}$ by drawing an edge between $M_1 \in \mathcal{M}$ and $M_2 \in \mathcal{M}$ if they are comparable, and M_1 is drawn below M_2 if $M_1 \leq M_2$ [Grätzer 2011]. Hence, $M_L \wedge M_T$ is connected to M_L and M_T directly below them because it is their greatest lower bound. Similarly, $M_L \vee M_T$ is connected to M_L and M_T directly above them because it is their least upper bound.

2.2.3 Extended value functions

For an agent to be able to solve new tasks optimally in an environment without extra learning, it needs to have gained sufficient information from its experience when learning to solve previous tasks. Its objective during learning hence should not just be how to optimally solve the current task, but how to gain as much knowledge as possible from its experience while doing so. Nangue Tasse *et al.* [2020b] recently introduced a new type of value function called *extended value functions* (EVFs) that captures this idea. They then showed that the rich knowledge it encodes is provably sufficient to solve the logical combination of tasks without further learning. To achieve these, they further restrict the task space with the following assumption:

Assumption 2.3 (Nangue Tasse *et al.* [2020b]). *Consider a set of tasks $\mathcal{M}$. $\mathcal{M}$ only contains deterministic shortest path tasks ($\gamma = 1$) as in van Niekerk* et al. *[2019]; Nangue Tasse* et al. *[2020a], or it only contains discounted tasks ($\gamma \in [0, 1)$) with zero non-terminal rewards and non-negative terminal rewards.*

An EVF is a goal-oriented value function ($\mathbf{Q}$) learned from a single task but that encodes more information about the solved task than a normal value function (Q). Let $\mathcal{G} \subset \mathcal{S}$ be the set of goals in an environment. Nangue Tasse *et al.* [2020b] extend the standard rewards and value functions used by an agent to define goal-oriented versions as follows:

Definition 2.6. *The extended reward function $\mathbf{R} : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ is given by the mapping*

$$(s, g, a) \mapsto \begin{cases} \mathbf{R}_{MIN} & \text{if } g \neq s \text{ and } s \in \mathcal{G} \\ r(s, a) & \text{otherwise,} \end{cases} \tag{2.8}$$

where $\mathbf{R}_{MIN} \leq \min\{R_{\text{MIN}}, (R_{\text{MIN}} - R_{\text{MAX}})D\}$, and D is the diameter of the MDP [Jaksch et al. *2010].*

Definition 2.7. *The extended action-value function $\mathbf{Q}^{\bar{\pi}} : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ is given by the mapping*

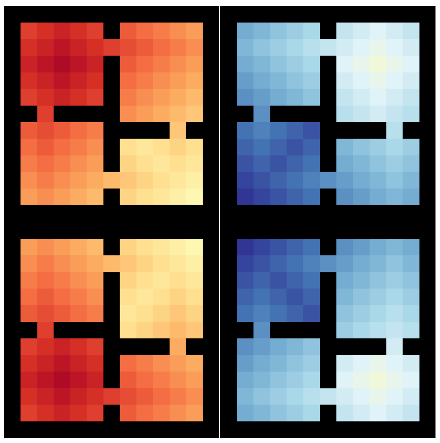
$$(s, g, a) \mapsto \mathbf{R}(s, g, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)\bar{V}^{\bar{\pi}}(s', g), \tag{2.9}$$

where $\bar{V}^{\bar{\pi}}(s, g) = \mathbb{E}_{\bar{\pi}}\left[\sum_{t=0}^{\infty} \mathbf{R}(s_t, g, a_t)\right]$ is the extended state-value function and $\bar{\pi} : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to [0, 1]$ is the extended policy.

By penalising the agent for achieving goals different from those it wanted to reach ($\mathbf{R}_{MIN}$ if $g \neq s$ and $s \in \mathcal{G}$), the extended reward function has the effect of driving the agent to learn how to separately achieve all desirable goals. Importantly, the standard reward and value functions can be recovered from their extended versions by simply maximising over goals. As such, the agent can also recover the task policy by maximising over both goals and actions: $\pi^*(s) \in \arg\max_{a \in \mathcal{A}} \max_{g \in \mathcal{G}} \mathbf{Q}^*(s, g, a)$.

Notice how EVFs are similar to other goal-oriented value functions [Schaul *et al.* 2015; Kaelbling 1993], except that they use task-dependent reward functions. This leads to agents that learn to achieve any goal that is achievable, and also the value of achieving those goals for the task.

Example 2.2. *Consider the Four Rooms domain in Example 2.1. Figure 2.5 shows the EVF of an agent trained to solve the left task. Here the agent is required to navigate to the center of the*


(a) The EVF. Each plot shows the value of all internal states for each goal. For example, the top-left plot shows the learned value of each position with respect to the top-left goal.

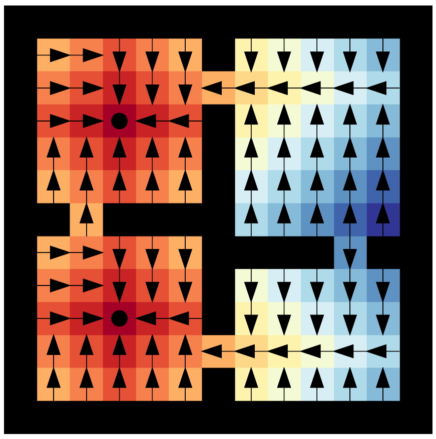

(b) The value function. This is obtained from the EVF by maximising over goals. The arrows represent the greedy action for each state, and the black dot represents the done action.

Figure 2.5: Showing the extended value function learned for the left task in the Four Rooms domain and the normal value function obtained by maximising over goals. The color map represents the values from lowest (blue) to highest (red).

left rooms. Observe in Figure 2.5a how the learned EVF does indeed encode how to achieve all the goals in the environment. Figure 2.5b shows how maximising over the goals on the EVF recovers the normal value function.

2.2.4 Skill composition

Finally, given the Boolean algebra over tasks satisfying both Assumptions 2.2 and 2.3, and their corresponding skills defined as EVFs, Nangue Tasse *et al.* [2020b] similarly defines a Boolean algebra over skills. They then show that they are homomorphic, meaning that any logical composition of tasks can be solved zero-shot (meaning without further learning) by similarly composing the extended value functions. For example, say we have learned the left task, L, and the top task, T, in the Four Rooms domain shown in Figure 2.5(a). Then we can provably solve their union, intersection, and negation respectively as follows:

$$\mathbf{Q}^*_{L\vee T}(s,g,a) = \max\{\mathbf{Q}^*_L(s,g,a), \mathbf{Q}^*_T(s,g,a)\}$$

$$\mathbf{Q}^*_{L\wedge T}(s,g,a) = \min\{\mathbf{Q}^*_L(s,g,a), \mathbf{Q}^*_T(s,g,a)\}$$

$$\mathbf{Q}^*_{\neg L}(s,g,a) = (\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) - \mathbf{Q}^*_L(s,g,a)$$

where $\mathbf{Q}^*_{SUP}$ is the EVF for the maximum task where all goals are desirable. Similarly $\mathbf{Q}^*_{INF}$ is EVF for the minimum task where all goals are undesirable. With this we also get for free the solution to any combination of unions, intersections, and negations of the left and top tasks. So we can solve more complex compositions like *exclusive or* where the agent needs to go to the left or top rooms excluding the top-left room. We get the skill to immediately solve it by simply composing the basis skills according to the Boolean expression for the *exclusive or*,

$$\begin{aligned}\mathbf{Q}^*_{L\veebar T}(s,g,a) &= \mathbf{Q}^*_{(L\vee T)\wedge\neg(L\wedge T)}(s,g,a)\\ &= \min\{\max\{\mathbf{Q}^*_L(s,g,a),\mathbf{Q}^*_T(s,g,a)\},\\ &\quad [(\mathbf{Q}^*_{SUP}(s,g,a)+\mathbf{Q}^*_{INF}(s,g,a))-\min\{\mathbf{Q}^*_L(s,g,a),\mathbf{Q}^*_T(s,g,a)\}]\}.\end{aligned}$$

These are demonstrated in Figure 2.6.

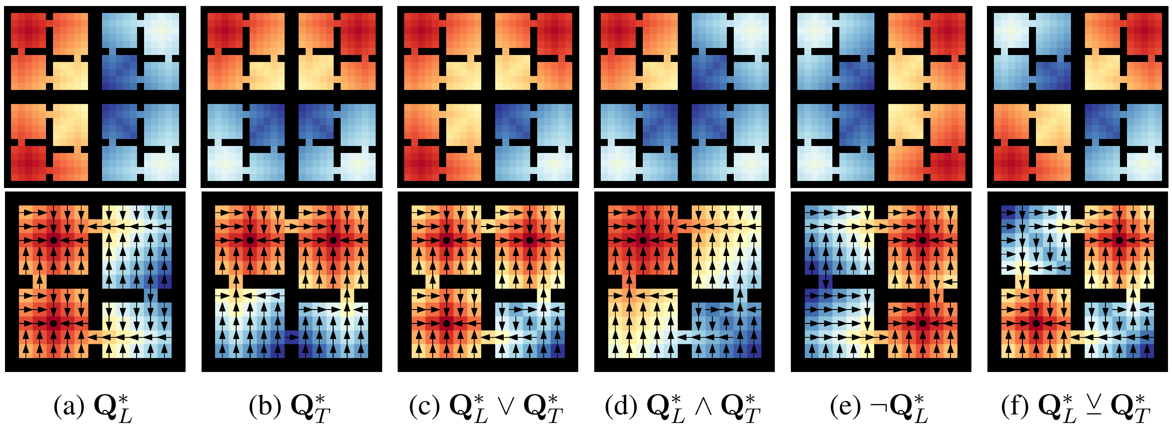


Figure 2.6: An example of Boolean algebraic composition using the learned extended value functions of the left and top tasks. Arrows represent the optimal action in a given state. (a–b) The learned optimal extended value functions for the basis tasks. (c) Zero-shot disjunctive composition. (d) Zero-shot conjunctive composition. (e) Zero-shot negation. (f) Combining operators to model exclusive-or composition.

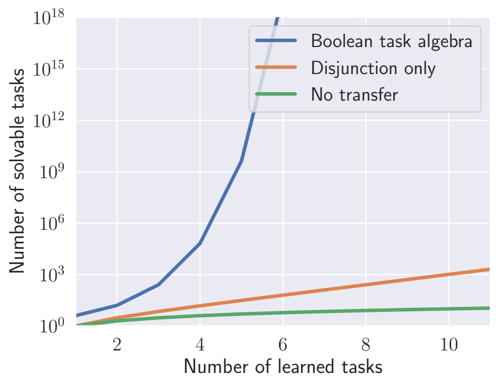
Finally, the notion of basis tasks means that if we learn K of them, it is guaranteed that we can solve 2^{2^K} tasks. This gives us a super-exponential explosion of skills. Figure 2.7 compares it with previous work that could only achieve zero-shot disjunction solving just $2^K - 1$ tasks. For example, with the Boolean algebra after learning 6 basis tasks we can solve about a quintillion tasks, while with only disjunction we can solve just 64 tasks.

2.3 Temporal logic composition

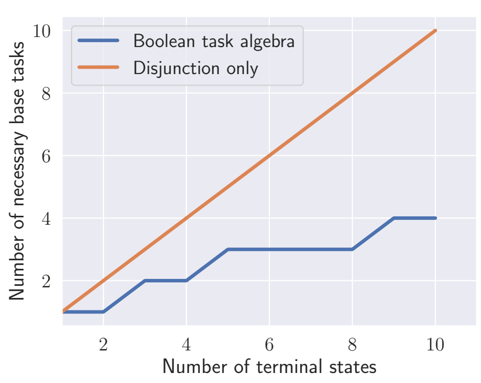
In the previous section, we have described how the logical composition of tasks and their corresponding skills can be obtained through the task algebra framework of Nangue Tasse *et al.* [2020b]. The natural follow-up question is how to obtain the temporal logic compositions of tasks and skills, to understand and solve tasks specified using language. Formal languages and automata theory provide a formal way of specifying such tasks.

2.3.1 Formal languages and automata theory

Formal Languages is a branch of computer science that deals with sets of strings defined over a finite alphabet—a set of propositional symbols—with well-defined rules. Formally, a formal language can be defined by a grammar—the syntactic rules of the language—which can be



(a) Number of tasks that can be solved as a function of the number of learned basis tasks. Results are plotted on a log-scale.



(b) Number of basis tasks that need to be solved to span all tasks as a function of number of terminal states.

Figure 2.7: Comparison of Boolean composition to the disjunctive composition of van Niekerk *et al.* [2019]. (a) The extended value functions allow us to solve exponentially more tasks than the disjunctive approach without further learning. (b) With extended value functions the number of basis tasks required to solve all tasks is logarithmic in the number of achievable goals.

thought of as a subset of the set of all possible strings over a given alphabet [Hopcroft *et al.* 2001]. For instance, programming languages like C++ or Python, and regular expressions are all examples of formal languages. This theory has applications in various fields, such as linguistics, AI, and recently RL [Littman *et al.* 2017; Camacho *et al.* 2019; Vaezipoor *et al.* 2021].

Similarly, automata theory is a branch of computer science that deals with abstract machines that process input strings to produce outputs following predefined rules. More formally, a *state machine*—also called an *automaton*—is an abstract machine that operates on input symbols from a given alphabet [Hopcroft *et al.* 2001]. It transitions between different states based on these inputs according to predefined rules. State machines are used to recognize patterns or sequences within strings and can be classified based on their power or expressiveness. One such classification that is especially popular in AI is the Chomsky Hierarchy [Chomsky 1956].

The Chomsky Hierarchy categorizes formal languages and automata into four types based on their generative power, with each type being more expressive than the previous one:

(i) **Type 0 - Recursively Enumerable Languages (Turing Machines):** These languages can be recognized by Turing machines, which are the most powerful computational model. They include all possible formal languages.

(ii) **Type 1 - Context-Sensitive Languages (Linear Bounded Automata):** Context-sensitive grammars generate these languages. They are more restricted than recursively enumerable languages and can be recognized by linear bounded automata, which have limited memory.

(iii) **Type 2 - Context-Free Languages (Pushdown Automata):** Context-free grammars generate these languages. They are recognized by pushdown automata, which have a stack-based memory structure. Context-free languages are widely used in programming languages and syntax analysis.

(iv) **Type 3 - Regular Languages (Finite State Machine):** Regular grammars generate these languages. They are recognized by finite state machines, which have a finite number of states. Regular languages are simple and commonly used in pattern matching, lexical analysis, and string manipulation.

While this classification is not exhaustive of all types of languages, it highlights how they are all defined by grammars, which in turn are all build upon propositional logics and can all be recognised by state machines. For example, Linear Temporal Logic (LTL) [Pnueli 1977]—and its regular fragments like co-safe LTL [Kupferman and Vardi 2001]—is a particularly popular type of formal language in RL [Konidaris and Barto 2009; Camacho *et al.* 2019; Liu *et al.* 2022]. LTL is used to specify a temporal ordering over events defined by propositional symbols $\mathcal{P}$—in RL, these represent high-level goals in the environment. Formally,

Definition 2.8 (LTL). *An LTL expression is defined using the following recursive syntax:* $\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid X\varphi \mid G\varphi \mid \varphi_1 U \varphi_2 \mid \varphi_1 F \varphi_2$*, where* $p \in \mathcal{P}$*;* $\neg$ *(not),* $\vee$ *(or),* $\wedge$ *(and) are the usual Boolean operators;* X *(neXt),* G *(Globally or always),* U *(Until),* F *(Finally or eventually) are the LTL temporal operators; and* $\varphi, \varphi_1, \varphi_2$ *are any valid LTL expression.*

Consider the office environment for example (Figure 1.1). In this environment, an agent (represented by the little robot head) can move to adjacent cells in any of the cardinal directions ($|\mathcal{A}| = 4$) and observe its (x, y) position ($|\mathcal{S}| = 120$). Cells marked ☕, $\boxtimes$, and 🚹 respectively represent the coffee, mail, and office locations. Those marked ✻ indicate decorations that are broken if the agent collides with them, and A–D indicate the corner rooms. Tasks in this environment can be specified over 10 propositions: $\mathcal{P} = \{A, B, C, D, \text{✻}, \text{☕}, \boxtimes, \text{🚹}, \boxtimes^{+}, \text{🚹}^{+}\}$, where the first 8 propositions are true when the agent is at their respective locations, $\boxtimes^{+}$ is true when the agent is at $\boxtimes$ and there is mail to be collected, and 🚹$^{+}$ is true when the agent is at 🚹 and there is someone in the office. Hence, the task *deliver coffee to the office without breaking decorations* can be specified in LTL as $\left(F\left(\text{☕} \wedge X\left(F\,\text{🚹}\right)\right)\right) \wedge (G\,\neg\text{✻})$. Similarly, the long example given at the beginning of Chapter 1 has LTL specification:

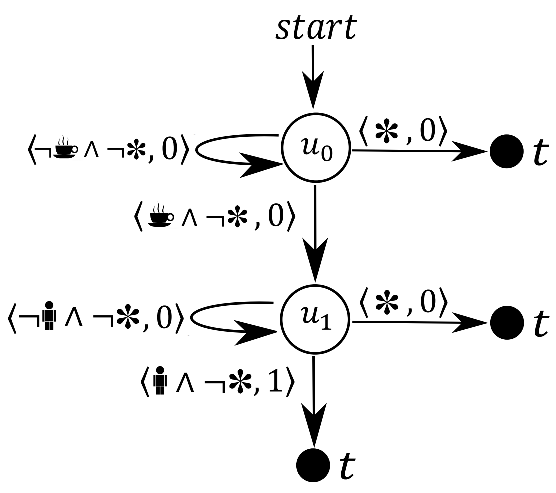
$$\left(F\left(\boxtimes \wedge X\left(F\left(\text{🚹} \wedge X\left(\neg\boxtimes U\left(\neg\boxtimes^{+} \wedge \boxtimes \wedge X\left(F\left(\text{☕} \wedge X\left(\neg\text{🚹} U\left(\neg\text{🚹}^{+} \wedge \text{🚹} \wedge X\right.\right.\right.\right.\right.\right.\right.\right.\right.\right.$$

$$(FA \wedge X(F(B \wedge X(F(C \wedge X(F(D \wedge X(FA)))))))))))))))))) \wedge (G\,\neg\text{✻})$$

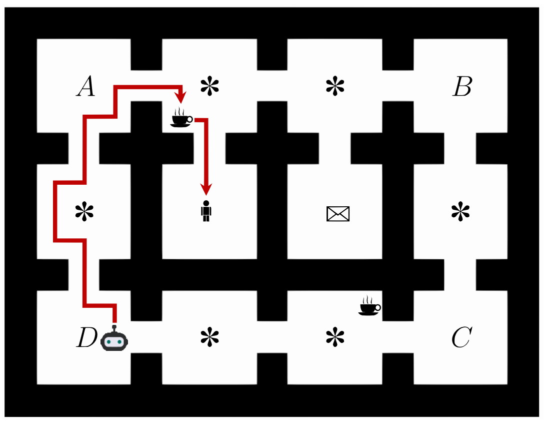
While formal languages like LTL are a natural way to specify tasks, an important question is what rewards they correspond to, since RL agents only understand the task to be solved as the maximisation of rewards. A popular approach to address is question is *reward machines*.

2.3.2 Reward machines

One difficulty with the standard MDP formulation is that the agent is often required to solve a complex long-horizon task using only a scalar reward signal as feedback from which to learn. To overcome this, a common approach is to use reward machines (RM) [Icarte *et al.* 2018], which provide structured feedback to the agent in the form of a state machine. In principle, since any formal language can be converted to a state machine, each can also be converted to a reward machine. For example, Camacho *et al.* [2019] show that temporal logic tasks specified using regular languages, such as regular fragments of LTL (like safe, co-safe, and finite trace LTL), can be converted to finite state machines (FSM) with rewards of 1 for accepting transitions and



(a) Reward Machine



(b) Trajectory of an optimal policy

Figure 2.8: Illustration of (a) the reward machine for the task *deliver coffee to the office without breaking decorations*, given by the LTL specification $\big(F\,(☕ \wedge X\,(F\,🧍))\big) \wedge (G\,\neg ✻)$; (b) the office gridworld where the blue circle represents the agent. The reward machine is obtained by converting the LTL expression into an FSM using Spot [Duret-Lutz *et al.* 2016], and then giving a reward of 1 for accepting transitions and 0 otherwise. Nodes labeled t represent terminal states.

0 otherwise.[2] These RMs encode the task to be solved using a set of propositional symbols $\mathcal{P}$ that represent high-level environment features as follows:

Definition 2.9 (RM)**.** *Given a set of environment states $\mathcal{S}$ and actions $\mathcal{A}$, a reward machine is a tuple $R_{\mathcal{SA}} = \langle \mathcal{U}, u_0, \delta_u, \delta_r \rangle$ where (i) $\mathcal{U}$ is a finite set of states; (ii) $u_0 \in \mathcal{U}$ is the initial state; (iii) $\delta_u : \mathcal{U} \times 2^{\mathcal{P}} \to \mathcal{U}$ is the state-transition function; and (iv) $\delta_r : \mathcal{U} \times 2^{\mathcal{P}} \to \{0, 1\}$ is the state-reward function.*[3]

Figure 2.8 shows an example. To incorporate RMs into the RL framework, the agent must be able to determine a correspondence between abstract RM propositions and states in the environment. To achieve this, the agent is equipped with a labelling function $L : \mathcal{S} \to 2^{\mathcal{P}}$ that assigns truth values to each state the agent visits in its environment. The agent's aim now is to learn a policy $\pi : \mathcal{S} \times \mathcal{U} \to \mathcal{A}$ that maximises the rewards from an RM while acting in an environment $\langle \mathcal{S}, \mathcal{A}, P, \gamma, \mathcal{P}, L \rangle$. However, the rewards from the reward machine are not necessarily Markov with respect to the environment. Icarte *et al.* [2022] shows that a **product MDP** (Definition 2.10 below) between the environment and a reward machine guarantees that the rewards are Markov such that the policy can be learned with standard algorithms such as Q-learning. This is because the product MDP uses the cross-product to consolidate how actions in the environment result in simultaneous transitions in the environment and state machine. Thus, product MDPs take the

[2]Accepting transitions are those at which the high-level task—described, for example, by LTL—is satisfied. Additionally, this type of state machine that defines outputs for each machine transition is called a Mealy machine. When the outputs are instead defined per machine state, it is called a Moore machine. For simplicity, we will keep referring to them more generally as FSMs.

[3]RMs are more general, but for clarity, we focus on the subset that is obtained from regular languages.

form of standard, learnable MDPs. In the rest of this work, we will refer to these product MDPs as *tasks*.

Definition 2.10 (Tasks). *Let $\langle \mathcal{S}, \mathcal{A}, P, \gamma, \mathcal{P}, L \rangle$ represent the environment and $\langle \mathcal{U}, u_0, \delta_u, \delta_r \rangle$ be an RM representing the task rewards. Then a task is a product MDP $M_\mathcal{T} = \langle \mathcal{S}_\mathcal{T}, \mathcal{A}, P_\mathcal{T}, R_\mathcal{T}, \gamma \rangle$ between the environment and the RM, where $\mathcal{S}_\mathcal{T} := \mathcal{S} \times \mathcal{U}$, $R_\mathcal{T}(\langle s, u \rangle, a, \langle s', u' \rangle) := \delta_r(u, l')$, $P_\mathcal{T}(\langle s, u \rangle, a) := \langle s', u' \rangle$, $s' \sim P(\cdot|s, a)$, $u' = \delta_u(u, l')$, and $l' = L(s')$.*

2.4 Conclusion

In this chapter, we have explored the foundational areas of interest to us for designing general-purpose agents capable of solving diverse tasks in their environment. Beginning with reinforcement learning (RL), we discussed how agents make sequential decisions to maximise the cumulative sum of rewards for given tasks in their environment. We then delved into the logical composition framework of Nangue Tasse *et al.* [2020b], which employs lattice theory to formalise the logical combination of tasks and skills—defined by a new type of goal-oriented value function that enables zero-shot composition. Importantly, we showed the main restrictions that Nangue Tasse *et al.* [2020b] imposes on the task space, which limits the instructability and flexibility of agents. To address these limitations, we will extend their logical composition framework to more general task spaces (Part I) and skills (Part II) Finally, we examined temporal logic composition, which utilises formal languages and reward machines to specify complex temporal tasks. In part III, we will show how logical composition can be leveraged to solve such tasks safely, and also solve natural language specified tasks.

Part I

Instructable Agents

In Chapter 3, we formally define the logical composition of tasks and explore the semantic meaning of such compositions. Given that we are now able to specify tasks as a composition of other tasks, Chapter 4 ensures that the rewards of tasks in general minimise the probability of leading to optimal but undesired behaviours.

Chapter 3

Logical composition of tasks

Before we can construct agents capable of solving new compositional tasks, we must formally define these composite tasks. Since reinforcement learning models tasks as MDPs, we require principled ways of composing these MDPs to produce new MDPs that model a desired task specification. Similarly to Nangue Tasse *et al.* [2020b], this is done by leveraging the structure of lattice algebras since, as outlined in Section 2.2.1, they abstract the notion of disjunction, conjunction, and negation. Hence, we make the following main contributions in this chapter:

(i) **Conjunction and disjunction of tasks (Section 3.1):** Since Boolean algebra is the most restrictive structure, we will first start with a general lattice defined over general tasks—relaxing Assumption 2.1. This extends the conjunction and disjunction operators of Nangue Tasse *et al.* [2020b] to arbitrary tasks in potentially stochastic environments.

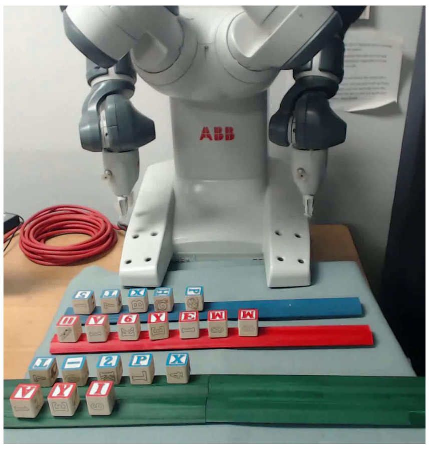
(ii) **Negation of tasks (Section 3.2):** We will then increasingly constrain the task space with necessary assumptions on the road to a Boolean algebra. In particular, we will show that by simply bounding the task space, we can obtain a De Morgan algebra over tasks. This extends the negation operator of Nangue Tasse *et al.* [2020b] to tasks with arbitrary bounded rewards in potentially stochastic environments.

(iii) **Propositional logics over tasks (Section 3.3):** Interestingly, to achieve this, we show that task rewards do not have to be restricted by Assumption 2.2—for example, reward of 1 at desirable goal states and zero rewards everywhere else. Instead, it is sufficient for the rewards for each transition to be the same as that of the task bounds—which can be arbitrary. This enables different semantics of logics from the choice of tasks bounds.

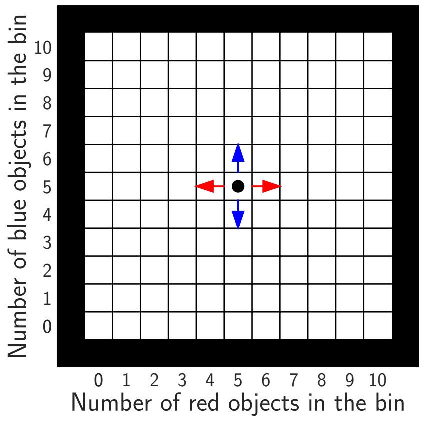
During this process, we will use the following running example to illustrate some of the different types of semantics of task compositions that result from the different lattice structures:

Example 3.1 (Bin-packing domain)**.** *Consider an environment where an agent needs to manipulate a robotic arm to pack (or unpack) objects into (from) a green bin (Figure 3.1). There are 10 red objects and 10 blue objects in the domain, resulting in* 11×11 *states where each state* $s = (r, b)$ *corresponds to the number of red* (r) *and blue* (b) *objects in the bin.*

The agent has actions to command the robotic arm to put a red object into the bin ($\rightarrow$)*, remove a red object from the bin* ($\leftarrow$)*, put a blue object into the bin* ($\uparrow$)*, and remove a blue object from the bin* ($\downarrow$)*. If there are no red or blue objects to put into the bin or remove for the respective* *actions, the robot arm does nothing. The agent also has a fifth action* (•) *for "done" that it chooses to terminate; a state only becomes terminal if the agent chooses the done action in it.*

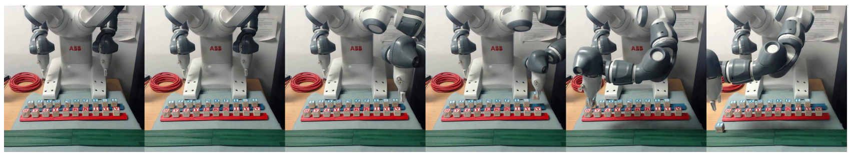


(a) Bin packing domain

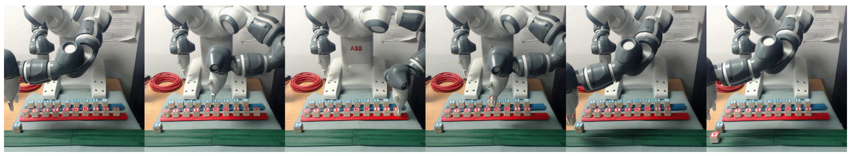

Number of blue objects in the bin
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10
Number of red objects in the bin

(b) Gridworld representation

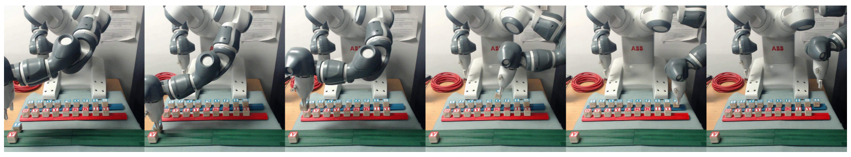
Figure 3.1: Gridworld representation of the simple bin packing domain.



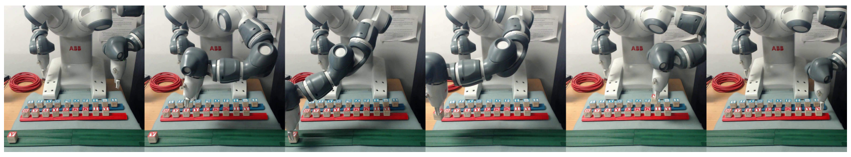
(a) ↑ action



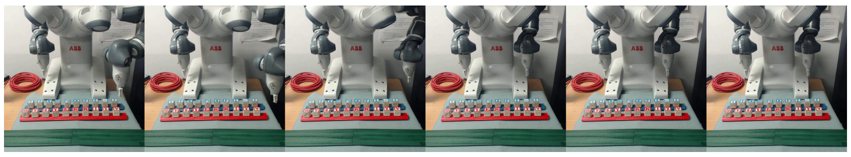
(b) → action



(c) ↓ action

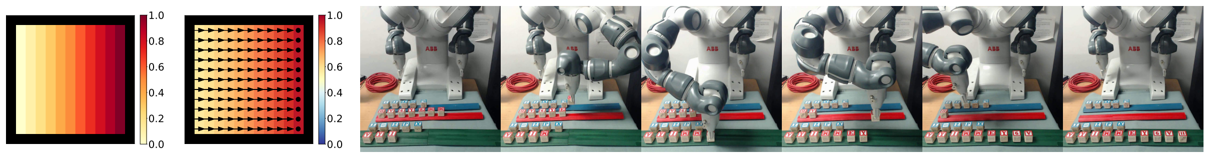


(d) ← action



(e) • action

Figure 3.2: Sample robot arm trajectories for each action in the bin packing domain.



(a) R (b) Q^* (c) Trajectory starting from state $(3, 5)$

Figure 3.3: Terminal rewards (heat map in (a)), optimal policy (arrows in (b)) and value functions (heat map in (b)), and resulting robot arm trajectories. The optimal policies and value functions are obtained using Q-learning.

Figure 3.1b illustrates the gridworld representation of this domain, and Figure 3.2 shows the effect of each action in it.

Consider a task in which the robot must pack all the red objects into the bin. The non-terminal rewards (rewards for all non-terminal transitions) are $R_{MIN} = 0$ and the terminal rewards range from $R_{MIN} = 0$ to $R_{MAX} = 1$. The discount factor used is $\gamma = 0.95$. That is, an agent receives a reward of R_{MIN} as it acts in the environment but when it chooses the done action at any state, it receives a reward between R_{MIN} and R_{MAX} depending on how close it is to the desired states. Figure 3.3 shows the terminal rewards, optimal policy, and trajectory of the robot packing red objects into the bin.

3.1 Task lattice

Having described how a lattice algebra abstracts the usual concept of disjunction and conjunction, we now formalise the meaning of disjunction and conjunction of tasks. Consider the set of all tasks in an environment:

$$\mathcal{M}_{\mathbb{R}} := \{M = (\mathcal{S}, \mathcal{A}, P, R_M, \gamma) | R_M(s, a, s') \in \mathbb{R}\}$$

Since we have constrained ourselves to the model-free setting, we require all our definitions to be usable even when the only information available to the agent from the MDP is its current state s, the next state $s' \sim P(\cdot|s, a)$ and reward $R(s, a, s')$ after taking an action a. Given that tasks differ only in their reward functions (Definition 2.1), we can achieve this by first defining the partial order over tasks using pointwise $\leq$ (the usual $\leq$ relation on $\mathbb{R}$) over the rewards. The resulting partially ordered set of tasks is formally stated as follows:

Proposition 3.1. *Let $M_1, M_2 \in \mathcal{M}_{\mathbb{R}}$ be tasks with reward functions R_{M_1} and R_{M_2} respectively. Then $(\mathcal{M}_{\mathbb{R}}, \leq)$ is a partially ordered set with the relation $\leq$ given by*

$$M_1 \leq M_2 \textit{ if } R_{M_1}(s, a, s') \leq R_{M_2}(s, a, s') \textit{ for all } (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}.$$

Proof. Follows from the usual $\leq$ relation on $\mathbb{R}$. □

Since the reward functions are real-valued, every pair of rewards has a least upper bound (sup) and a greatest lower bound (inf). The resulting real functions after pointwise inf and sup are then clearly still valid task reward functions. Hence the partially-ordered set of tasks $(\mathcal{M}_{\mathbb{R}}, \leq)$ has a least upper bound $\sup\{M_1, M_2\} \in \mathcal{M}_{\mathbb{R}}$ and a greatest lower bound $\inf\{M_1, M_2\} \in \mathcal{M}_{\mathbb{R}}$ for any pair of task $M_1, M_2 \in \mathcal{M}_{\mathbb{R}}$ (since they only differ on their rewards). The lattice $(\mathcal{M}_{\mathbb{R}}, \vee, \wedge)$ induced by this partial order trivially follows with the binary operators $\vee$ and $\wedge$ given by $M_1 \vee M_2 := \sup\{M_1, M_2\} \in \mathcal{M}_{\mathbb{R}}$ and $M_1 \wedge M_2 := \inf\{M_1, M_2\} \in \mathcal{M}_{\mathbb{R}}$. We define these operators formally as follows:

Definition 3.1. *The join* $\vee : \mathcal{M}_{\mathbb{R}} \times \mathcal{M}_{\mathbb{R}} \to \mathcal{M}_{\mathbb{R}}$ *and meet* $\wedge : \mathcal{M}_{\mathbb{R}} \times \mathcal{M}_{\mathbb{R}} \to \mathcal{M}_{\mathbb{R}}$ *operators over tasks are given by*

$$\vee(M_1, M_2) := (\mathcal{S}, \mathcal{A}, P, R_{M_1 \vee M_2}, \gamma), \textit{ where } R_{M_1 \vee M_2} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$$
$$(\cdot) \mapsto \sup\{R_{M_1}(\cdot), R_{M_2}(\cdot)\},$$

$$\wedge(M_1, M_2) := (\mathcal{S}, \mathcal{A}, P, R_{M_1 \wedge M_2}, \gamma), \textit{ where } R_{M_1 \wedge M_2} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$$
$$(\cdot) \mapsto \inf\{R_{M_1}(\cdot), R_{M_2}(\cdot)\}.$$

In fact, $(\mathcal{M}_{\mathbb{R}}, \vee, \wedge)$ forms a *distributive lattice* since inf and sup with the usual $\leq$ in $\mathbb{R}$ is distributive:

Proposition 3.2. $(\mathcal{M}_{\mathbb{R}}, \vee, \wedge)$ *is a distributive lattice.*

Proof. Follows from the properties of inf and sup and their distributivity. □

This allows us to jointly apply disjunction and conjuction operators to sets of tasks. Given a non-empty finite set $\mathcal{O}$ of lower bounded subsets of tasks $\mathcal{N} \subset \mathcal{M}_{\mathbb{R}}$, the task lattice $(\mathcal{M}_{\mathbb{R}}, \vee, \wedge)$ gives us the principled way of specifying the disjunction of conjunctions:

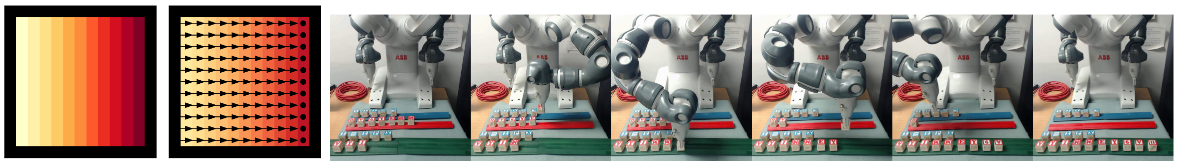
$$\bigvee_{\mathcal{N} \in \mathcal{O}} \left(\bigwedge_{N \in \mathcal{N}} N \right) = (\mathcal{S}, \mathcal{A}, P, R_{\underset{\mathcal{O}}{\vee}\underset{\mathcal{N}}{\wedge}}, \gamma), \text{ where } R_{\underset{\mathcal{O}}{\vee}\underset{\mathcal{N}}{\wedge}}(s, a, s') := \sup_{\mathcal{N} \in \mathcal{O}} \left(\inf_{N \in \mathcal{N}} R_N(s, a, s') \right).$$

Similarly, given a non-empty finite set $\mathcal{O}$ of upper bounded subsets of tasks $\mathcal{N} \subset \mathcal{M}_{\mathbb{R}}$, the conjunction of disjunctions is given by,

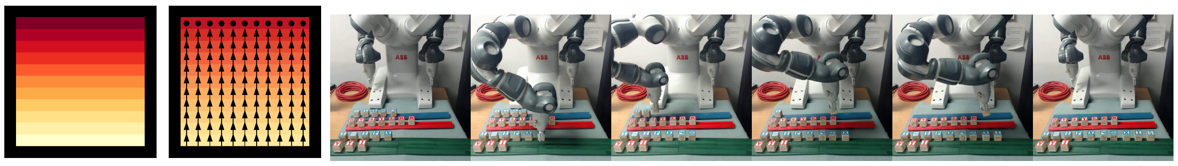
$$\bigwedge_{\mathcal{N} \in \mathcal{O}} \left(\bigvee_{N \in \mathcal{N}} N \right) = (\mathcal{S}, \mathcal{A}, P, R_{\underset{\mathcal{O}}{\wedge}\underset{\mathcal{N}}{\vee}}, \gamma), \text{ where } R_{\underset{\mathcal{O}}{\wedge}\underset{\mathcal{N}}{\vee}} : (s, a, s') := \inf_{\mathcal{N} \in \mathcal{O}} \left(\sup_{N \in \mathcal{N}} R_N(s, a, s') \right).$$

Example 3.2. *Consider the bin packing domain introduced in Example 3.1. Further consider the specification of two tasks, 🟥 and 🟦, in which the robot must pack all the red and blue objects into the bin respectively. Figure 3.4 shows the terminal rewards (all non-terminal rewards are 0), optimal policy and value function of the task specified by their disjunction 🟥 $\vee$ 🟦 and conjunction 🟥 $\wedge$ 🟦. We can observe how the* sup *and* inf *of task rewards indeed result in composite task rewards with the correct semantics for the disjunctive and conjunctive tasks respectively. To learn the corresponding optimal policies and value functions, we use Q-learning where the transition rewards are obtained by doing* $\sup\{R_{🟥}(s, a, s'), R_{🟦}(s, a, s')\}$ *for their disjunction and* $\inf\{R_{🟥}(s, a, s'), R_{🟦}(s, a, s')\}$ *for their conjunction.*

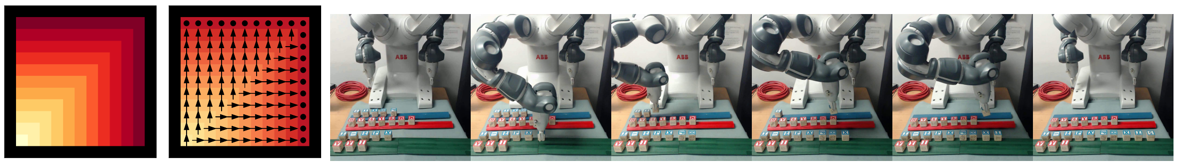
Finally, we can also use a Hasse diagram to visualise the task sub-lattice generated by all combinations of disjunction and conjunction of 🟥 and 🟦 (Figure 3.5).



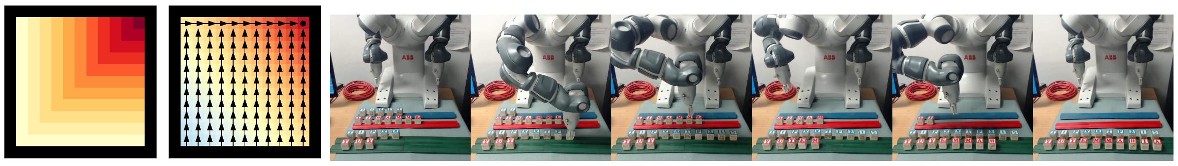
(a) Pack all red objects into the bin: 🟥



(b) Pack all blue objects into the bin: 🟦



(c) Pack all red or blue objects into the bin: 🟥 $\vee$ 🟦



(d) Pack all red and blue objects into the bin: 🟥 $\wedge$ 🟦

Figure 3.4: Terminal rewards (left column heat maps), optimal value functions (middle column heat maps), optimal policies (middle column arrows), and sample robot trajectories (right column images) for the disjunction (c) and conjunction (d) of tasks (a-b) in the bin packing domain. The optimal policies and value functions are obtained using Q-learning.

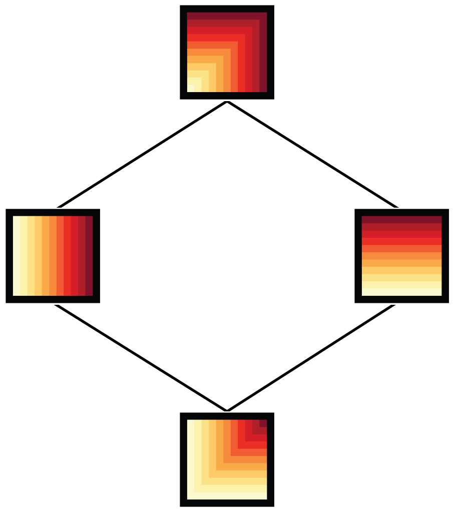


Figure 3.5: Hasse diagram of the task sub-lattice with basis {🟥, 🟦}.

3.2 De Morgan task algebra

Having formalised the meaning of task disjunction and conjunction, we next turn our attention to the negation of tasks. As discussed in Section 2.2.1, the De Morgan algebra allows us to define this operator by adding the minimal required properties that encapsulate the desired semantics of a negation. In particular, we only need the set of tasks to be bounded by some tasks $\mathcal{M}_{INF}, \mathcal{M}_{SUP} \in \mathcal{M}_{\mathbb{R}}$:

$$\mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]} := \{M = (\mathcal{S}, \mathcal{A}, P, R_M, \gamma) \mid R_M(s,a,s') \in [R_{INF}(s,a,s'), R_{SUP}(s,a,s')]\}$$

where R_{INF} and R_{SUP} are the reward functions of $\mathcal{M}_{INF}$ and $\mathcal{M}_{SUP}$ respectively. We can now define the negation of a task as follows:

Definition 3.2. *Define the negation operator* $\neg : \mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]} \to \mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]}$ *as*
$\neg(M) := (\mathcal{S}, \mathcal{A}, P, R_{\neg M}, \gamma)$, *where* $R_{\neg M} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$
$$(\cdot) \mapsto (R_{SUP}(\cdot) + R_{INF}(\cdot)) - R_M(\cdot).$$

The above definition captures the intuition behind negation. For example, if an agent takes an action at a given state and receives the smallest reward for the resulting transition, then the agent acting in the opposite task should receive the highest reward for that same transition. Note how for $M \in \mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]}$, $(\mathcal{S}, \mathcal{A}, P, R_{\neg M}, \gamma) \in \mathcal{M}$ because $R_{\neg M}(s,a,s')$ is also bounded by $[R_{INF}(s,a,s'), R_{SUP}(s,a,s')]$. Hence, $\neg$ is closed in $\mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]}$.

Finally, we formalise the interaction of the negation of tasks with the conjunction and disjunction of tasks as follows:

Proposition 3.3. $(\mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *is a De Morgan algebra.*

Proof. Let $M_1, M_2 \in \mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]}$. We show that $\neg, \vee, \wedge$ satisfy the De Morgan algebra axioms.

(i)–(v): These follow from the properties of $\inf$ and $\sup$.

(vi): This follows from the bounds $\mathcal{M}_{SUP}, \mathcal{M}_{INF} \in \mathcal{M}_{[\mathcal{M}_{INF}\ \mathcal{M}_{SUP}]}$ which are guaranteed to exist by definition.

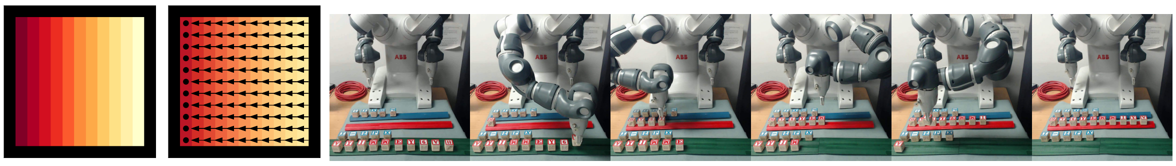
(vii): The first condition easily follows from the definition of $\neg$. For the second condition, let $R_{\neg(M_1 \vee M_2)}$ be the reward function for $\neg(M_1 \vee M_2)$. Then for all (s,a,s') in $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$,

$$\begin{aligned}
R_{\neg(M_1 \vee M_2)}(s,a,s') &= (R_{SUP}(s,a,s') + R_{INF}(s,a,s')) - \sup_{M \in \{M_1, M_2\}} R_M(s,a,s') \\
&= (R_{SUP}(s,a,s') + R_{INF}(s,a,s')) + \inf_{M \in \{M_1, M_2\}} -R_M(s,a,s') \\
&= \inf_{M \in \{M_1, M_2\}} (R_{SUP}(s,a,s') + R_{INF}(s,a,s')) - R_M(s,a,s') \\
&= R_{\neg M_1 \wedge \neg M_2}(s,a,s').
\end{aligned}$$

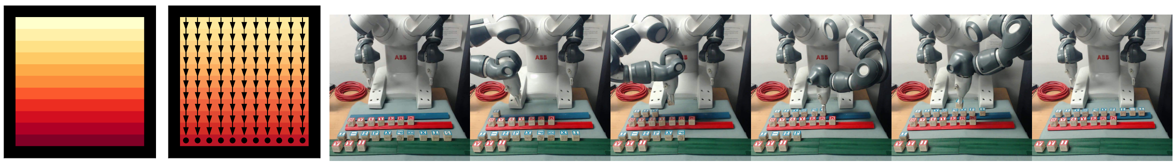
Thus $\neg(M_1 \vee M_2) = \neg M_1 \wedge \neg M_2$.

□

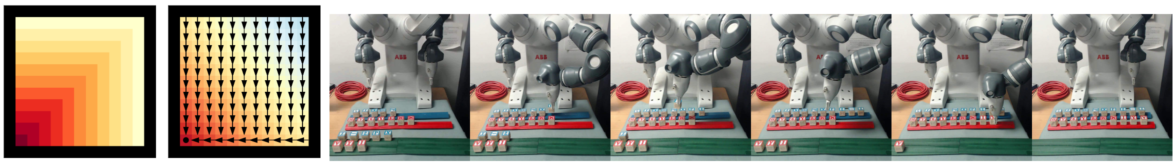
We can now specify arbitrary disjunction, conjunction, and negation of tasks.



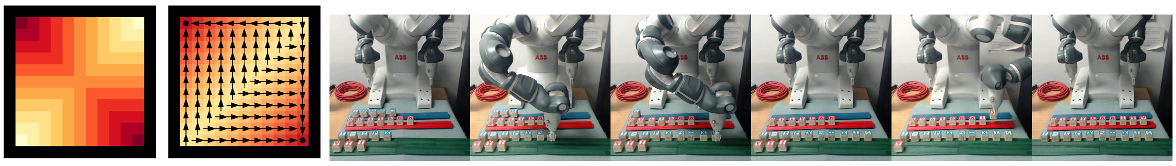
(a) Unpack all red objects from the bin: ¬🟥



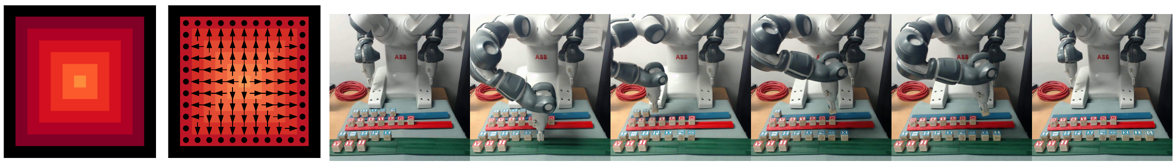
(b) Unpack all blue objects from the bin: ¬🟦



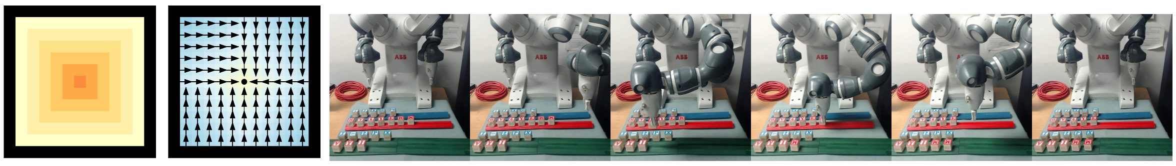
(c) Unpack all objects from the bin: 🟥 $\bar{\vee}$ 🟦 $:=$ ¬(🟥 ∨ 🟦)



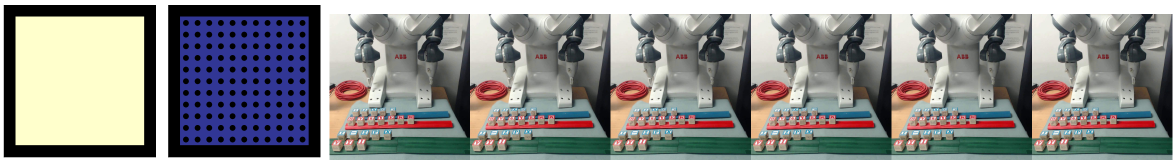
(d) Pack only red or only blue objects into the bin: 🟥 $\veebar$ 🟦 $:=$ (🟥 ∨ 🟦) ∧ ¬(🟥 ∧ 🟦)



(e) Pack or unpack all red or blue objects into (from) the bin: (🟥 ∨ ¬🟥) ∨ (🟦 ∨ ¬🟦)



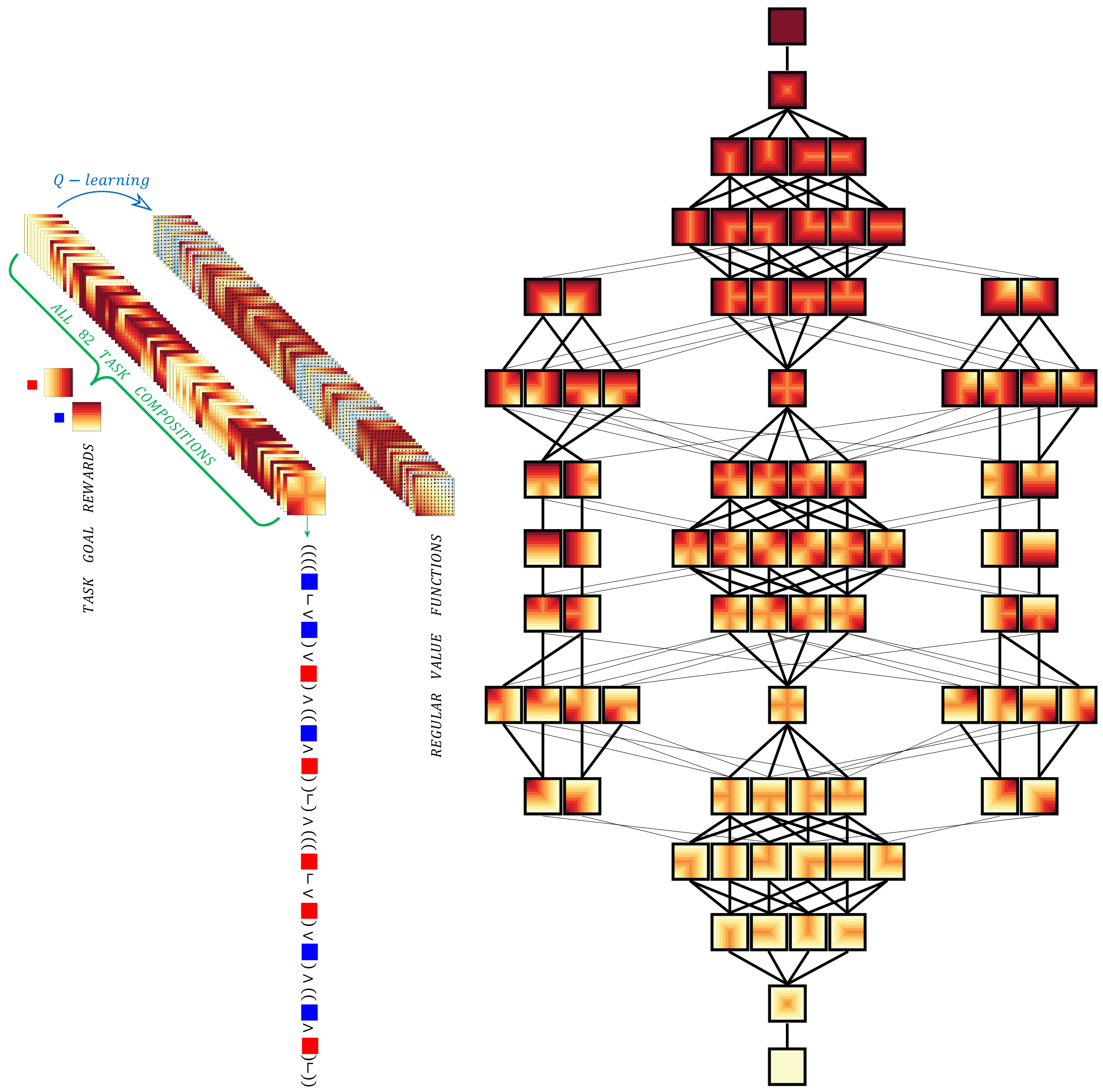
(f) Pack exactly half of the red and blue objects into the bin: (🟥 ∧ ¬🟥) ∧ (🟦 ∧ ¬🟦)



(g) Do nothing: $\mathcal{M}_{INF}$

Figure 3.6: Terminal rewards (left column heat maps), optimal value functions (middle column heat maps), optimal policies (middle column arrows), and sample robot trajectories (right column images) for the lower bound task $\mathcal{M}_{INF}$ and compositions of tasks (🟥, 🟦) in the bin packing domain. Optimal policies and value functions are obtained using Q-learning.

Example 3.3. *In the bin packing environment, consider the De Morgan lattice bounded by the tasks where all non-terminal rewards are 0, and all terminal rewards are 0 (🟨) for the lower bound and 1 (🟫) for the upper bound. Further consider the specification of two tasks, 🟥 and 🟦, in which the robot must pack all the red and blue objects into the bin respectively (the same ones from Example 3.2). Figure 3.6 shows the rewards, optimal policies, optimal value functions, and resulting robot trajectories for sample task compositions. We can observe how the negation of tasks, and its interaction with the disjunction and conjunction of tasks, results in composite tasks with desired semantics. For example, the negations ¬🟥 and ¬🟦 specify the tasks in which the robot must remove all red objects and all blue objects from the bin respectively. Figure 3.7 shows the De Morgan sub-lattice generated by all combinations of disjunction, conjunction, and negation of 🟥 and 🟦.*



(a) All task compositions of 🟥 and 🟦

(b) Hasse diagram

Figure 3.7: Illustration of the De Morgan sub-lattice with basis {🟥, 🟦}.

Finally, notice that the task bounds are what define the semantic meaning of the negation operator, with different task bounds leading to different semantics. For example, when using the task bounds {□, ■}*, the negation of the task in which the robot must pack or unpack all red objects into (from) the bin* (🟥) *is to pack or unpack exactly half of the red objects into (from) the bin* (¬🟥 = 🟨)*. However, when using the task bounds* {⊞, ⊠}*, the negation is instead to pack or unpack all the blue objects into (from) the bin* (¬🟥 = ▤).

3.3 Boolean task algebra

While the De Morgan task algebra allows for logical composition of tasks with arbitrary bounded rewards, it provides no guarantees on certain desired properties. In particular, these task compositions do not always satisfy the laws of the excluded middle ($M_1 \vee \neg M_1 = \mathcal{M}_{SUP}$), and of non-contradiction ($M_1 \wedge \neg M_1 = \mathcal{M}_{INF}$). This can clearly be seen in Figure 3.6f, where the agent needs to *pack all red objects into the bin and remove all red objects from the bin* and *pack all blue objects into the bin and remove all blue objects from the bin*. In this case, the choice of rewards produces a meaningful task—*pack exactly half of the red and blue objects into the bin*—but in general, we may want to guarantee that contradicting task specifications are meaningless. To achieve this, we need to restrict the set of tasks to those with *binary rewards*—binary here means the rewards are either $R_{INF}(s,a,s')$ or $R_{SUP}(s,a,s')$—which ensures that tasks have a Boolean nature:

$$\mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}} := \{M = (\mathcal{S},\mathcal{A},P,R_M,\gamma) \mid R_M(s,a,s') \in \{R_{INF}(s,a,s'), R_{SUP}(s,a,s')\}\}$$

We can now formalise a Boolean logic on the set of tasks.

Proposition 3.4. $(\mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *is a Boolean algebra.*

Proof. Let $M_1, M_2 \in \mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}}$. We show that $\neg, \vee, \wedge$ satisfy the Boolean algebra axioms.

(i)–(vi): These follow from the De Morgan task algebra since $\mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}}$ satisfies its assumptions.

(vii): Let $R_{M_1 \wedge \neg M_1}$ be the reward function for $M_1 \wedge \neg M_1$. Then for all (s,a,s') in $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$,

$$\begin{aligned}
R_{M_1 \wedge \neg M_1}(s,a,s') &= \inf\{R_{M_1}(s,a,s'), \\
&\qquad (R_{SUP}(s,a,s') + R_{INF}(s,a,s')) - R_{M_1}(s,a,s')\} \\
&= \begin{cases} R_{INF}(s,a,s'), & \text{if } R_{M_1}(s,a,s') = R_{SUP}(s,a,s') \\ R_{INF}(s,a,s'), & \text{if } R_{M_1}(s,a,s') = R_{INF}(s,a,s') \end{cases} \\
&= R_{INF}(s,a,s').
\end{aligned}$$

Thus $M_1 \wedge \neg M_1 = \mathcal{M}_{INF}$, and similarly $M_1 \vee \neg M_1 = \mathcal{M}_{SUP}$.

□

Having established a Boolean algebra over tasks, we show that there exists an equivalence between the task algebra and a power set algebra. We note that the assumption of binary rewards establishes a bijection F between the set of tasks $\mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}}$ and the power-set $2^{\mathcal{S}\times\mathcal{A}\times\mathcal{S}}$, given by:

$$
\begin{aligned}
F : 2^{\mathcal{S}\times\mathcal{A}\times\mathcal{S}} &\to \mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}} \\
\mathcal{H} &\mapsto (\mathcal{S},\mathcal{A},P,R_{\mathcal{H}},\gamma), \text{ where } R_{\mathcal{H}} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R} \\
&\qquad (s,a,s') \mapsto \begin{cases} R_{SUP}(s,a,s'), & \text{if } (s,a,s') \in \mathcal{H} \\ R_{INF}(s,a,s'), & \text{if } (s,a,s') \notin \mathcal{H}. \end{cases}
\end{aligned}
$$

The Boolean task algebra together with the bijection between tasks $\mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}}$ and the power-set $2^{\mathcal{S}\times\mathcal{A}\times\mathcal{S}}$ gives us the following result:

Proposition 3.5. *The Boolean task algebra on $\mathcal{M}_{\{\mathcal{M}_{INF},\mathcal{M}_{SUP}\}}$ is isomorphic to the power set Boolean algebra on $2^{\mathcal{S}\times\mathcal{A}\times\mathcal{S}}$.*

Proof. This follows from the bijection F and the fact that it is clearly homomorphic. □

Consequently, all results that hold for power set Boolean algebras now also hold for Boolean task algebras. In particular, consider the Boolean algebra on a set of tasks $\mathcal{M}$ with rewards that are binary and depend only on the current state (such as Example 3.4). Then the respective Boolean task sub-algebra is isomorphic to the power set algebra on $\mathcal{S}$ with the isomorphism F given by:

$$
\begin{aligned}
F : 2^{\mathcal{S}} &\to \mathcal{M} \\
\mathcal{H} &\mapsto (\mathcal{S},\mathcal{A},P,R_{\mathcal{H}},\gamma), \text{ where } R_{\mathcal{H}} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R} \\
&\qquad (s,a,s') \mapsto \begin{cases} R_{SUP}(s,a,s'), & \text{if } s \in \mathcal{H} \\ R_{INF}(s,a,s'), & \text{if } s \notin \mathcal{H}. \end{cases}
\end{aligned}
$$

This means that for $\mathcal{M}$ with finite $\mathcal{S}$, we need only a logarithmic number of basis tasks (minimal generators) $\lceil \log_2 |\mathcal{S}| \rceil$ (for $|\mathcal{S}| > 1$) to specify an exponential number of composed tasks $|\mathcal{M}| = 2^{|\mathcal{S}|}$.

Finally, we summarise the main differences between the developed task algebras in Table 3.1. While the Boolean task algebra necessitates the most assumptions, it is also the most powerful. In the next section, we will show that an additional benefit of Boolean task algebra is its basis can be obtained directly from the task bounds.

	Assumptions			Benefits		
	Same environment	Same reward bounds	Binary rewards	Union and intersection	Negation	Full Boolean logic
Task lattice	✓			✓		
De Morgan task algebra	✓	✓		✓	✓	
Boolean task algebra	✓	✓	✓	✓	✓	✓

Table 3.1: Trade-offs between the necessary assumptions and benefits of each lattice structure.

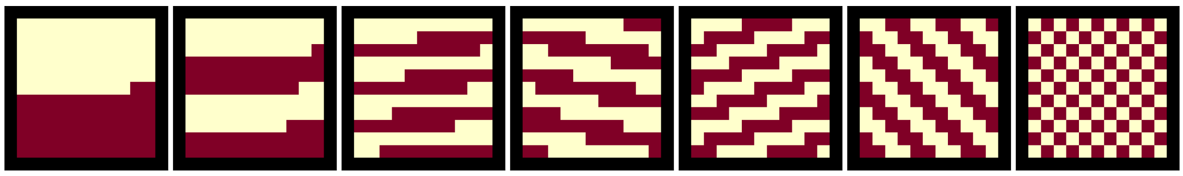
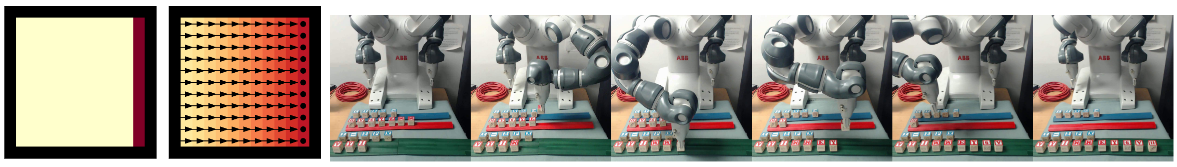
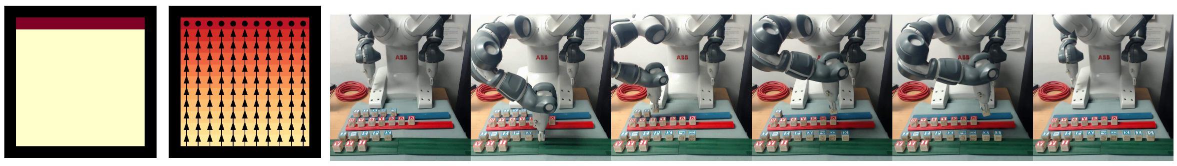


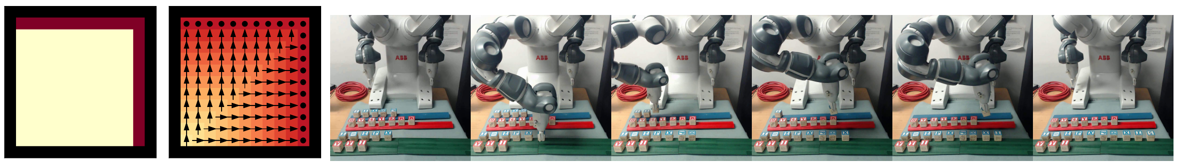
Figure 3.8: Basis (minimal generators) for the Boolean task algebra bounded by {⬜, 🟥}



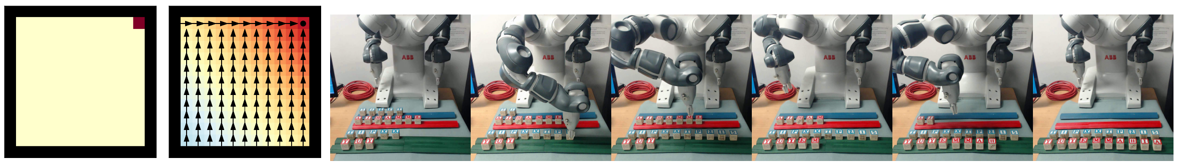
(a) Pack all red objects into the bin: 🟥



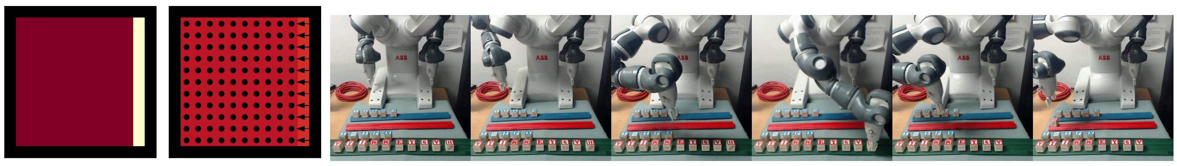
(b) Pack all blue objects into the bin: 🟦



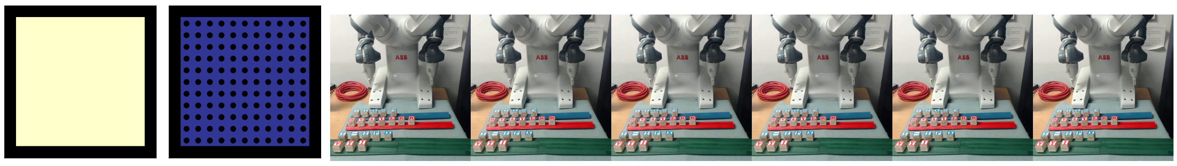
(c) Pack all red or blue objects into the bin: 🟥 $\vee$ 🟦



(d) Pack all red and blue objects into the bin: 🟥 $\wedge$ 🟦



(e) Do not pack all red objects into the bin: $\neg$🟥



(f) Pack and do not pack all red and blue objects into the bin: (🟥 $\wedge$ $\neg$🟥) $\wedge$ (🟦 $\wedge$ $\neg$🟦)

Figure 3.9: Terminal rewards (left), optimal values and policies (middle) and sample robot trajectories (right) for the compositions of Boolean tasks in the bin packing domain. Optimal policies and value functions are obtained using Q-learning.

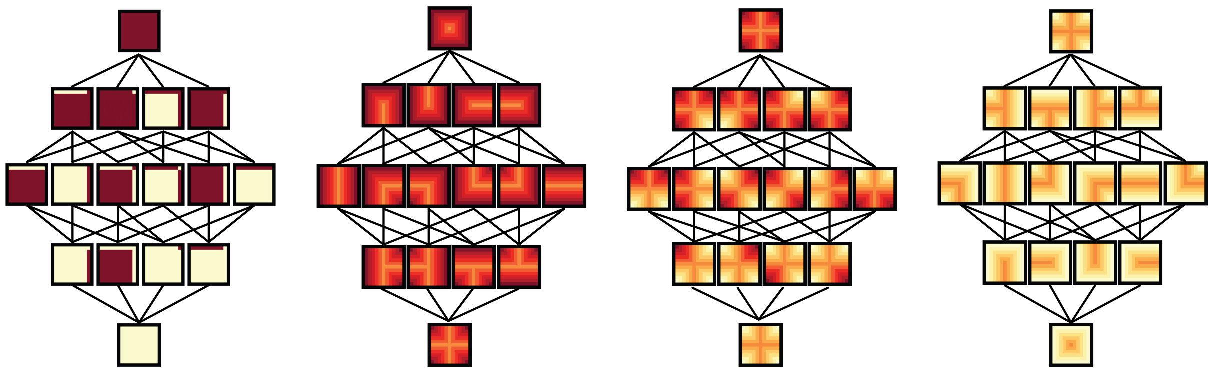


Figure 3.10: Examples of Boolean task sub-algebra with basis {□,□}, {▨,▨}, {▨,▨}, {▨,▨} and task space bounds {□, ■}, {▨,▨}, {▨,▨}, {▨,▨} respectively.

Example 3.4. *In the bin packing domain, consider the Boolean algebra bounded by the tasks where all non-terminal rewards are 0, and all terminal rewards are 0 (□) for the lower bound and 1 (■) for the upper bound. This gives $|\mathcal{M}| = 2^{121}$ possible tasks, which can all be specified by composing only $\lceil \log_2 121 \rceil = 7$ basis tasks (Figure 3.8). This basis is obtained using the same process described by Nangue Tasse* et al. *[2020b] for the Four Rooms ones (Example 2.1).*

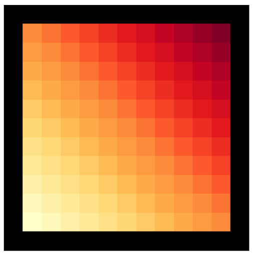
Now consider the specification of two tasks, 🟥 and 🟦, in which the robot must pack all the red and blue objects into the bin respectively. Unlike before, their respective rewards are binary for this example: □, □. Figure 3.9 shows the rewards, optimal policies, optimal value functions, and resulting robot trajectories for sample logical compositions. Note how the binary rewards result in semantically different compositions. For example, the negation of the 🟥 task here now means "do not pack all red objects in the bin". Also, meaningless compositions like (🟥 $\wedge$ $\neg$🟥)$\wedge$(🟦 $\wedge$ $\neg$🟦) now produce the lower bound task □, where all states have 0 rewards.

Figure 3.10 shows the Boolean sub-algebra generated by all logical compositions of {🟥, 🟦} (left-most), and other examples resulting from different choices of task space bounds and basis. This illustrates an interesting result: It is not necessary to restrict all transition rewards to 0 or 1—that is, use the task bounds {□, ■}—to obtain Boolean logic over tasks. As it turns out, any transition reward can be used as long as they are the transition rewards of some choice of task bounds. All such rewards are valid, with different task bounds simply leading to different operator semantics.

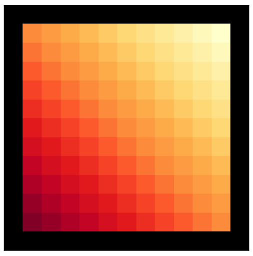
3.4 Related works

3.4.1 Logical Composition

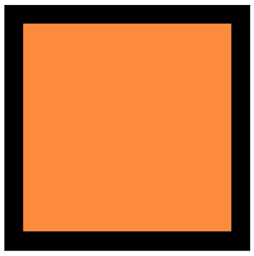
The ability to compose tasks and value functions was first demonstrated using the linearly-solvable MDP framework [Todorov 2007], where value functions could be composed to solve tasks similar to the disjunctive case [Todorov 2009]. van Niekerk *et al.* [2019] show that the same kind of composition can be achieved for deterministic tasks using entropy-regularised RL [Fox *et al.* 2016], and extend the results to the standard RL setting, where agents can optimally solve the disjunctive case. Using entropy-regularised RL, Haarnoja *et al.* [2018a] approximates the conjunction of tasks by averaging their reward functions, and demonstrates that by averaging the optimal value functions of the respective tasks, the agent can achieve near-optimal performance.



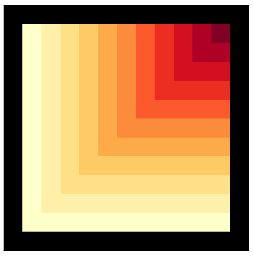
(a) $0.5(R_{🟥} + R_{🟦})$



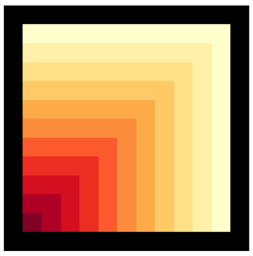
(b) $0.5(R_{\neg🟦} + R_{\neg🟥})$



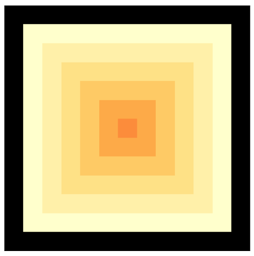
(c) $0.5\left(\begin{matrix}0.5(R_{🟥}+R_{🟦})\\+\\0.5(R_{\neg🟦}+R_{\neg🟥})\end{matrix}\right)$



(d) $\min\{R_{🟥}, R_{🟦}\}$



(e) $\min\{R_{\neg🟦}, R_{\neg🟥}\}$



(f) $\min\left\{\begin{matrix}\min\{R_{🟥},R_{🟦}\},\\ \min\{R_{\neg🟦},R_{\neg🟥}\}\end{matrix}\right\}$

Figure 3.11: Consider four tasks, 🟥, ¬🟥, 🟦, and ¬🟦 in the bin packing domain. The top row shows their conjunctions approximated using average of rewards while the bottom one shows their true conjunctions using the min of rewards. (Adapted from Nangue Tasse *et al.* [2020b])

Adamczyk *et al.* [2023ab] later generalises these results by providing optimally bounds for different classes of composition operators, beyond averaging and logical composition. Beyond zero-shot composition, some works introduce few-shot learning approaches to compose policies similarly to the disjunctive case [Hunt *et al.* 2019; Alver and Precup 2022a] or the conjunctive case [Peng *et al.* 2019; Urain *et al.* 2023]. Although lacking theoretical foundations, their results show that an agent can learn compositions of existing basis skills to solve a new complex task.

All these works consider disjunctions and conjunctions of tasks separately. In contrast, we unify these to obtain tasks specified as an arbitrary disjunction of conjunctions (or conjunction of disjunctions), and further extend them to include negations. We also note that while previous work has used the average reward function to approximate the conjunction operator [Haarnoja *et al.* 2018a; Hunt *et al.* 2019; van Niekerk *et al.* 2019], tasks specified by averaging the rewards quickly diverge from the task specified by the min of rewards (the conjunction). In particular, Figure 3.11 shows how the average reward task becomes very different to the min rewards task after only 3 operations. This highlights the importance of working towards zero-shot composition using the true logical operators, as it enables multiple arbitrary logical compositions while retaining the meaning of the specified tasks.

3.4.2 Temporal-Logic Composition

Finally, while we focus on tasks specified using only logic operators (like $\vee, \wedge, \neg$), there is a significant number of works that also consider temporal operators (like `THEN`, `UNTIL`, `EVENTUALLY`) [Littman *et al.* 2017; Jothimurugan *et al.* 2019; Camacho *et al.* 2019]. For example, tasks like 🟥 $\wedge$ $\neg$🟦 THEN $\neg$🟥 $\wedge$ 🟦, where the agent needs to first pack only red objects into the bin and then pack only blue objects into the bin. These works assume a given or predefined reward function—commonly rewards of 1 for successful task completions and zero rewards otherwise—and instead focus on state augmentation and reward shaping approaches for improving the sample efficiency of RL. More recent works focus on skill composition approaches for improving sample-efficiency [Jothimurugan *et al.* 2021; Araki *et al.* 2021; Icarte *et al.* 2022], however they only consider the temporal composition of sub-tasks and sub-skills (and not their logical composition). This is commonly done by converting a temporal logic task specification into a finite-state machine that represents the temporal order of the logic-specified sub-tasks (sub-tasks like 🟥 $\wedge$ $\neg$🟦). These works also assume that the rewards for said sub-tasks are given or predefined, so that temporally composable sub-skills can be learned for each sub-task—often through the options framework [Sutton *et al.* 1999]. In contrast, we provide a framework for obtaining the logical composition of tasks with arbitrary rewards (and skills for goal-reaching tasks without further learning), with guarantees on the semantics of the logic operators.

3.5 Conclusion

This chapter presents significant contributions to the formalisation of task specification within the realm of reinforcement learning. We have extended the existing framework of Nangue Tasse *et al.* [2020b] by generalising the operations for conjunction, disjunction, and negation—gradually progressing from a general lattice to a Boolean algebra—to tasks with arbitrary rewards in potentially stochastic environments. Interestingly, through this formalisation, we have shown that task rewards need not be constrained to only goal rewards that are either desirable or undesirable to obtain a Boolean task algebra. They need only be defined by the rewards of arbitrary choices of task bounds. Finally, we have shown that this Boolean algebra over tasks is isomorphic to a Boolean algebra over the power-set of environment transitions. This formalises the semantics of propositional logics on the space of tasks, since Boolean algebra is also the structure that formalises the semantics of propositional logics. Since propositional logics in turn are the foundation of all formal languages, this is an important result that lays the foundation for task specifications in arbitrary formal languages.

Chapter 4

Learning safe rewards

This chapter is based on the under-review work
"ROSARL: Reward-Only Safe Reinforcement Learning" [Nangue Tasse et al. 2023b], in collaboration with Tamlin Love, Mark Nemecek, Steven James, and Benjamin Rosman.

In the previous chapter, we looked at how to compose tasks to produce composite tasks with valid logical semantics. However, how do we design the rewards of the tasks to be composed to guarantee safe optimal policies? If we hope to deploy RL in the real world, agents must be capable of completing tasks while avoiding unsafe or costly behaviour. For example, a navigating robot must avoid colliding with objects and actors around it, while simultaneously learning to solve the required task. Figure 4.1 shows an example.

Many approaches in RL deal with this problem by allocating arbitrary penalties to unsafe states when hand-crafting the reward function. However, the problem of specifying a reward function for desirable, safe behaviour is notoriously difficult [Amodei *et al.* 2016]. *Importantly, penalties that are too small may result in unsafe behaviour, while penalties that are too large may result in increased learning times.* Furthermore, these rewards must be specified by an expert for each new task an agent faces. If our aim is to design truly autonomous, general agents, it is then simply impractical to require that a human designer specify penalties to guarantee optimal but safe behaviours for every task.

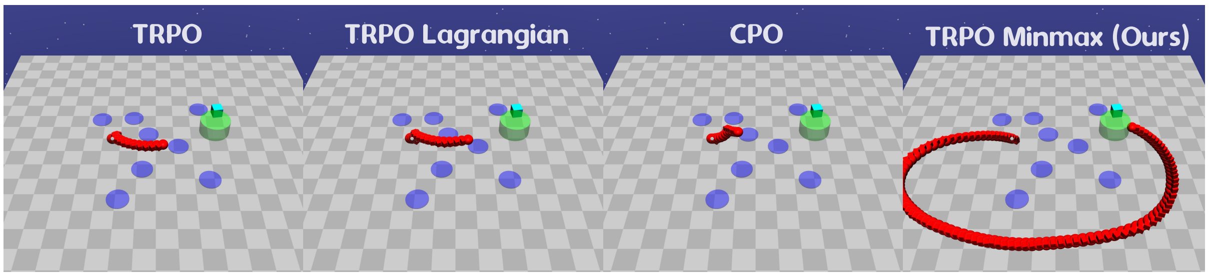


Figure 4.1: Example trajectories of prior work—**TRPO** [Schulman *et al.* 2015] (left-most), **TRPO-Lagrangian** [Ray *et al.* 2019] (middle-left), **CPO** [Achiam *et al.* 2017] (middle-right)—compared to ours (right-most) in the Safety Gym domain [Ray *et al.* 2019]. For each, a point mass agent learns to reach a goal location (green cylinder) while avoiding unsafe regions (blue circles). The cyan block is a randomly placed movable obstacle.

When safety is an explicit goal, a common approach is to constrain policy learning according to some threshold on cumulative cost [Ray *et al.* 2019; Achiam *et al.* 2017]. While effective, these approaches require the design of a cost function whose specification can be as challenging as designing a reward function. Additionally, these methods may still result in unacceptably frequent constraint violations in practice, due to the large cost threshold typically used.

Rather than attempting to both maximise a reward function and minimise a cost function, which requires specifying both rewards and costs and a new learning objective, we should simply aim to have a better reward function—since we then do not have to specify yet another scalar signal nor change the learning objective. This approach is consistent with the *reward hypothesis* [Sutton and Barto 2018] which states:

“ *All of what we mean by goals and purposes can be well thought of as maximisation of the expected value of the cumulative sum of a received scalar signal (reward).* ”

Therefore, the question we examine in this chapter is how to determine the *Minmax penalty*—the smallest penalty assigned to unsafe states such that the probability of reaching safe goals is maximised by an optimal policy. Rather than requiring an expert's input, we show that this penalty can be bounded by taking into account the *diameter* and *controllability* of an environment, and a practical estimate of it can be learned by an agent using its current value estimates. We make the following main contributions:

(i) **Bounding the Minmax penalty (Section 4.1.3)**: We obtain the analytical form of an upper and lower bound on the Minmax penalty and prove that using the upper bound results in learned behaviours that minimise the probability of visiting unsafe states (Theorem 4.2); We also show that these bounds can be accurately estimated using policy evaluation [Sutton and Barto 2018] (Theorem 4.1).

(ii) **Learning safe policies (Section 4.2)**: We show that accurately estimating the Minmax penalty or bounds is NP-hard (Theorem 4.3). We then propose a simple model-free algorithm for learning a practical estimate of the Minmax penalty while learning the task policy. Since the approach only modifies the rewards for unsafe transitions, it can be integrated into any RL pipeline that learns value functions.

(iii) **Experiments (Section 4.3)**: Finally, we investigate the behaviour of agents that only rely on their learned Minmax penalty to solve tasks safely. Our results demonstrate that these reward-only agents are capable of learning to solve tasks while avoiding unsafe states. Additionally, while prior methods often violate safety constraints, we observe that reward-only agents consistently learn safer policies.

4.1 Avoiding unsafe absorbing states

Given an environment, we aim to bound the smallest penalty (hence the largest reward) to use as feedback for unsafe transitions to guarantee safe optimal policies. To model this problem, we focus on undiscounted MDPs ($\gamma = 1$) with bounded rewards $R\colon \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [R_{\text{MIN}}\ R_{\text{MAX}}]$ that model stochastic shortest path tasks [Bertsekas and Tsitsiklis 1991]. Here, an agent must reach some goals in the non-empty set of absorbing states $\mathcal{G} \subset \mathcal{S}$ while avoiding unsafe absorbing states $\mathcal{G}^! \subset \mathcal{G}$. Since tasks are undiscounted, π^* is guaranteed to exist by assuming that the value function of *improper policies* is unbounded from below—where *proper policies* are those that are guaranteed to reach an absorbing state [van Niekerk *et al.* 2019]. Since there always exists a deterministic π^* [Sutton and Barto 1998], and π^* is proper, we will focus our attention on the set of all deterministic proper policies Π.

We formally define a safe policy as a proper policy that minimises the probability of reaching any unsafe terminal state from any internal state:

Definition 4.1 (Safe Policy)**.** *Consider an environment $\langle \mathcal{S}, \mathcal{A}, P \rangle$. Where s_T is the final state of a trajectory and $\mathcal{G}^! \subset \mathcal{G}$ is the non-empty set of unsafe absorbing states, let $P_s^\pi(s_T \in \mathcal{G}^!)$ be the probability of reaching $\mathcal{G}^!$ from s under a proper policy $\pi \in \Pi$. Then π is called safe if $\pi \in \arg\min_{\pi' \in \Pi} P_s^{\pi'}(s_T \in \mathcal{G}^!)$ for all $s \in \mathcal{S}$.*

Remark 4.1 (Safety vs Optimality)**.** *Since proper policies reach $\mathcal{G}$, Definition 4.1 equivalently says that safe policies are those that maximise the probability of reaching safe goal states $\mathcal{G} \setminus \mathcal{G}^!$. Since optimal policies are also proper, this means that safe optimal policies also maximise the probability of reaching $\mathcal{G} \setminus \mathcal{G}^!$. For example, an agent that loops forever in a non-absorbing region of the state space is neither proper, nor safe, nor optimal.*

We now define the Minmax penalty as the largest reward for unsafe transitions that lead to safe optimal policies:

Definition 4.2 (Minmax Penalty)**.** *Consider an environment $\langle \mathcal{S}, \mathcal{A}, P \rangle$ where task rewards $R(s, a, s')$ are bounded by $[R_{MIN}\ R_{MAX}]$ for all $s' \notin \mathcal{G}^!$. Let π^* be an optimal policy for one such task $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$. We define the Minmax penalty of this environment as the scalar $R_{Minmax} \in \mathbb{R}$ that satisfies the following:*

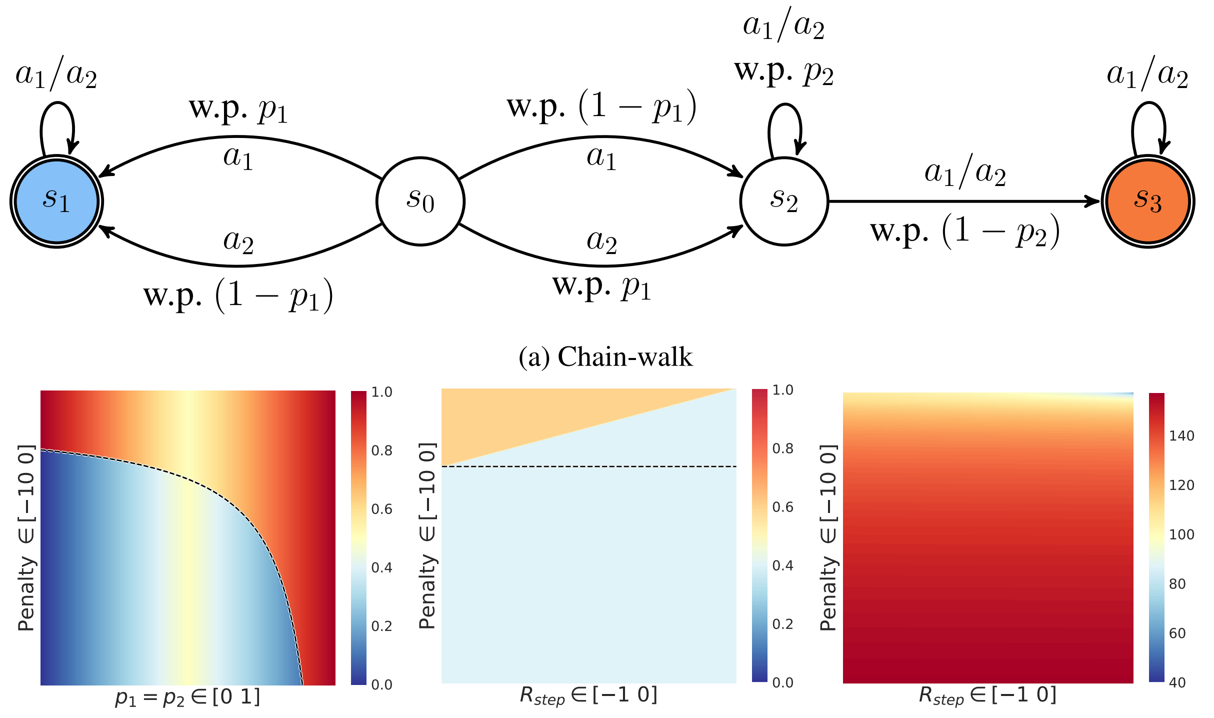
(i) If $R(s, a, s') < R_{Minmax}$ for all $s' \in \mathcal{G}^!$, then π^* is safe for all R;

(ii) If $R(s, a, s') > R_{Minmax}$ for some $s' \in \mathcal{G}^!$ reachable from $\mathcal{S} \setminus \mathcal{G}$, then there exists an R s.t. π^* is unsafe.

4.1.1 A motivating example: The chain-walk environment

To illustrate the difficulty in designing reward functions for safe behaviour, consider the simple *chain-walk* environment in Figure 4.2a. It consists of four states s_0, s_1, s_2, s_3 where $\mathcal{G} = \{s_1, s_3\}$ and $\mathcal{G}^! = \{s_1\}$. The agent has two actions a_1, a_2, the initial state is s_0, and the diagram denotes the transition probabilities. Rewards for safe transitions are bounded by $[R_{\text{MIN}}\ R_{\text{MAX}}] = [-1\ 0]$. The absorbing transitions have a reward of 0 while all other transitions have a reward of $R_{step} = -1$, and the agent must reach the goal state s_3, but not the unsafe state s_1. Hence, we want to use a sufficiently high penalty for transitions into s_1, such that the optimal policy is safe—maximises the probability of transitioning from s_0 to s_2. Figures 4.2b-4.2d exemplify how too large penalties result in longer convergence times (for example a penalty of -10), while too small ones result in unsafe policies (for example a penalty of 0), demonstrating the need to find the Minmax penalty.

Since the transitions per action are stochastic, controlled by $p_1, p_2 \in [0\ 1]$, and s_3 is further from the start state s_0 than s_1, the agent may not always be able to avoid s_1. In fact, for $p_1 = p_2 = 0$ and -1 penalty for transitions into s_1, the optimal policy is to always pick a_2



(a) Chain-walk

(b) Failure rates for $R_{step} = -1$

(c) Failure rates for $p_1 = p_2 = 0.4$

(d) Total timesteps for $p_1 = p_2 = 0.4$

Figure 4.2: A chain-walk environment (a), and the effect of different penalties (b–d) on the failure rate of optimal policies and the total timesteps needed to learn them in it (using value iteration [Sutton and Barto 1998]). The black dashed lines in (b–d) show the Minmax penalty.

which always reaches s_1. For a sufficiently high penalty for reaching s_1 (any penalty higher than -2), the optimal policy is to always pick action a_1, which always reaches s_3. However, for $p_1 = p_2 = 0.4$ (Figure 4.2c), a higher penalty is required for a_1 to stay optimal. To capture this relationship between the stochasticity of an environment and the required penalty to obtain safe policies, we introduce a notion of *controllability*, which measures the ability of an agent to reach safe goals. Additionally, observe that as p_2 increases, the probability that the agent can transition from s_2 to s_3 decreases—thereby increasing the number of timesteps spent to reach the goal. Therefore, the penalty for s_1 must also consider the environment's *diameter* to ensure an optimal policy will not simply reach s_1 to avoid self-transitions in s_2.

4.1.2 On the diameter and controllability of MDPs

Clearly, the size of the penalty that needs to be given for unsafe states depends on the *size* of the environment. We define this size as the *diameter* of the environment, which is the highest expected timesteps to reach an absorbing state from an internal state when following a proper policy:

Definition 4.3 (Diameter)**.** *Define the diameter of an environment as*

$$D := \max_{s \in \mathcal{S} \setminus \mathcal{G}} \max_{\pi \in \Pi} \mathbb{E}\left[T(s_T \in \mathcal{G} | \pi)\right],$$

where $T(s_T \in \mathcal{G}|\pi)$ is the number of timesteps taken to reach $\mathcal{G}$ from s when following a proper policy π.

Given the diameter of an environment, a possible natural choice for the reward for unsafe states is to give a penalty that is as large as receiving the smallest task reward for the longest path to safe goal states: $\mathbf{R}_{\text{MAX}} := R_{\text{MIN}}D'$, where D' is the diameter for safe policies $D' := \max_{s\in\mathcal{S}\setminus\mathcal{G}} \max_{\pi\in\Pi} \mathbb{E}\left[T(s_T \in \mathcal{G}\setminus\mathcal{G}^!|\pi)\right]$. However, while $\mathbf{R}_{\text{MAX}}$ aims to make reaching unsafe states worse than reaching safe goals, it does not consider the controllability of an environment, nor the possibility that an unsafe policy receives R_{MAX} everywhere in its trajectory. We can formally define the controllability of an environment as follows:

Definition 4.4 (Controllability)**.** *Define the degree of controllability as*

$$C := \min_{s\in S\setminus\mathcal{G}} \min_{\substack{\pi\in\Pi \\ P_s^\pi(s_T\notin\mathcal{G}^!)\neq 0}} P_s^\pi(s_T \notin \mathcal{G}^!).$$

C measures the degree of controllability of the environment by simply taking the smallest non-zero probability of reaching safe goal states by following a proper policy. For example, if the dynamics are deterministic, then any deterministic policy π will either reach a safe goal or not. That is, $P_s^\pi(s_T \notin \mathcal{G}^!)$ will either be 0 or 1. Since we require $P_s^\pi(s_T \notin \mathcal{G}^!) \neq 0$, it must be that $C = 1$. Consider, for example, the chain-walk environment with different choices for p. Since actions in s_2 do not affect the transition probability, there are only 2 relevant deterministic policies $\pi_1(s) = a_1$ and $\pi_2(s) = a_2$. This gives $P_{s_1}^{\pi_1}(s_T \notin \mathcal{G}^!) = (1-p_1)\mathbb{1}(p_2 = 1)$ and $P_{s_1}^{\pi_2}(s_T \notin \mathcal{G}^!) = p_1\mathbb{1}(p_2 = 1)$. Here, $C = 1$ when $p_1 = p_2 = 0$ because the task is deterministic and s_3 is reachable. C then tends to 0.5 as p_1 and p_2 gets closer to 0.5, making the environment uniformly random. Finally, the environment is not controllable when $p = 1$ since s_3 is unreachable from s_2.

Remark 4.2 (Uncontrollability)**.** *We can think of $C = 0$ as the limit of C when safe goals are unreachable.*

Given the diameter and controllability of an environment, we can now define a choice for the Minmax penalty that takes into account both D, C, and R_{MAX}: $\mathbf{R}_{\text{MIN}} := (R_{\text{MIN}} - R_{\text{MAX}})\frac{D}{C}$. This choice of penalty says that since stochastic shortest path tasks require an agent to learn to achieve desired terminal states, if the agent enters an unsafe terminal state, it should receive the largest penalty possible by a proper policy. We now investigate the effect of these penalties on the failure rate of optimal policies.

4.1.3 On the failure rate of optimal policies

We begin by proposing a simple model-based algorithm for estimating the diameter and controllability, from which the penalties are then obtained. We describe the method here and present the pseudo-code in **Algorithm 4**. Here, the diameter is estimated as follows: (i) For each deterministic policy π, estimate its expected timesteps $T(s_T \in \mathcal{G})$ (or $T(s_T \in \mathcal{G}\setminus\mathcal{G}^!)$ for D') by using policy evaluation [Sutton and Barto 2018] with rewards of 1 at all internal states; (ii) Then, calculate D using the equation in Definition 4.3. Similarly, the controllability is estimated by estimating the reach probability $P_s^\pi(s_T \notin \mathcal{G}^!)$ of each deterministic policy π using rewards of 1 for transitions into safe goal states and zero rewards otherwise. This approach converges via the convergence of policy evaluation (**Theorem 4.1**).

Algorithm 4: Estimating the Diameter and Controllability

```
Input      : ⟨$\mathcal{S}, \mathcal{A}, P$⟩, $R_D(s') := \mathbb{1}(s' \notin \mathcal{G})$, $R_C(s,a,s') := \mathbb{1}(s \notin \mathcal{G}$ and $s' \in \mathcal{G} \setminus \mathcal{G}^!)$
Initialise : Diameter $D = 0$, Controllability $C = 1$, Value functions $V_D^\pi(s) = 0$, $V_C^\pi(s) = 0$,
             Error $\Delta = 1$

for $\pi \in \Pi$ do
    /* Policy evaluation for D */
    while $\Delta > 0$ do
        $\Delta \leftarrow 0$
        for $s \in \mathcal{S}$ do
            $v' \leftarrow \sum_{s'} P(s'|s,\pi(s))(R_D(s') + V_D^\pi(s'))$
            $\Delta = \max\{\Delta, |V_D^\pi(s) - v'|\}$
            $V_D^\pi(s) \leftarrow v'$
    for $s \in \mathcal{S}$ do
        $D = \max\{D, V_D^\pi(s)\}$

for $\pi \in \Pi$ do
    /* Policy evaluation for C */
    while $\Delta > 0$ do
        $\Delta \leftarrow 0$
        for $s \in \mathcal{S}$ do
            $v' \leftarrow \sum_{s'} P(s'|s,\pi(s))(R_C(s,\pi(s),s') + V_C^\pi(s'))$
            $\Delta = \max\{\Delta, |V_C^\pi(s) - v'|\}$
            $V_C^\pi(s) \leftarrow v'$
    for $s \in \mathcal{S}$ do
        $C = \min\{C, V_C^\pi(s)\}$ if $V_C^\pi(s) \neq 0$ else $C$
```

Theorem 4.1 (Estimation)**.** *Algorithm 4 converges to D and C for any controllable environment.*

Proof. This follows from the convergence guarantee of policy evaluation [Sutton and Barto 1998]. □

Figure 4.3 shows the result of applying this algorithm in the chain-walk MDP. Here, R_{Minmax} is compared to accounting for D only ($\mathbf{R}_{\text{MAX}}$) and accounting for both C and D ($\mathbf{R}_{\text{MIN}}$). Interestingly, we can observe $\mathbf{R}_{\text{MIN}} \leq R_{\text{Minmax}}$ and $\mathbf{R}_{\text{MAX}} \geq R_{\text{Minmax}}$ consistently, highlighting how considering the diameter only is insufficient to guarantee safe optimal policies. It also indicates that these penalties may bound R_{Minmax} in general. We show in **Theorem 4.2** that this is indeed the case.

Theorem 4.2 (Safety Bounds)**.** *Consider a controllable environment where task rewards are bounded by* $[R_{MIN}\ R_{MAX}]$ *for all* $s' \notin \mathcal{G}^!$*. Then* $\mathbf{R}_{MIN} \leq R_{Minmax} \leq \mathbf{R}_{MAX}$.

Proof. Let π^* be an optimal policy for an arbitrary task $\langle \mathcal{S}, \mathcal{A}, P, R\rangle$ in the environment. Given the definition of the Minmax penalty (Definition 4.2), we need to show the following:

(i) If $R(s,a,s') < \mathbf{R}_{\text{MIN}}$ for all $s' \in \mathcal{G}^!$, then π^* is safe for all R; and

(ii) If $R(s,a,s') > \mathbf{R}_{\text{MAX}}$ for some $s' \in \mathcal{G}^!$ reachable from $\mathcal{S} \setminus \mathcal{G}$, then there exists an R s.t. π^* is unsafe.

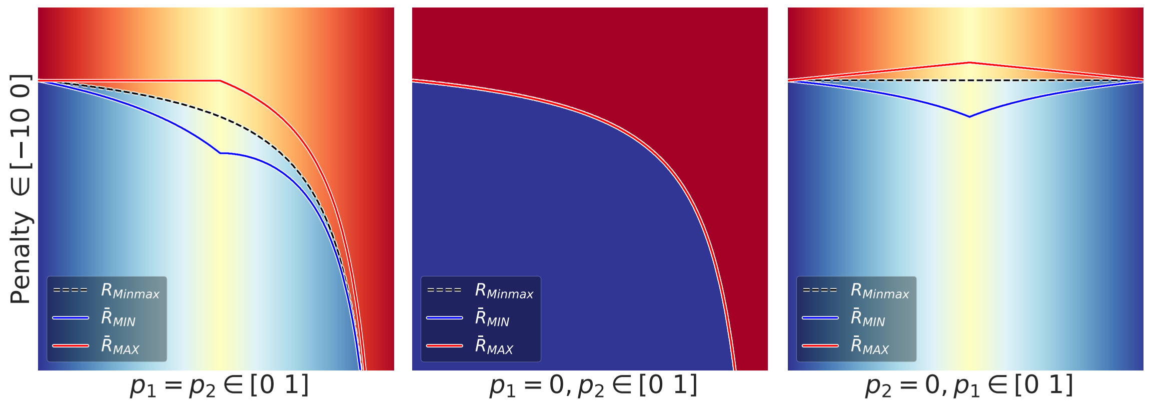
(i) Since π^* is optimal, it is also proper and hence must reach $\mathcal{G}$.

Assume π^* is unsafe. Then there exists another proper policy π that is safe, such that

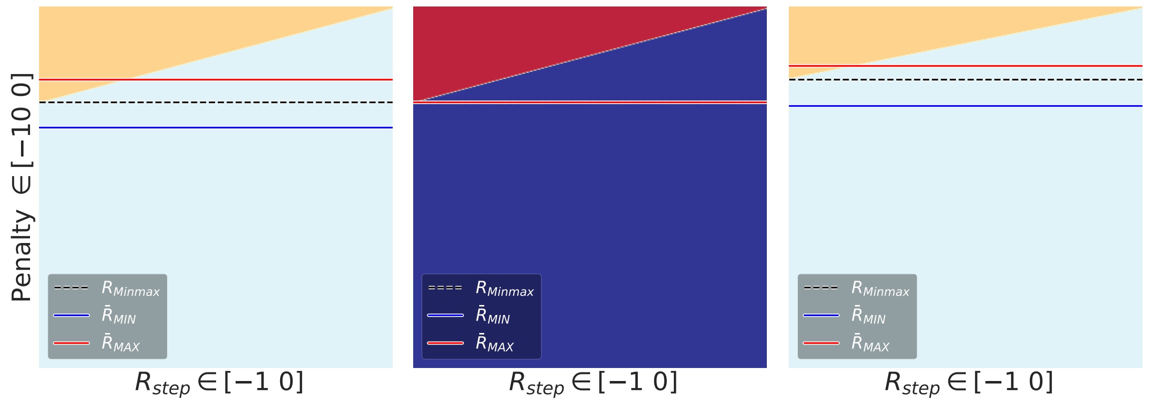
$$P_s^\pi(s_T \in \mathcal{G}^!) < P_s^{\pi^*}(s_T \in \mathcal{G}^!) \quad \text{for some } s \in \mathcal{S}.$$

Then,

$$V^{\pi^*}(s) \geq V^\pi(s)$$



(a) $R_{step} = -1$ (b) $R_{step} = -1$ (c) $R_{step} = -1$



(d) p_1=p_2=0.4 (e) p_1=0, p_2=0.4 (f) p_1=0.4, p_2=0

Figure 4.3: Effect of stochasticity (p_1 and p_2) and task rewards (R_{step}) on the bounds ($\mathbf{R}_{\text{MIN}}$ and $\mathbf{R}_{\text{MAX}}$) of the Minmax penalty (R_{Minmax}) in the chain-walk environment. The controllability and diameter for the bounds are estimated using Algorithm 4. The optimal policies are obtained via value iteration [Sutton and Barto 1998].

$$\implies \mathbb{E}_s^{\pi^*}\left[\sum_{t=0}^{\infty} R(s_t, a_t, s_{t+1})\right] \geq \mathbb{E}_s^{\pi}\left[\sum_{t=0}^{\infty} R(s_t, a_t, s_{t+1})\right]$$

$$\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1} + R(s_T, a_T, s_{T+1})\right] \geq \mathbb{E}_s^{\pi}\left[G^{T-1} + R(s_T, a_T, s_{T+1})\right],$$

where $G^{T-1} = \sum_{t=0}^{T-1} R(s_t, a_t, s_{t+1})$ and T is a random variable denoting when $s_{T+1} \in \mathcal{G}$.

$$\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \notin \mathcal{G}^!)R(s_T, a_T, s_{T+1}) + P_s^{\pi^*}(s_T \in \mathcal{G}^!)\mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})\right)$$
$$\geq \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!)R(s_T, a_T, s_{T+1}) + P_s^{\pi}(s_T \in \mathcal{G}^!)\mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})\right),$$

where $\mathbf{R}_{\text{unsafe}}$ denotes the rewards for transitions into $\mathcal{G}^!$ and $a_T = \pi^*(s_T)$.

$$\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \notin \mathcal{G}^!)R(s_T, a_T, s_{T+1}) + \mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})\right)$$
$$\geq \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!)R(s_T, a_T, s_{T+1}) + P_s^{\pi}(s_T \in \mathcal{G}^!)\mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})\right),$$

$$\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(1 - P_s^{\pi}(s_T \in \mathcal{G}^!)\right)\mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})$$
$$\geq \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right)R(s_T, a_T, s_{T+1})$$

$$
\begin{aligned}
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(1 - P_s^{\pi}(s_T \in \mathcal{G}^!)\right)\mathbf{R}_{\text{MIN}}, \text{ since } \mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1}) < \mathbf{R}_{\text{MIN}}.\\
&\qquad > \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1})\\
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(1 - P_s^{\pi}(s_T \in \mathcal{G}^!)\right)(R_{\text{MIN}} - R_{\text{MAX}})\frac{D}{C}\\
&\qquad > \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1})\\
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + (R_{\text{MIN}} - R_{\text{MAX}})D\\
&\qquad > \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1}), \text{ using definition of } C.\\
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] - R_{\text{MAX}}D\\
&\qquad > \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1}) - R_{\text{MIN}}D\\
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] - R_{\text{MAX}}D > 0,\\
&\qquad \text{since } \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1}) \geq R_{\text{MIN}}D\\
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] > R_{\text{MAX}}D.
\end{aligned}
$$

But this is a contradiction since the expected return of following an optimal policy up to a terminal state without the reward for entering the terminal state must be less than receiving R_{MAX} for every step of the longest possible trajectory to $\mathcal{G}$. Hence we must have $\pi^* \in \arg\min_{\pi} P_s^{\pi}(s_T \in \mathcal{G}^!)$.

(ii) Assume π^* is safe. Then, $P_s^{\pi^*}(s_T \notin \mathcal{G}^!) \geq P_s^{\pi'}(s_T \notin \mathcal{G}^!)$ for all $s \in \mathcal{S}$, $\pi' \in \Pi$.

Let π be the policy that maximises the probability of reaching $s' \in \mathcal{G}^!$ from some state $s \in \mathcal{G}$. Then, similarly to (i), we have

$$
\begin{aligned}
&\qquad V^{\pi^*}(s) \geq V^{\pi}(s)\\
&\implies \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \in \mathcal{G}^!) - P_s^{\pi}(s_T \in \mathcal{G}^!)\right)\mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})\\
&\qquad \geq \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \notin \mathcal{G}^!) - P_s^{\pi^*}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1})\\
&\implies \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \in \mathcal{G}^!) - P_s^{\pi^*}(s_T \in \mathcal{G}^!)\right)\mathbf{R}_{\text{unsafe}}(s_T, a_T, s_{T+1})\\
&\qquad \leq \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \notin \mathcal{G}^!) - P_s^{\pi}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1})\\
&\implies \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \in \mathcal{G}^!) - P_s^{\pi^*}(s_T \in \mathcal{G}^!)\right)\mathbf{R}_{\text{MAX}}\\
&\qquad < \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \notin \mathcal{G}^!) - P_s^{\pi}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1}), \text{ since } \mathbf{R}_{\text{unsafe}} > \mathbf{R}_{\text{MAX}}.\\
&\implies \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + \left(P_s^{\pi}(s_T \in \mathcal{G}^!) - P_s^{\pi^*}(s_T \in \mathcal{G}^!)\right) R_{\text{MIN}}D'\\
&\qquad < \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \notin \mathcal{G}^!) - P_s^{\pi}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1}), \text{ by definition of } \mathbf{R}_{\text{MAX}}.\\
&\implies \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + R_{\text{MIN}}D'\\
&\qquad < \mathbb{E}_s^{\pi^*}\left[G^{T-1}\right] + \left(P_s^{\pi^*}(s_T \notin \mathcal{G}^!) - P_s^{\pi}(s_T \notin \mathcal{G}^!)\right) R(s_T, a_T, s_{T+1})\\
&\implies \mathbb{E}_s^{\pi}\left[G^{T-1}\right] + R_{\text{MIN}}D' < 0
\end{aligned}
$$

But this is a contradiction when R is such that the agent receives a reward of $R_{\text{MAX}} \geq |R_{\text{MIN}}|D'$ at least once in its trajectory when following π and zero everywhere else.

□

Theorem 4.2 says that for any MDP whose rewards for unsafe transitions are bounded above by $\mathbf{R}_{\text{MIN}}$, the optimal policy both minimises the probability of reaching unsafe states and maximises the probability of reaching safe goal states. Hence, any penalty $\mathbf{R}_{\text{MIN}} - \epsilon$, where $\epsilon > 0$ can be arbitrarily small, will guarantee safe optimal policies. Similarly, the theorem shows that any reward higher than $\mathbf{R}_{\text{MAX}}$ may have optimal policies that do not minimise the probability of reaching unsafe states. These can be observed in Figure 4.3. The figure demonstrates why considering both the diameter and controllability of an MDP is necessary to guarantee safe policies, because the diameter alone does not always minimise the failure rate.

4.2 A practical algorithm for learning safe policies

While the Minmax penalty of an MDP can be accurately estimated using policy evaluation (Algorithm 4), it requires knowledge of the environment dynamics (or an estimate of it). These are difficult quantities to estimate from an agent's experience, which is further complicated by the need to also learn the true optimal policy for the estimated Minmax penalty. Hence, obtaining an accurate estimate of the Minmax penalty is impractical in model-free and function approximation settings where the state and action spaces are large. In fact, it is NP-hard since it depends on the diameter, which requires solving a longest-path problem.

Theorem 4.3 (Complexity)**.** *Estimating the Minmax penalty R_{Minmax} accurately is NP-hard.*

Proof. This follows from the NP-hardness of longest-path problems. Since the Minmax penalty is bounded by $\mathbf{R}_{\text{MIN}}$ and $\mathbf{R}_{\text{MAX}}$, both are defined by the diameter, which is in turn defined as the expected total timesteps of the longest path. $\square$

Given the above challenges, we require a practical method for learning the Minmax penalty. Ideally, this method should require no knowledge of the environment dynamics and should easily integrate with existing RL approaches. To achieve this, we first note that $(R_{\text{MIN}} - R_{\text{MAX}})\frac{D}{C} = (DR_{\text{MIN}} - DR_{\text{MAX}})\frac{1}{C} = (V_{\text{MIN}} - V_{\text{MAX}})\frac{1}{C}$, where V_{MIN} and V_{MAX} are the value function bounds. Hence, a practical estimate of the Minmax penalty can be efficiently learned by estimating the value gap $V_{\text{MIN}} - V_{\text{MAX}}$ using observations of the reward and the agent's estimate of the value function. We describe the method here and present the pseudo-code in **Algorithm 5**. This algorithm requires initial estimates of R_{MIN} and R_{MAX}, which in this work are initialised to 0. The agent receives a reward r_t after each environment interaction and updates its estimate of the reward bounds $R_{\text{MIN}} \leftarrow \min(R_{\text{MIN}}, r_t)$ and $R_{\text{MAX}} \leftarrow \max(R_{\text{MAX}}, r_t)$, the value bounds $V_{\text{MIN}} \leftarrow \min(V_{\text{MIN}}, R_{\text{MIN}}, V(s_t))$ and $V_{\text{MAX}} \leftarrow \max(V_{\text{MAX}}, R_{\text{MAX}}, V(s_t))$, and the Minmax penalty $\mathbf{R}_{\text{MIN}} \leftarrow V_{\text{MIN}} - V_{\text{MAX}}$, where $V(s_t)$ is the learned value function at time step t. We note how the controllability C is not explicitly considered in this estimate of $\mathbf{R}_{\text{MIN}}$. Given that the main purpose of C is to make $\mathbf{R}_{\text{MIN}}$ more negative the more stochastic the environment is, we notice that this is already achieved in practice by the reward and value estimates. Since R_{MIN} is estimated using $R_{\text{MIN}} \leftarrow \min(R_{\text{MIN}}, r_t)$, then every time the agent enters an unsafe state, we have that: $r_t \leftarrow \mathbf{R}_{\text{MIN}}$, $R_{\text{MIN}} \leftarrow \mathbf{R}_{\text{MIN}}$, and then $\mathbf{R}_{\text{MIN}} \leftarrow \mathbf{R}_{\text{MIN}} - V_{\text{MAX}}$. This means that when the estimated V_{MAX} is greater than zero, the penalty estimate $\mathbf{R}_{\text{MIN}}$ become more negative every time the agent enters an unsafe state.

Finally, whenever an agent encounters an unsafe state, the reward can be replaced by $\mathbf{R}_{\text{MIN}}$ to disincentivise unsafe behaviour. Since V_{MAX} is estimated using $V_{\text{MAX}} \leftarrow \max(V_{\text{MAX}}, R_{\text{MAX}}, V(s_t))$, it leads to an optimistic estimation of $\mathbf{R}_{\text{MIN}}$. Hence, in practice, we observe no need to add $\epsilon > 0$ to $\mathbf{R}_{\text{MIN}}$.

Algorithm 5: RL while learning Minmax penalty

```
Input      : RL algorithm $\mathbf{A}$, max timesteps $T$
Initialise : $R_{\text{MIN}} = 0, R_{\text{MAX}} = 0, V_{\text{MIN}} = R_{\text{MIN}}, V_{\text{MAX}} = R_{\text{MAX}}$, $\pi$ and $V$ as per $\mathbf{A}$
  for t in T do
    observe a state $s_t$, take an action $a_t$ using $\pi$ as per $\mathbf{A}$, and observe $s_{t+1}, r_t$
    $R_{\text{MIN}}, R_{\text{MAX}} \leftarrow \min(R_{\text{MIN}}, r_t), \max(R_{\text{MAX}}, r_t)$
    $V_{\text{MIN}}, V_{\text{MAX}} \leftarrow \min(V_{\text{MIN}}, R_{\text{MIN}}, V(s_t)), \max(V_{\text{MAX}}, R_{\text{MAX}}, V(s_t))$
    $\mathbf{R}_{\text{MIN}} \leftarrow V_{\text{MIN}} - V_{\text{MAX}}$
    $r_t \leftarrow \mathbf{R}_{\text{MIN}}$ if $s_{t+1} \in \mathcal{G}^!$ else $r_t$
    update $\pi$ and $V$ with $(s_t, a_t, s_{t+1}, r_t)$ as per $\mathbf{A}$
```

4.3 Experiments

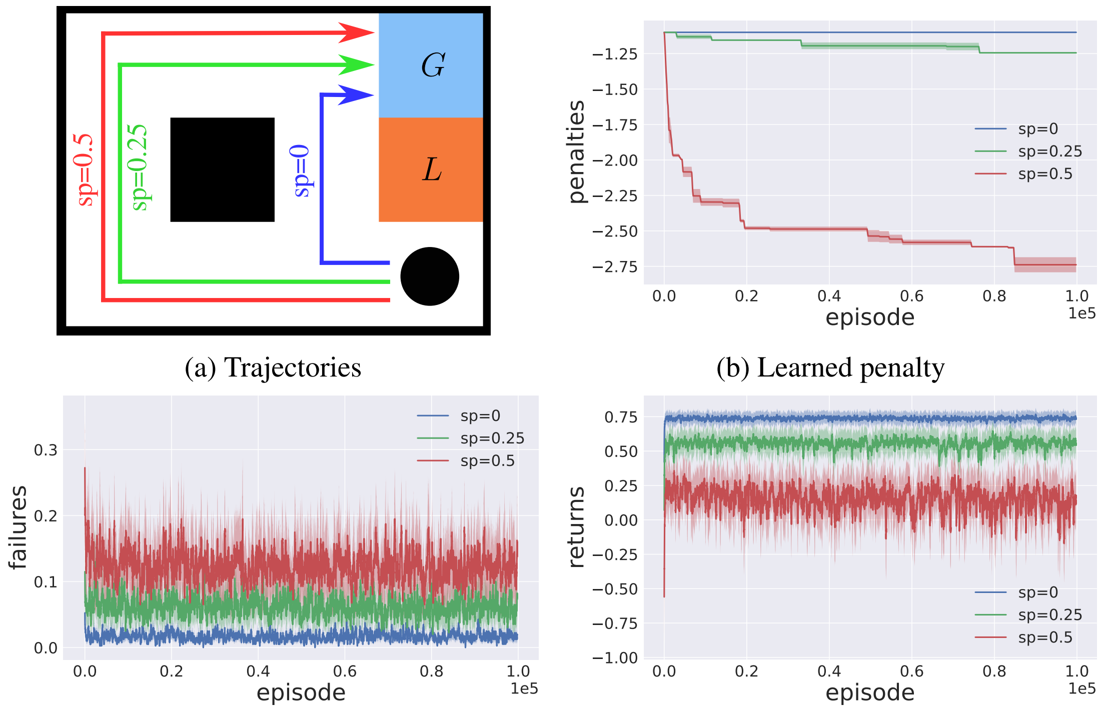
While the theoretical Minmax penalty is guaranteed to lead to optimal safe policies, it is unclear whether this also holds for the practical estimate proposed in Section 4.2. Hence, this section aims to investigate three main natural questions regarding the proposed practical algorithm: (i) How does Algorithm 5 behave when the theoretical assumptions are satisfied? (ii) How does Algorithm 5 behave when the theoretical assumptions are *not* satisfied? (iii) How does Algorithm 5 compare to prior approaches towards Safe RL? For each result, we report the mean (solid line) and one standard deviation around it (shaded region).

4.3.1 Behaviour when theory holds

To answer this question, we consider the LAVA GRIDWORLD which is a tabular stochastic shortest path domain.

Domain (LAVA GRIDWORLD) This is a simple gridworld environment with 11 positions ($|\mathcal{S}| = 11$) and 4 cardinal actions ($|\mathcal{A}| = 4$). The agent here must reach a goal location G while avoiding a lava location L (hence $\mathcal{G} = \{L, G\}$ and $\mathcal{G}^! = \{L\}$). A wall is also present in the environment and, while not unsafe, must be navigated around. The environment has a *slip probability (sp)*, so that with probability sp the agent's action is overridden with a random action. The agent receives $R_{\text{MAX}} = +1$ reward for reaching the goal, as well as $R_{step} = -0.1$ reward at each timestep to incentivise taking the shortest path to the goal. To test our approach, we modify Q-learning [Watkins 1989] with ϵ-greedy exploration such that the agent updates its estimate of the Minmax penalty as learning progresses and uses it as the reward whenever the lava state is reached, following the procedure outlined in Section 4.2. The action-value function is initialised to 0 for all states and actions, $\epsilon = 0.1$ and the learning rate $\alpha = 0.1$.

Setup and Results We examine the performance of our modified Q-learning approach across three values of the slip probability of the LAVA GRIDWORLD. A slip probability of 0 represents a fully deterministic environment, while a slip probability of 0.5 represents a more stochastic environment. Results are plotted in Figure 4.4. In the case of the fully deterministic environment, the Minmax penalty bound obtained via Algorithm 4 is $\mathbf{R}_{\text{MIN}} = -9.9$, since $C = 1$ and $D = 9$. However, the agent is able to learn a relatively smaller penalty (-1.1 in Figure 4.4b) to consistently minimise failure rate and maximise returns (Figures 4.4c and 4.4d). The resulting optimal policy then chooses the shorter path that passes near the lava location ($sp = 0$ in Figure 4.4a). As the stochasticity of the environment increases, a larger penalty is learned to incentivise longer, safer policies. Given the starting position of the agent next to the lava, the failure rate inevitably increases with increased stochasticity. The resulting optimal policy then chooses the longer path that passes to the left of the centre wall ($sp = 0.25$ and $sp = 0.5$ in Figure 4.4a). We can, therefore, conclude that while there is a gap between the true Minmax penalty and the one learned via Algorithm 5, this algorithm can still learn optimal safe policies when the theoretical setting holds.



(a) Trajectories (b) Learned penalty

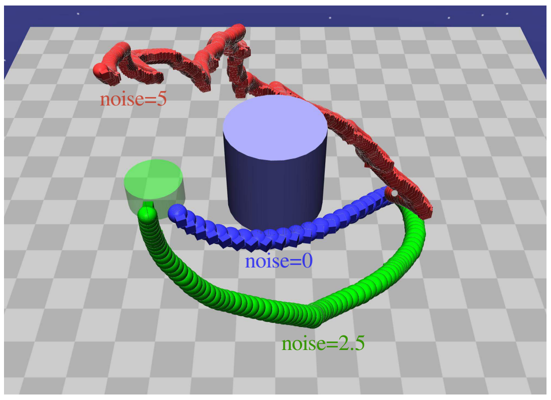
(c) Failure rate (d) Average returns

Figure 4.4: Effect of increase in the slip probability of the LAVA GRIDWORLD on the learned Minmax penalty and corresponding failure rate and returns. The black circle in (a) represents the agent. The experiments are run over 10,000 episodes and averaged over 70 random seeds. The shaded regions indicate one standard deviation.

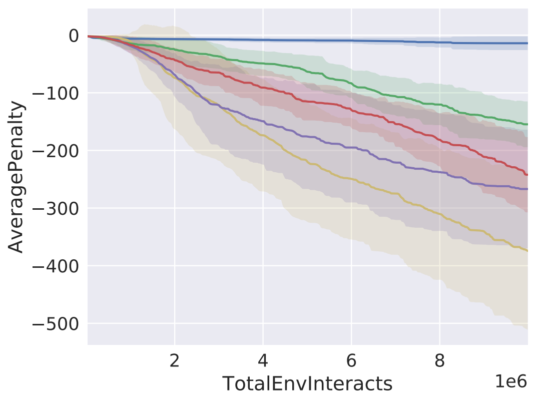
4.3.2 Behaviour when theory does not hold

To answer this question, we consider the Safety Gym PILLAR domain, a discounted continuous control setting.

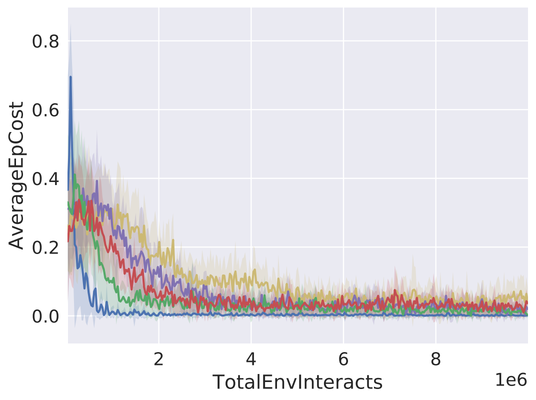
Domain (Safety Gym PILLAR) This is a custom Safety Gym environment [Ray *et al.* 2019], in which the simple point robot must navigate to a goal location 🟢 around a large pillar 🔵 (hence $\mathcal{G} = \{$🔵, 🟢$\}$ and $\mathcal{G}^! = \{$🔵$\}$). Just as in Ray *et al.* [2019], the agent uses *pseudo-lidar* to observe the distance to objects around it ($|\mathcal{S}| = \mathbb{R}^{60}$), and the action space is continuous over two actuators controlling the direction and forward velocity ($|\mathcal{A}| = \mathbb{R}^2$). The goal, pillar, and agent locations remain unchanged for all episodes. The agent is rewarded for reaching the goal, as well as for moving towards it (the default dense distance-based reward). Each episode terminates after 1000 timesteps or once the agent reaches the goal. Also collisions with the pillar results in immediate episode termination with a reward of -1. To test our approach, we modify Trust Region Policy Optimisation (TRPO) [Schulman *et al.* 2015] (denoted TRPO-Minmax) to use the estimate of the Minmax penalty as described in Algorithm 5. We use TRPO since it is a state-of-the-art RL algorithm for continuous action spaces, and it is also a canonical baseline in prior works [Ray *et al.* 2019; Sootla *et al.* 2022]. We also use the same hyperparameters for TRPO as in Ray *et al.* [2019], since those are the hyperparameters they found to work best for tasks in the Safety Gym environment.



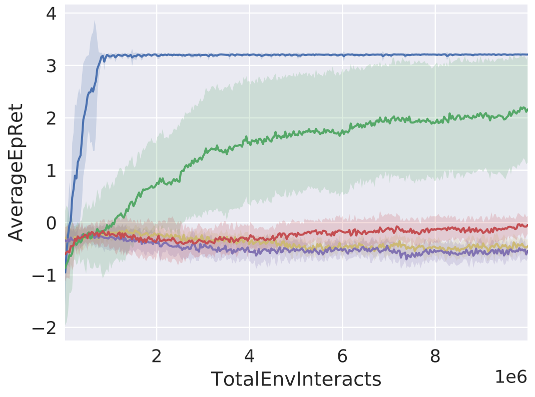
(a) Trajectories



(b) Learned penalty



(c) Failure rate



(d) Average returns

Figure 4.5: Effect of increase in the action noise of the PILLAR domain on the learned Minmax penalty and corresponding failure rate and returns. The spheres in (a) show the agent's trajectories. The experiments are run over 10 million episodes and averaged over 10 random seeds. The shaded regions indicate one standard deviation.

Setup and Results We examine the performance of TRPO-Minmax for five levels of noise in the PILLAR environment, similarly to the experiments in Section 4.3.1. Here, the value of the noise denotes the scalar by which a random action vector is scaled before vector addition with the agent's action. Note that this noise value is different from the random seed used for the random number generator in our experiments. Results are plotted in Figure 4.5. We observe similar results to Section 4.3.1, where the agent uses its learned Minmax penalty (Figure 4.5b) to successfully learn safe policies (Figure 4.5c) while solving the task (Figure 4.5d), using safer paths for more noisy dynamics (Figure 4.5a). Interestingly, it also correctly prioritises low failure rates when the dynamics are too noisy to safely reach the goal ($noise \geq 5$). We can, therefore, conclude that Algorithm 5 can learn safe policies even in discounted high-dimensional continuous-control domains requiring function approximation.

4.3.3 Comparison to baselines

To answer this question, we consider representative baselines in the PILLAR environment and canonical Safety Gym ones.

Domains (Safety Gym POINTGOAL1-HARD, POINTPUSH1-HARD, CARBUTTON1-HARD) These domains are respectively the modified version of the POINTGOAL1, POINTPUSH1, and CARBUTTON1 tasks from OpenAI's Safety Gym environments [Ray *et al.* 2019], which represents complex, high-dimensional, continuous control tasks. In all of the original domains, $\mathcal{G} = \emptyset$ by default. We only modify each of them to make unsafe transitions terminal $\mathcal{G} = \mathcal{G}^! = \{\text{states with cost} > 0\}$, leaving the safe goal states non-terminal ($\mathcal{G} \setminus \mathcal{G}^! = \emptyset$). In POINTGOAL1-HARD, a

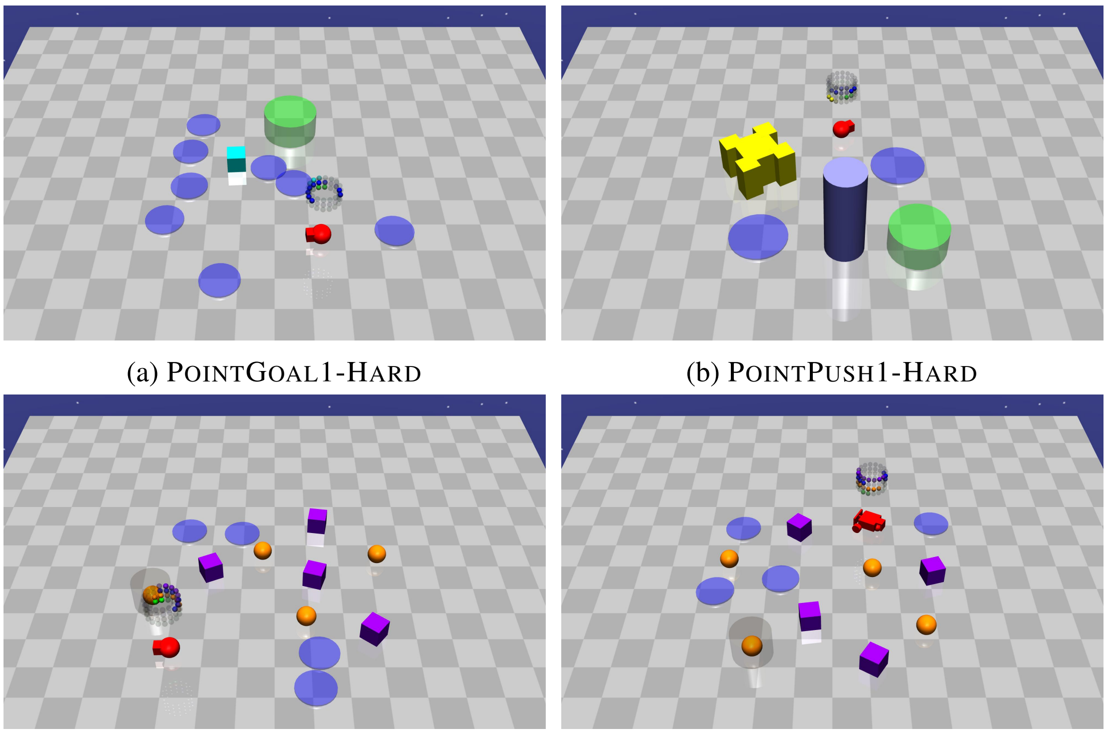
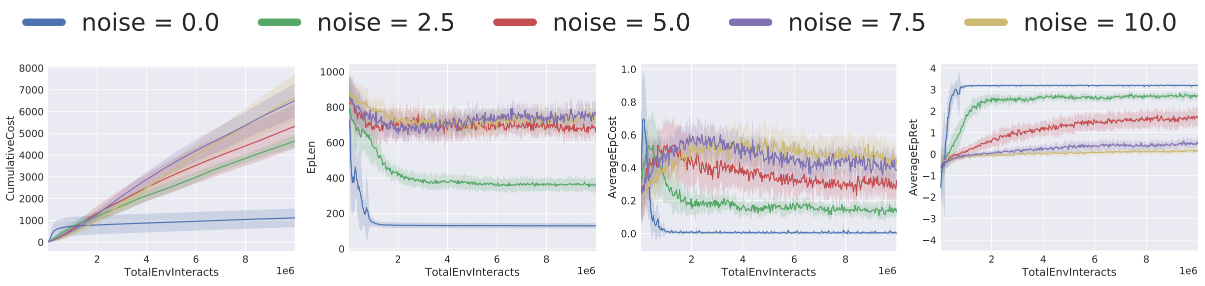

(a) POINTGOAL1-HARD (b) POINTPUSH1-HARD (c) POINTBUTTON1-HARD (d) CARBUTTON1-HARD

Figure 4.6: Safety Gym *Hard* environments.

simple robot must navigate to a goal location ⬮ across a 2D plane while avoiding several hazards ◯ (where $\mathcal{G} = \mathcal{G}^! = \{$◯$\}$). The agent's sensors, actions, and rewards are identical to the Pillar domain. Unlike the Pillar domain, the goal's location is randomly reset when the agent reaches it, but does not terminate the episode. Each episode only terminates after 1000 timesteps or once the agent enters a hazard blue region, thereby making the problem much harder than the original PointGoal1 task. PointPush1-Hard is similar to PointGoal1-Hard, but with the addition of a pillar obstacle ⬮ and a large box 🟨 the agent must push to the goal location ⬮ to receive the goal reward (where $\mathcal{G} = \mathcal{G}^! = \{$◯, ⬮$\}$). Finally, PointButton1-Hard and CarButton1-Hard are also similar to PointGoal1-Hard, but with the more complex car robot for CarButton1-Hard and the addition of these to both: (i) *Gremlins* ■, which are dynamic obstacles that move around the environment and must be avoided; and (ii) Buttons ●, where the agent must reach the goal button with a cylinder ⬮ to receive the goal reward (where $\mathcal{G} = \mathcal{G}^! = \{$◯, ■, ●$\}$). Figure 4.6 illustrates all these domains.

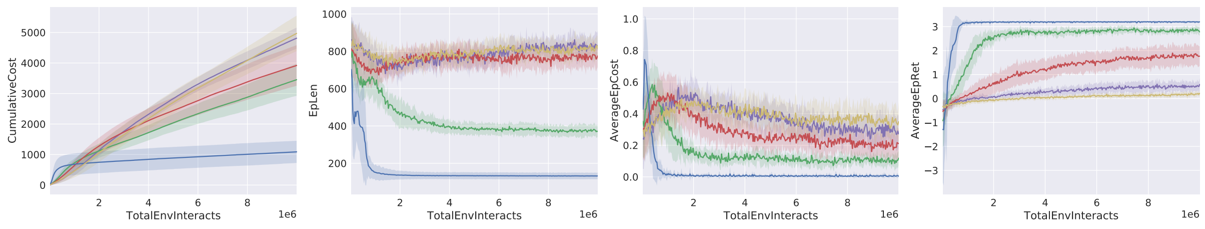
Baselines As a baseline representative of typical RL approaches, we use Trust Region Policy Optimisation (TRPO) [Schulman *et al.* 2015]. To represent constraint-based approaches, we compare against Constrained Policy Optimisation (CPO) [Achiam *et al.* 2017], TRPO with Lagrangian constraints (TRPO-Lagrangian) [Ray *et al.* 2019], and Sauté RL with TRPO (Sauté-TRPO) [Sootla *et al.* 2022]. All baselines except Sauté-TRPO use the implementations provided by Ray *et al.* [2019], and form a set of widely used baselines in safety domains [Zhang *et al.* 2020; Sootla *et al.* 2022; Yang *et al.* 2023]. Sauté-TRPO uses the implementation provided by Sootla *et al.* [2022]. As in Ray *et al.* [2019], all approaches use feed-forward MLPs, value networks of size (256,256), and $tanh$ activation functions. For the constrained algorithms, the cost threshold is set to 0, the best we found from trying $c \in \{0, 1, 25, 100\}$ (Ray *et al.* [2019] used a threshold of 25). Finally, all the TRPO hyperparameters are kept the same as in prior work since those are the hyperparameters they found to work best for a variety of tasks in the Safety Gym environment. The experiments are run over 10 million episodes and averaged over 10 random seeds. The shaded regions in the plots indicate one standard deviation over the 10 random seeds.

Pillar Results (Figures 4.7-4.11) The baselines all achieve similar performance (except Sauté-TRPO), maximising returns but maintaining a relatively high failure rate. Their similar performance also suggests that despite these baselines being different ways of constrained learning of policies, they may still result in similar learning dynamics when everything else is kept constant—such as fixed goal states and fixed unsafe states. By examining the failure rates in



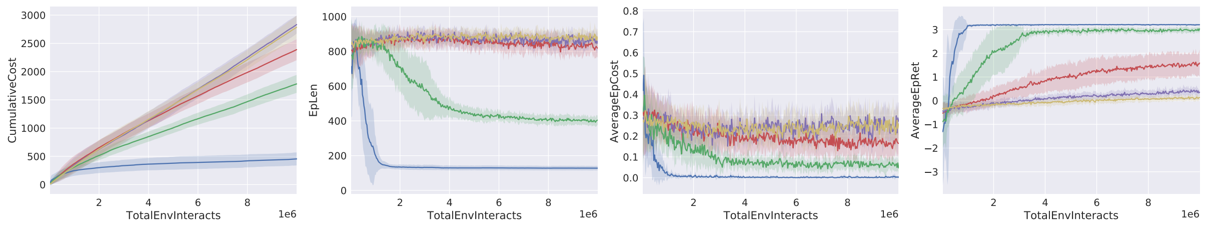
(a) Cumulative cost (b) Episode length (c) Failure rate (d) Average returns

Figure 4.7: Performance of **TRPO** in the Pillar environment with varying noise.



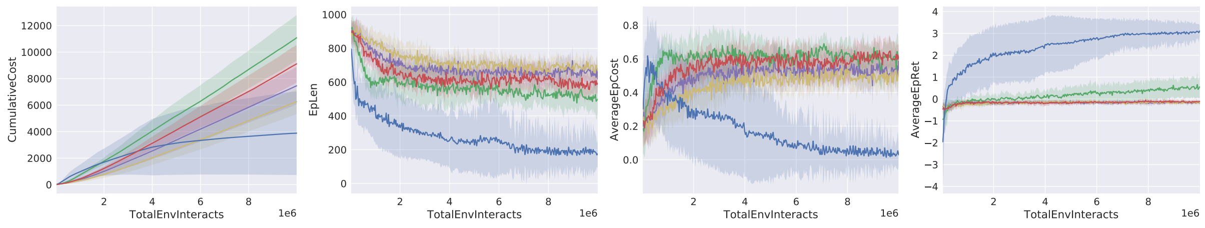
(a) Cumulative cost (b) Episode length (c) Failure rate (d) Average returns

Figure 4.8: Performance of **TRPO-Lagrangian** in the PILLAR environment with varying noise.



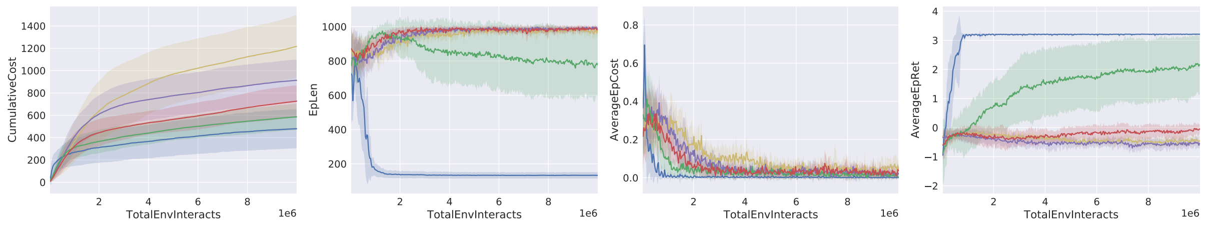
(a) Cumulative cost (b) Episode length (c) Failure rate (d) Average returns

Figure 4.9: Performance of **CPO** in the PILLAR environment with varying noise.



(a) Cumulative cost (b) Episode length (c) Failure rate (d) Average returns

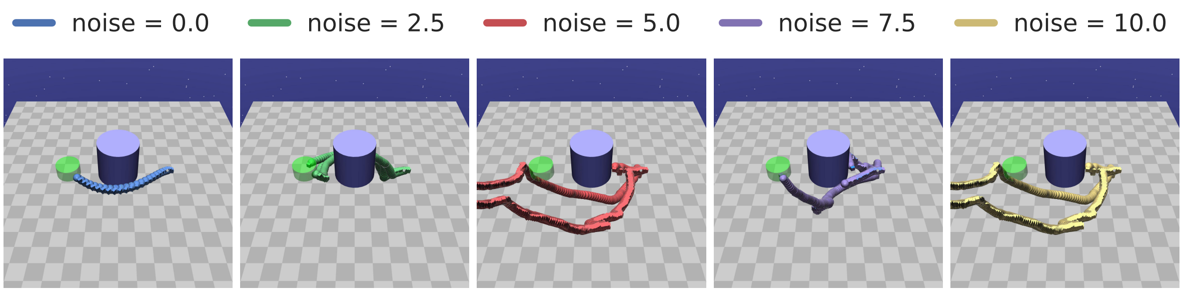
Figure 4.10: Performance of **Sauté-TRPO** in the PILLAR environment with varying noise.



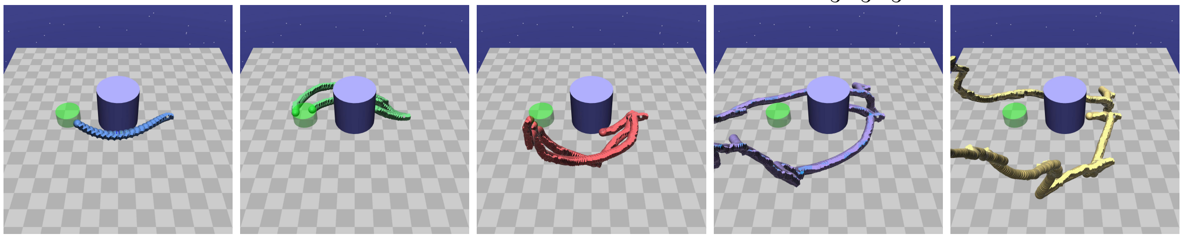
(a) Cumulative cost (b) Episode length (c) Failure rate (d) Average returns

Figure 4.11: Performance of **TRPO-Minmax** in the PILLAR environment with varying noise.

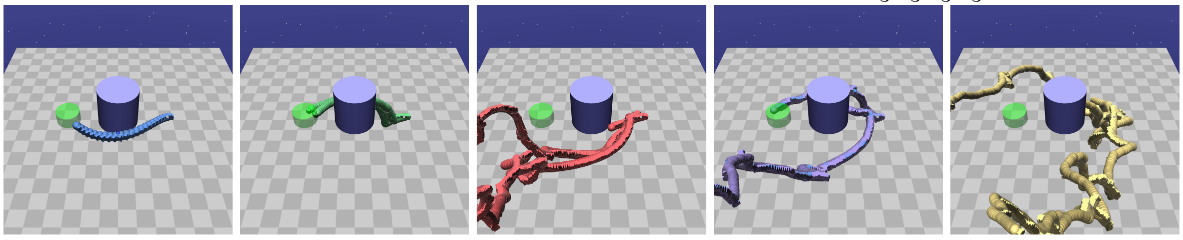
Figures 4.7-4.9 and sample trajectories in Figure 4.12 for the stochastic cases $noise > 0$, we can also conclude that all the baselines have learned risky policies—maximise rewards over short trajectories that are highly likely to result in collisions. Interestingly, SautéTRPO is the worst-performing of all the baselines (Figure 4.10. It successfully maximises returns while minimising cost only for the deterministic environment ($noise = 0$), but completely fails for the stochastic ones ($noise > 0$). By comparison, the results obtained in Figure 4.11 show TRPO-Minmax successfully maximising returns while minimising cost for both deterministic and stochastic environments. In addition, when the noise level is too high ($noise > 2.5$), TRPO-Minmax consistently prioritises maintaining low failure rates over maximising returns.



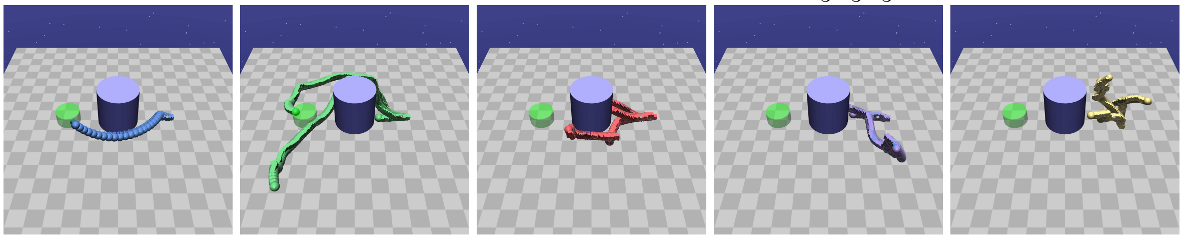
(a) **TRPO**. Failures per noise left to right: $0, 0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$



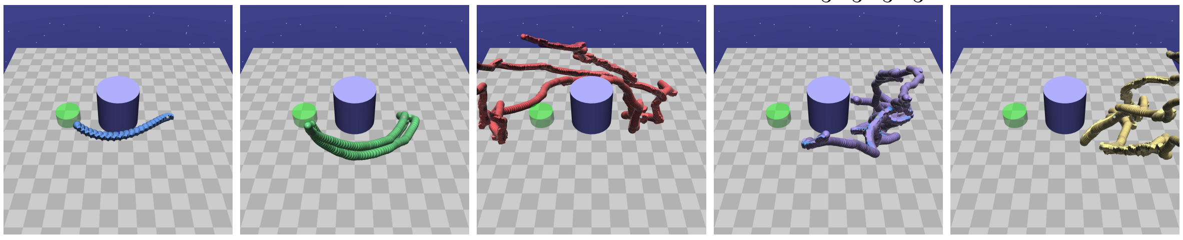
(b) **TRPO-Lagrangian**. Failures per noise left to right: $0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$



(c) **CPO**. Failures per noise left to right: $0, 0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$



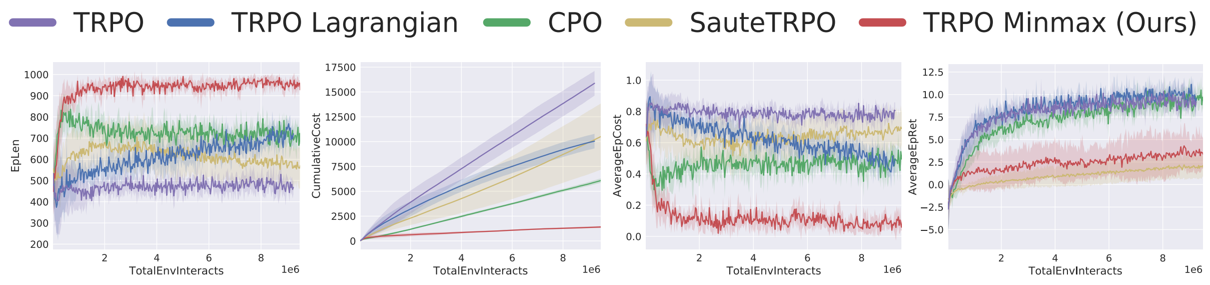
(d) **Sauté-RL**. Failures per noise left to right: $0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}$



(e) **TRPO-Minmax**. Failures per noise left to right: $0, 0, 0, \frac{1}{3}, 0$

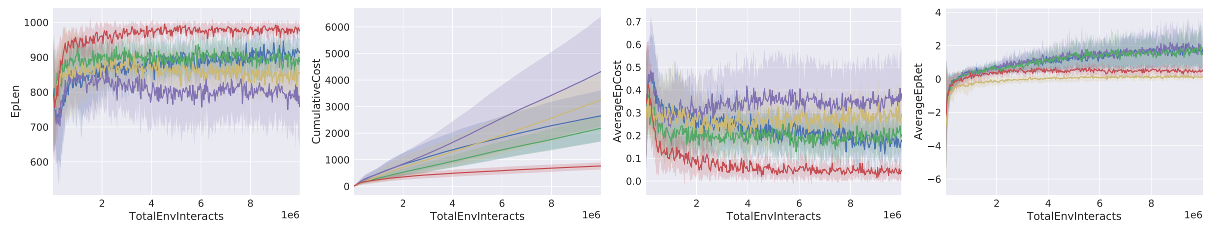
Figure 4.12: Sample trajectories of policies learned by each baseline and our **TRPO-Minmax** approach in the Safety Gym PILLAR environment with varying noise levels. To sample the trajectories for each noise level, we use the same three environment random seeds across all the algorithms.

POINTGOAL1-HARD Results (Figures 4.13-4.16) Unlike the previous PILLAR results, the baselines here achieve significantly different cost and reward performance. We assume this is due to the high variance in the dynamics from random goal positions and unsafe states. However, similar to the PILLAR results, they all achieve significantly high returns at the expense of a rapidly increasing cumulative cost. These results are also consistent with the benchmarks of Ray *et al.* [2019] where the cumulative cost of TRPO is much greater than that of TRPO-Lagrangian, which is greater than that of CPO. By comparison, TRPO-Minmax dramatically reduces the failure rate while still being able to solve the task, as observed by average returns achieved as well as the trajectories observed (Figure 4.17). However, returns are lower due to the dense reward function that incentivises moving towards the goal despite the large number of hazards in-between.



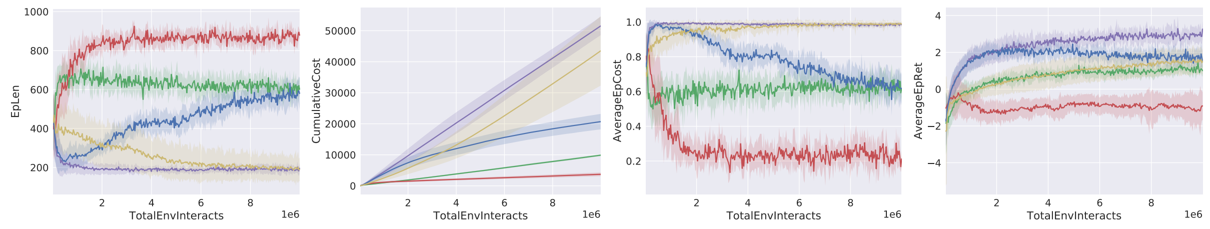
(a) Episode length. (b) Cumulative cost. (c) Failure rate (d) Average returns

Figure 4.13: Comparison with baselines in the POINTGOAL1-HARD environment.



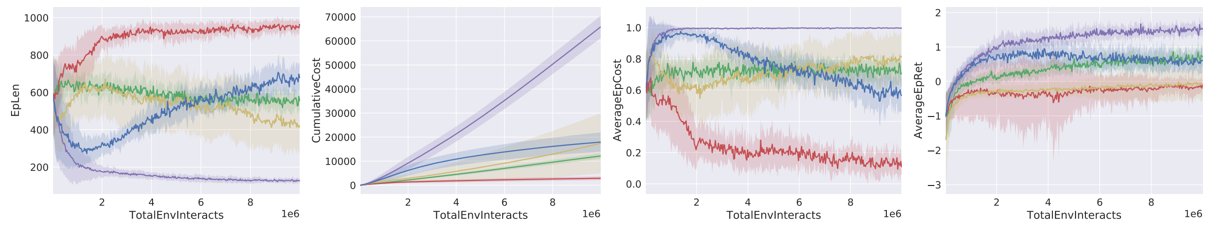
(a) Episode length. (b) Cumulative cost. (c) Failure rate (d) Average returns

Figure 4.14: Comparison with baselines in the POINTPUSH1-HARD environment.



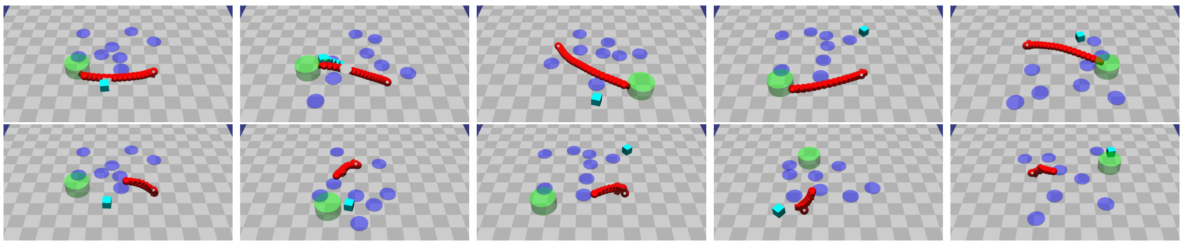
(a) Episode length. (b) Cumulative cost. (c) Failure rate (d) Average returns

Figure 4.15: Comparison with baselines in the POINTBUTTON1-HARD environment.

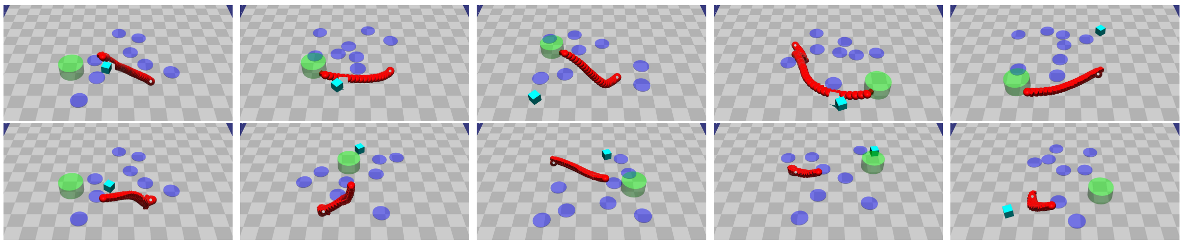


(a) Episode length. (b) Cumulative cost. (c) Failure rate (d) Average returns

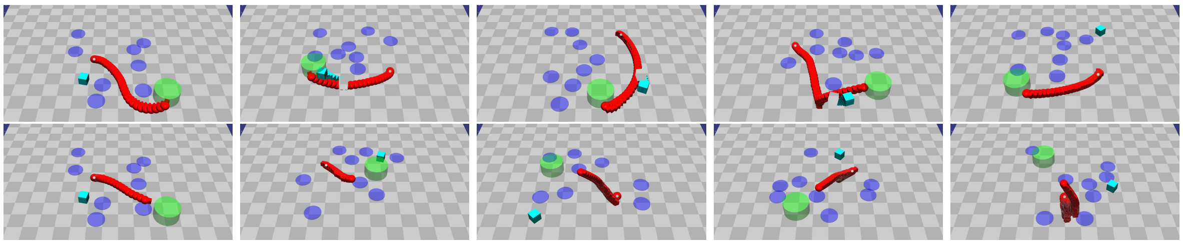
Figure 4.16: Comparison with baselines in the CARBUTTON1-HARD environment.



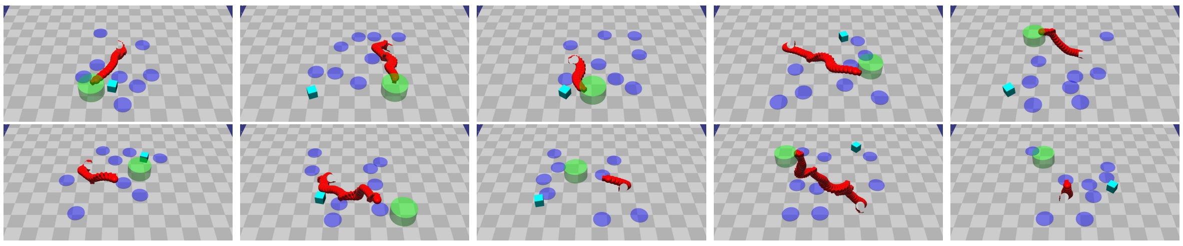
(a) **TRPO** successes (top) and failures (bottom)



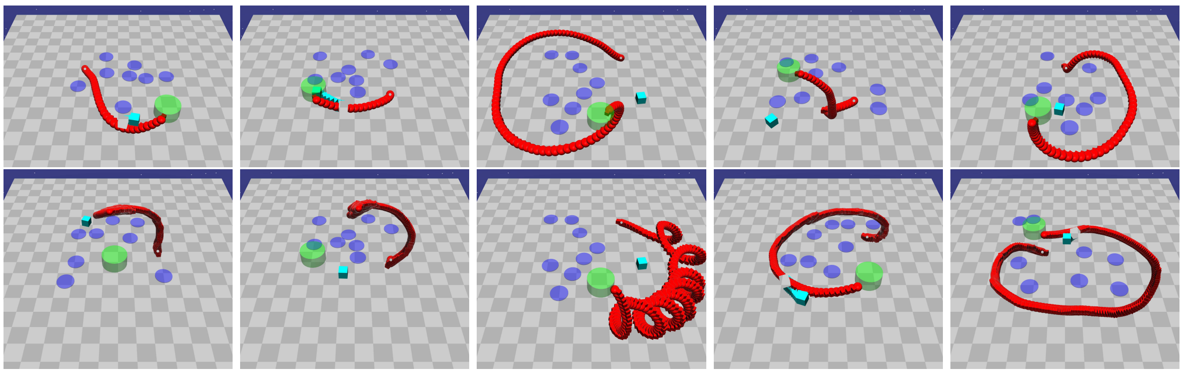
(b) **TRPO-Lagrangian** successes (top) and failures (bottom)



(c) **CPO** successes (top) and failures (bottom)



(d) **Sauté-RL** successes (top) and failures (bottom)



(e) **TRPO-Minmax** successes (top) and failures (bottom)

Figure 4.17: Sample trajectories of policies learned by each baseline and our Minmax approach in the Safety Gym POINTGOAL1-HARD domain, in the experiments of Figures 4.13-4.16. Trajectories that hit hazards or take more than 1000 timesteps to reach the goal location are considered failures.

4.4 Related works

Reward shaping: The problem of designing reward functions to produce desired policies in RL settings is well-studied [Singh *et al.* 2009]. Particular focus has been placed on the practice of *reward shaping*, in which an initial reward function provided by an MDP is augmented in order to improve the rate at which an agent learns the same optimal policy [Ng *et al.* 1999; Devidze *et al.* 2021]. While sacrificing some optimality, other approaches like Lipton *et al.* [2016] propose shaping rewards using an idea of intrinsic fear. Here, the agent trains a supervised fear model representing the probability of reaching unsafe states in a fixed horizon, scales said probabilities by a fear factor, and then subtracts the scaled probabilities from Q-learning targets. These approaches differ from ours in that they seek to find reward functions that improve convergence while preserving the optimality from an initial reward function. In contrast, we seek to determine the optimal rewards for terminal states in order to minimise undesirable behaviours irrespective of the original reward function and optimal policy.

Constrained RL: Disincentivising or preventing undesirable behaviours is core to the field of safe RL. A popular approach is to define constraints on the behaviour of an agent, tasking the agent with limiting the accumulation of costs associated with violating safety constraints while simultaneously maximising reward [Altman 1999; Achiam *et al.* 2017; Chow *et al.* 2018; Ray *et al.* 2019; HasanzadeZonuzy *et al.* 2021]. Widely used examples of these approaches include constrained policy optimisation (CPO) [Achiam *et al.* 2017], which augments TRPO [Schulman *et al.* 2015] with constraints to satisfy a constrained MDP, and TRPO-Lagrangian [Ray *et al.* 2019], which combines Lagrangian methods with TRPO. Another example is Sauté RL [Sootla *et al.* 2022], which incorporates the cost function into the rewards and augments the state with the remaining ”cost budget” spent by violating safety constraints. Other constraint-based approaches include Projection-based CPO [Yang *et al.* 2020], which projects a TRPO policy onto a space defined by constraints, and PID Lagrangian methods [Stooke *et al.* 2020], which augment Lagrangian methods with PID control.

Shielding: Another important line of work involves relying on interventions from a model [Dalal *et al.* 2018; Wagener *et al.* 2021] or human [Tennenholtz *et al.* 2022] to prevent unsafe actions from being considered by the agent (shielding the agent) or prevent the environment from executing those unsafe actions by correcting them (shielding the environment). Other approaches here also look at using temporal logics to define or enforce safety constraints on the actions considered or selected by the agent [Alshiekh *et al.* 2018]. These approaches fit seamlessly into our proposed reward-only framework since they are primarily about modifications on the transition dynamics and not the reward function—for example, unsafe actions here can simply lead to unsafe goal states.

4.5 Conclusion

This chapter investigates a new approach towards safe RL by asking the question: *Is a scalar reward enough to solve tasks safely?* To answer this question, we bound the Minmax penalty, which takes into account the diameter and controllability of an environment in order to minimise the probability of encountering unsafe states. We prove that the penalty does indeed minimise this probability, and present a method that uses an agent's value estimates to learn an estimate of the penalty. Our results in tabular and high-dimensional continuous settings have demonstrated that, by encoding the safe behaviour directly in the reward function via the Minmax penalty, agents are able to solve tasks while prioritising safety, learning safer policies than popular constraint-based approaches. Our method is also easy to incorporate with any off-the-shelf RL algorithms that maintain value estimates, requiring no changes to the algorithms themselves. By autonomously learning the penalty, our method also alleviates the need for a human designer to manually tweak rewards or cost functions to elicit safe behaviour. While it may be feasible to handcraft reward or cost functions to induce safe behaviour for individual tasks, our ultimate aim is to have general agents capable of operating safely in a variety of environments, and thus we cannot rely on human-crafted reward or cost functions.

Finally, while we show that scalar rewards are indeed enough for safe RL, the current analysis is only applicable to unsafe terminal states—which only covers tasks that can be naturally represented by stochastic-shortest path MDPs. Given that other popular RL settings like discounted MDPs can be converted to stochastic shortest path MDPs [Bertsekas 1987; Sutton and Barto 1998], a promising future direction could be to find the dual of our results for other theoretically equivalent settings. In conclusion, we see this reward-only approach as a promising direction towards truly autonomous agents capable of independently learning to solve tasks safely.

Part II

Flexible Agents

In Chapter 5, we introduce *world value functions*, a new knowledge representation for RL agents that provably leads to the ability to solve any given goal-reaching task in the environment. We then show in Chapter 6 that zero-shot skill composition now holds for goal-reaching tasks with potentially stochastic dynamics. Finally, we show in Chapter 7 that this leads to agents that are provably sample efficient in lifelong RL.

Chapter 5

World value functions

This chapter is based on the published work
"World Value Functions: Knowledge representation for multitask reinforcement learning"
[Nangue Tasse et al. 2022a] and the peer-reviewed work
"World Value Functions: Knowledge Representation for Learning and Planning" [Nangue Tasse et al. 2022b], both in collaboration with Steven James and Benjamin Rosman.

In RL, tasks are specified through a reward function from which the agent receives feedback. Most commonly, an agent represents its knowledge in the form of a value function, representing the sum of future rewards it expects to receive. However, since the value function is directly tied to one single reward function (and hence task), it is definitionally insufficient for constructing agents capable of solving a wide range of tasks.

Hence, for an agent to be capable of solving new tasks without additional learning, it needs to have gained sufficient information from its experience when learning to solve previous tasks. Thus, its goal during learning should not be to learn an optimal policy or standard value function, since it only encodes how to maximise the current task rewards. Instead, it may need to learn an optimal general value function (GVF) that encodes how to maximise both the current task rewards as well as all task rewards in a task space—GVFs are simply sets of standard value functions corresponding to some set of MDPs [Sutton *et al.* 2011]. Precisely, it may need to learn $|\mathcal{M}|$ value functions. However, this is clearly impractical since the number of tasks is potentially infinite ($\mathcal{M}_{\mathbb{R}}$ for example).

In this chapter, we seek to overcome this limitation by proposing *world value functions* (WVFs), a goal-oriented knowledge representation that encodes how to achieve not only the current task goals, but also the goals of any other *goal-reaching task*. Precisely, WVFs are a form of GVF defined by $|\mathcal{G}| \leq |\mathcal{S}|$ value functions (where $\mathcal{G}$ is the goal space). We make the following main contributions in this chapter:

(i) **World value functions (Section 5.1.2):** We formally introduce WVFs, which extend the goal-oriented value functions introduced by Nangue Tasse *et al.* [2020b] (EVFs) to more general goal-reaching tasks. Importantly, we show that WVFs can be learned from a single stream of experience; no additional information or modifications to the standard RL framework are required.

(ii) **Mastery (Section 5.1.1):** In the literature, agents that can achieve every goal in their environment are said to possess *mastery* [Veeriah *et al.* 2018]. We prove that WVFs do, in fact, possess this property in goal-reaching tasks. Importantly, given a learned WVF, this property can be leveraged to solve any new task by just estimating its reward function, which reduces the RL problem to supervised learning. We also demonstrate these experimentally, showing how an agent learns not only how to achieve the given task goals, but also how to achieve all the other goals in the environment.

(iii) **Planning (Section 5.1.4):** We show that WVFs implicitly encode the dynamics of the world and can be used for model-based RL. Experimental results in the Four Rooms domain [Sutton *et al.* 1999] validate our theoretical findings, while demonstrating that not only can WVFs be learned faster than regular value functions, they can also be leveraged to perform Dyna-style planning [Sutton 1990] to improve sample efficiency.

5.1 Theory

We first define goal-reaching tasks as a generalisation of the shortest path tasks considered in prior zero-shot composition work [van Niekerk *et al.* 2019; Nangue Tasse *et al.* 2020a]—and significantly relax Assumption 2.3:

Definition 5.1. *We define goal-reaching tasks as tasks in a deterministic environment, where the rewards across tasks differ only at terminal transitions (transitions into absorbing states), and there exists an optimal policy that reaches a terminal transition with non-zero probability.*

Examples of such goal-reaching tasks are discounted tasks ($\gamma \in [0, 1)$) with zero non-terminal rewards—such as the bin-packing grid-world examples—and undiscounted tasks ($\gamma = 1$) with strictly negative non-terminal rewards [van Niekerk *et al.* 2019; Nangue Tasse *et al.* 2020a]—such as the Four Rooms example.

Now let $M = (\mathcal{S}, \mathcal{A}, P, R_M, \gamma)$ be a task from some task space $\mathcal{M}$ with absorbing space $\mathcal{G}$. We first define the internal goal space $\mathcal{G} \subseteq \mathcal{S}$ of an agent as all states where it experiences a terminal transition. Different from other goal-conditioned approaches where goals are specified by the environment, here the goal an agent wishes to achieve is chosen by itself. The agent's aim now is to simultaneously solve the current task, while also learning how to achieve its own internal goals. To do so, the agent can define its own goal-conditioned reward function $\mathbf{R}_M$, which extends R_M to penalise itself for achieving goals it did not intend to:

Definition 5.2. *For a task M with reward function R_M bounded by $[R_{MIN}, R_{MAX}] \subset \mathbb{R}$, the extended reward function $\mathbf{R}_M : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is given by*

$$\mathbf{R}_M(s, g, a, s') := \begin{cases} \mathbf{R}_{MIN} & \textit{if } s' \textit{ is absorbing and } s \neq g, \\ R_M(s, a, s') & \textit{otherwise}, \end{cases} \tag{5.1}$$

where $\mathbf{R}_{MIN}$ is the lower bound we derived for the Minmax penalty in Chapter 4.

This new reward function represents the idea that if an agent terminates in a state ($s \in \mathcal{G}$) that is not the goal state it was trying to reach ($s \neq g$), it should receive the smallest reward possible ($\mathbf{R}_{\text{MIN}}$). Intuitively, the penalty $\mathbf{R}_{\text{MIN}}$ adds one bit of information to the agent's rewards, and we will later prove this is sufficient for the agent to learn the value of achieving its internal goals in a goal-reaching task. The agent must now compute a *world Markov policy* $\pi : \mathcal{S} \times \mathcal{G} \to Pr(\mathcal{A})$ that optimally reaches any reachable goal. A given world policy π is characterised by a *world value function* defined as follows:

Definition 5.3. *For a task M, the world value function* $\mathbf{Q}^{\pi}_{M} : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ *is given by*

$$\mathbf{Q}^{\pi}_{M}(s, g, a) := \mathbb{E}^{\pi}_{s}\left[\mathbf{R}_M(s, g, a, s') + \sum_{t=1}^{\infty} \gamma^t \mathbf{R}_M(s_t, g, a_t, s_{t+1})\right]. \tag{5.2}$$

This specifies the expected return obtained by executing a from s, and thereafter following π to reach g. Since for each g these WVFs are equivalent to standard value functions, it follows that all standard results on standard value functions also hold for WVFs by extension. This can be shown by simply noting that each $g \in \mathcal{G}$ corresponds to a well defined MDP $M_g := (\mathcal{S}, \mathcal{A}, P, R_g, \gamma)$ with reward function $R_{M_g}(s, a) := \mathbf{R}_M(s, g, a)$. In particular, we have that there exists an optimal deterministic world policy π^*, and unique optimal WVF $\mathbf{Q}^*_M$, such that $\mathbf{Q}^*_M(s, g, a) := \mathbf{Q}^{\pi^*}_M(s, g, a) = \max_\pi \mathbf{Q}^{\pi}_M(s, g)$.

Similarly to standard value functions, we want to still be able to extract the optimal policy that solves the current task by acting greedily over $\mathbf{Q}^*_M$: $\pi^*_M(s) \in \arg\max_a \max_g \mathbf{Q}^*_M(s, g, a)$. Theorem 5.1 shows that this is possible, at least in the case of goal-reaching tasks.

Theorem 5.1. *Let* R_M*,* $\mathbf{R}_M$*,* Q^*_M *and* $\mathbf{Q}^*_M$ *be the standard reward function, extended reward function, optimal value function and optimal world value function respectively for a goal-reaching task* M*. Then for all* (s, a) *in* $\mathcal{S} \times \mathcal{A}$*, we have*

$$R_M(s, a, s') = \max_{g \in \mathcal{G}} \mathbf{R}_M(s, g, a, s') \textit{ and } Q^*_M(s, a) = \max_{g \in \mathcal{G}} \mathbf{Q}^*_M(s, g, a).$$

Proof. We will omit the task subscript M for better readability. First note that

$$\max_{g \in \mathcal{G}} \mathbf{R}(s, g, a, s') = \begin{cases} \max\{\mathbf{R}_{\text{MIN}}, R(s, a, s')\}, & \text{if } s' \text{ is absorbing} \\ R(s, a, s'), & \text{otherwise.} \end{cases} \tag{5.3}$$

$$= R(s, a, s'). \tag{5.4}$$

Now define

$$Q^*_{max}(s, a) := \max_{g \in \mathcal{G}} \mathbf{Q}^*(s, g, a).$$

Then if $s' \sim P(\cdot|s,a)$ is not absorbing, it follows that

$$
\begin{aligned}
&[\mathcal{T}Q^*_{max}](s,a,s') = R(s,a,s') + \gamma \sum_{s'' \in \mathcal{S}} P(s''|s,a) \max_{a' \in \mathcal{A}} Q^*_{max}(s',a') \\
&= R(s,a,s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} \left[\max_{g \in \mathcal{G}} \mathbf{Q}^*(s',g,a') \right] \\
&= R(s,a,s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{g \in \mathcal{G}} \left[\max_{a' \in \mathcal{A}} \mathbf{Q}^*(s',g,a') \right] \\
&= R(s,a,s') + \max_{g \in \mathcal{G}} \left[\gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} \mathbf{Q}^*(s',g,a') \right] \\
&\quad \text{(Since } \gamma \geq 0 \text{ and the dynamics are deterministic)} \\
&= \max_{g \in \mathcal{G}} \mathbf{R}(s,g,a,s') + \max_{g \in \mathcal{G}} \left[\gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} \mathbf{Q}^*(s',g,a') \right] \quad \text{(Using Equation 5.4)} \\
&= \max_{g \in \mathcal{G}} \left[\mathbf{R}(s,g,a,s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a' \in \mathcal{A}} \mathbf{Q}^*(s',g,a') \right], \\
&\text{since } \mathbf{R}(s,g,a,s') = 0 \text{ and } p(s',a,s'') = 1 \text{ with } \mathbf{Q}^*(s'',g,a') = 0 \ \text{ if } s'' \text{ is absorbing }. \\
&= \max_{g \in \mathcal{G}} \mathbf{Q}^*(s,g,a) \\
&= Q^*_{max}(s,a).
\end{aligned}
$$

Hence Q^*_{max} is a fixed point of the Bellman optimality operator.

If $s' \sim p(\cdot|s,a)$ is absorbing, then

$$
Q^*_{max}(s,a) = \max_{g \in \mathcal{G}} \mathbf{Q}^*(s,g,a) = \max_{g \in \mathcal{G}} \mathbf{R}(s,g,a,s') = R(s,a,s') = Q^*(s,a).
$$

Since $Q^*_{max} = Q^*$ holds in $\mathcal{G}$ and Q^*_{max} is a fixed point of $\mathcal{T}$, then $Q^*_{max} = Q^*$ holds everywhere.

□

Theorem 5.1 is critical: despite changing the standard RL objective (the standard reward function, value function, and policy), an agent can always recover these original objects, and can also solve the current task by simply maximising over goals. WVFs are therefore a strict generalisation of regular value functions for this type of task. However, we note that this is not necessarily true for all other types of tasks. For example, consider a stochastic MDP with three goal states g_1, g_2, g_3, an initial state w with three actions a_1, a_2, a_3, non-terminal rewards of 0, and goal rewards of $1, 1, 0$ respectively. Table 5.1 shows the transition probabilities, WVF at each goal, the maximisation over the WVF, and the regular value function for each action in state W. Clearly for this task, acting greedily over the WVF is suboptimal compared to the standard value function.

Actions	$P(g_1)$	$P(g_2)$	$P(g_3)$	$\mathbf{Q}^*_M(g_1)$	$\mathbf{Q}^*_M(g_2)$	$\mathbf{Q}^*_M(g_3)$	$\max_g \mathbf{Q}^*_M(g)$	Q^*
a_1	0.6	0.2	0.2	0.6	0.2	0	0.6	0.8
a_2	0.2	0.6	0.2	0.2	0.6	0	0.6	0.8
a_3	0.5	0.5	0.0	0.5	0.5	0	0.5	1

Table 5.1: Counter example in a stochastic MDP. We omit s and a in the notation for clarity.

5.1.1 Mastery with WVFs

Having established WVFs as a task-specific general value function, we next prove in Theorem 5.2 that those of goal-reaching tasks have mastery—that is, they learn the value of reaching all achievable goal states in the world. We define mastery as follows:

Definition 5.4. *Let $\mathbf{Q}^*_M$ be the optimal world value function for a task M in $\mathcal{M}$. Then $\mathbf{Q}^*_M$ has mastery if for all $g \in \mathcal{G}$ reachable from $s \in \mathcal{S} \setminus \mathcal{G}$, there exists an optimal world policy $\pi^*(s,g) \in \arg\max_{a \in \mathcal{A}} \mathbf{Q}^*_M(s,g,a)$ that maximises the probability of reaching g from s.*

Theorem 5.2. *For all goal-reaching tasks M, $\mathbf{Q}^*_M$ has mastery.*

Proof. Let each g in $\mathcal{G}$ define an MDP M_g with reward function

$$R_g(s,a,s') := \mathbf{R}_M(s,g,a,s')$$

for all (s,a) in $\mathcal{S} \times \mathcal{A}$. By Definition 5.1, there exists an optimal policy that reaches a terminal transition. Let

$$\pi^*_g(s) \in \arg\max_{a \in \mathcal{A}} Q^*_g(s,a) \text{ for all } s \in \mathcal{S}.$$

be such a policy. If g *is* reachable from $s \in \mathcal{S} \setminus \{g\}$, then we show that following π^*_g must reach g. Assume π^*_g reaches a different goal state g', with $g' \neq g$. Let π_g be a policy that produces the shortest trajectory to g. Also let $G^{\pi^*_g}$ and G^{π_g} be the returns for the respective policies. Then,

$$G^{\pi^*_g} \geq G^{\pi_g}$$
$$\implies G^{\pi^*_g}_{T-1} + R_g(g', \pi^*_g(g'), s') \geq G^{\pi_g},$$

where $G^{\pi^*_g}_{T-1} = \sum_{t=0}^{T-1} \gamma^t R_{M_g}(s_t, \pi^*_g(s_t), s_{t+1})$ and T is the time at which g' is reached.

$$\implies G^{\pi^*_g}_{T-1} + \mathbf{R}_{\text{MIN}} \geq G^{\pi_g}, \text{ since } g \neq g' \in \mathcal{G}$$
$$\implies \mathbf{R}_{\text{MIN}} \geq G^{\pi_g} - G^{\pi^*_g}_{T-1}$$
$$\implies (R_{\text{MIN}} - R_{\text{MAX}})D \geq G^{\pi_g} - G^{\pi^*_g}_{T-1}, \text{ by definition of } \mathbf{R}_{\text{MIN}}$$
$$\implies G^{\pi^*_g}_{T-1} - R_{\text{MAX}}D \geq G^{\pi_g} - R_{\text{MIN}}D, \text{ since } G^{\pi_g} \geq R_{\text{MIN}}D$$
$$\implies G^{\pi^*_g}_{T-1} - R_{\text{MAX}}D \geq 0$$

$$\implies G^{\pi^*_g}_{T-1} \geq R_{\text{MAX}} D.$$

But this is a contradiction, since the result obtained by following an optimal trajectory up to a terminal state without the reward for entering the terminal state must be strictly less than receiving R_{MAX} for every step of the longest possible optimal trajectory. Hence we must have $g' = g$.

□

These results are important as they show that a WVF encodes the optimal policy for the current task (Theorem 5.1)—meaning that we can henceforth focus only WVFs—while also encoding the optimal policy for achieving any goal in the environment and the value of achieving said goals in the current task (Theorem 5.2)—which will later be useful for zero-shot composition.

5.1.2 Multitask transfer with WVFs

We now show the advantage of WVFs for multitask transfer under the assumption that an agent may be faced with solving several goal-reaching tasks $\mathcal{M}$ within the same environment $(\mathcal{S}, \mathcal{A}, P)$. One immediate result is that if goal-reaching tasks share the same environment, then their WVFs share the same world policy. That is, the agent has the same notion of goals and how to reach them, regardless of the current goal-reaching task. Similarly, if we require that the world policies be the same across goal-reaching tasks, then we have that the goal-reaching tasks must come from the same world. This is formalised by Theorem 5.3 below.

Theorem 5.3. *Let $\mathcal{Q}^*$ be the set of optimal world action-value functions with mastery of goal-reaching tasks in $\mathcal{M}$. Then for all $s \neq g \in \mathcal{S} \times \mathcal{G}$,*

$$\pi^*(s,g) \in \arg\max_{a \in \mathcal{A}} \mathbf{Q}^*_{M_1}(s,g,a)$$
$$\iff$$
$$\pi^*(s,g) \in \arg\max_{a \in \mathcal{A}} \mathbf{Q}^*_{M_2}(s,g,a) \ \forall M_1, M_2 \in \mathcal{M}.$$

Proof. Let $g \in \mathcal{G}, s \in \mathcal{S} \setminus \{g\}$.

If g *is* reachable from s, then we are done since $\mathbf{Q}^*_{M_1}$ and $\mathbf{Q}^*_{M_2}$ have mastery (Theorem 5.2).

If g is unreachable from s, then for all (a, s') in $\mathcal{A} \times \mathcal{S}$ we have

$$\mathbf{R}_{M_1}(s,g,a,s') = \begin{cases} \mathbf{R}_{\text{MIN}}, & \text{if } s' \text{ is absorbing} \\ r_{s,a,s'}, & \text{otherwise} \end{cases}$$
$$\text{where } r_{s,a,s'} \text{ is the reward for the non-terminal transition } (s,a,s')$$
$$= \mathbf{R}_{M_2}(s,g,a,s')$$
$$\implies \mathbf{Q}^*_{M_1}(s,g,a) = \mathbf{Q}^*_{M_2}(s,g,a).$$

□

We can think of the world as a background MDP $M_0 = (\mathcal{S}_0, \mathcal{A}_0, P_0, R_0)$ with its own state space, action space, transition dynamics and background reward function. Any individual goal-reaching task M is defined by a reward function $R^\tau_M(s,a)$ that is non-zero only for transitions entering terminal states. The reward function for the resulting MDP is then simply $R_M(s,a,s') \coloneqq R_0(s,a,s') + R^\tau_M(s,a)$.

Since all goal-reaching tasks in $\mathcal{M}$ share the same dynamics (and consequently the same world policy), their corresponding WVFs can be written as $\mathbf{Q}^*_M(s,g,a) = G^*_{s,g,a} + \mathbf{R}^\tau_M(s',a')$ for some $s', a' \in \mathcal{S} \times \mathcal{A}$, where $G^*_{s,g,a}$ is a constant across goal-reaching tasks that represents the sum of rewards starting from s and taking action a up until g, but not including the terminal reward. Using this fact, Theorem 5.4 shows that the optimal value function and policy for any goal-reaching task can be obtained zero-shot from an arbitrary WVF given the task-specific rewards:

Theorem 5.4. *Let R^τ_M be the given task-specific reward function for a goal-reaching task $M \in \mathcal{M}$, and let $\mathbf{Q}^* \in \mathbfcal{Q}^*$ be an arbitrary WVF. Let $\tilde{\mathbf{V}}_M(s,g)$ be the estimated WVF of M given by*

$$\max_{a\in\mathcal{A}} \mathbf{Q}^*(s,g,a) + \left(\max_{a\in\mathcal{A}} R^\tau_M(g,a) - \max_{a\in\mathcal{A}} \mathbf{Q}^*(g,g,a) \right).$$

Then,

(i) for all $g \in \mathcal{G}$ reachable from $s \in \mathcal{S}$, $\mathbf{V}^*_M(s,g) = \tilde{\mathbf{V}}_M(s,g)$.

(ii) $V^*_M(s) = \max_{g\in\mathcal{G}} \tilde{\mathbf{V}}(s,g)$, and $\pi^*_M(s) \in \arg\max_{a\in\mathcal{A}} \mathbf{Q}^*(s, \arg\max_{g\in\mathcal{G}} \tilde{\mathbf{V}}_M(s,g), a)$.

Proof.

(i): Let $g \in \mathcal{G}$ be a goal reachable from state $s \in \mathcal{S}$. If $g = s$, then

$$\begin{aligned}
&\max_{a\in\mathcal{A}} \mathbf{Q}^*(s,g,a) + \left(\max_{a\in\mathcal{A}} R^\tau_M(g,a) - \max_{a\in\mathcal{A}} \mathbf{Q}^*(g,g,a) \right) \\
&= \max_{a\in\mathcal{A}} R^\tau(g,a) + \left(\max_{a\in\mathcal{A}} R^\tau_M(g,a) - \max_{a\in\mathcal{A}} R^\tau(g,a) \right) \\
&= \max_{a\in\mathcal{A}} R^\tau_M(g,a) = \mathbf{V}^*_M(s,g)
\end{aligned}$$

If $g \neq s$, then

$$\begin{aligned}
&\max_{a\in\mathcal{A}} \mathbf{Q}^*(s,g,a) + \left(\max_{a\in\mathcal{A}} R^\tau_M(g,a) - \max_{a\in\mathcal{A}} \mathbf{Q}^*(g,g,a) \right) \\
&= \max_{a\in\mathcal{A}} [G^*_{s,g,a} + R^\tau(g,a^{\max})] + \left(\max_{a\in\mathcal{A}} R^\tau_M(g,a) - \max_{a\in\mathcal{A}} R^\tau(g,a) \right), \\
&\text{follows from Theorem 5.2 and Theorem 5.3} \\
&= \max_{a\in\mathcal{A}} G^*_{s,g,a} + R^\tau(g,a^{\max}) + (R^\tau_M(g,a^{\max}_M) - R^\tau(g,a^{\max})) \\
&= \max_{a\in\mathcal{A}} [G^*_{s,g,a} + \mathbf{R}^\tau_M(g,a^{\max}_M)] = \mathbf{V}^*_M(s,g)
\end{aligned}$$

(ii): Follows directly from (i) above and Theorem 5.3.

□

This has several important implications for transfer learning. Most importantly, an agent can learn an arbitrary WVF with unsupervised pretraining and then solve any new goal-reaching task by simply estimating the reward function (from experience or demonstrations).

5.1.3 Learning WVFs

Since WVFs satisfy the Bellman equations, $\mathbf{Q}^*$ can be learned using any suitable RL algorithm [Kaelbling 1993; Veeriah *et al.* 2018; Andrychowicz *et al.* 2017; Colas *et al.* 2019; Foster and Dayan 2002; Mirowski *et al.* 2017; Moore *et al.* 1999]. Here, we wish to learn WVFs only with rewards per state-action obtained by interacting with the environment. Hence, we propose a simple modification to Q-learning to learn $\mathbf{Q}^*$ from a single stream of experience. The algorithm differs from standard Q-learning in several ways: it keeps track of the set of terminating states seen so far, and at each timestep updates the WVF with respect to both the current state and action, as well as all goals encountered so far. We extend the algorithm to use the definition of the extended rewards for the task-dependent rewards.

Algorithm 6: Q-learning for WVFs

```
Initialise : WVF $\mathbf{Q}(s,g,a)=0$, goal buffer $\mathcal{G}=\{\emptyset\}$
foreach episode do
    Observe initial state $s \in \mathcal{S}$ and sample a goal $g \in \arg\max_g \max_a \mathbf{Q}(s,g,a)$
    while episode is not done do
        $a \leftarrow \begin{cases} \arg\max_{a\in\mathcal{A}} \mathbf{Q}(s,g,a) & \text{w.p. } 1-\varepsilon \\ \text{a random action} & \text{w.p. } \varepsilon \end{cases}$
        Execute $a$, observe reward $r$ and next state $s'$
        if $s'$ is absorbing then $\mathcal{G} \leftarrow \mathcal{G} \cup \{s\}$
        for $g' \in \mathcal{G}$ do
            $\bar{r} \leftarrow \mathbf{R}_{\text{MIN}}$ if $g' \neq s$ and $s \in \mathcal{G}$ else $r$
            $\mathbf{Q}(s,g',a) \overset{\alpha}{\leftarrow} \left(\bar{r} + \max_{a'} \mathbf{Q}(s',g',a')\right) - \mathbf{Q}(s,g',a)$
        $s \leftarrow s'$
```

5.1.4 Planning with WVFs

If the agent's goal space coincides with the state space ($\mathcal{G} = \mathcal{S}$), then an optimal WVF will implicitly encode the dynamics of the world. We can then estimate the transition probabilities for each $s, a \in \mathcal{S} \times \mathcal{A}$ using only the reward function and optimal WVF. That is, $P(s,a,s')$ for all $s' \in \mathcal{S}$ can be obtained by simply solving the system of Bellman optimality equations given by each goal $g \in \mathcal{S}$: $\mathbf{Q}^*(s,g,a) = \sum_{s' \in \mathcal{S}} p(s,a,s') \left[\mathbf{R}(s,g,a,s') + \mathbf{V}^*(s',g)\right]$. In practice, if the transition probabilities are known to be non-zero only in a neighbourhood $\mathcal{N}(s)$ of state s (as is common in most domains), then we only require that the WVF be near-optimal for $s', g \in \mathcal{N}(s) \times \mathcal{N}(s)$.

These inferred dynamics can then be used to improve planning by integrating WVFs into a Dyna-style architecture [Sutton 1990]. Our approach is illustrated in Algorithm 7, where we combine both model-free and model-based updates to learn the WVF. Importantly, since the dynamics are inferred from the WVF, using them to plan (Dyna-style) at the start of training is detrimental, since the WVF will make incorrect predictions. We mitigate this by computing the mean-squared error of the Bellman equations using the inferred next state, $MSE = \frac{1}{|\mathcal{N}(s)|} \sum_{g \in \mathcal{N}(s)} (\mathbf{Q}(s, g, a) - [\mathbf{R}(s, g, a, s') + \mathbf{V}(s', g)])$, and only use the WVF to plan when the error is less than a threshold ($MSE \leq 10^{-5}$).

Algorithm 7: Dyna for WVFs using inferred transition functions

```
Initialise : WVF $\mathbf{Q}(s, g, a) = 0$, Reward function $R(s, a, s') = 0$, goal buffer $\mathcal{G} = \emptyset$
foreach episode do
    Observe initial state $s \in \mathcal{S}$ and sample $g \in \mathcal{G}$
    while episode is not done do
        $a \leftarrow \begin{cases} \arg\max_{a \in \mathcal{A}} \mathbf{Q}(s, g, a) & \text{w.p. } 1 - \varepsilon \\ \text{a random action} & \text{w.p. } \varepsilon \end{cases}$
        Execute $a$, observe reward $r$ and next state $s'$
        $R(s, a, .) \leftarrow r$
        if $s'$ is absorbing then $\mathcal{G} \leftarrow \mathcal{G} \cup \{s\}$
        for $g' \in \mathcal{G}$ do
            $\bar{r} \leftarrow \mathbf{R}_{\text{MIN}}$ if $g' \neq s$ and $s \in \mathcal{G}$ else $r$
            $\delta \leftarrow \left[\bar{r} + \max_{a'} \mathbf{Q}(s', g', a')\right] - \mathbf{Q}(s, g', a)$
            $\mathbf{Q}(s, g', a) \leftarrow \mathbf{Q}(s, g', a) + \alpha\delta$
        $N$ $s \leftarrow$ random previous state
        $a \leftarrow$ random previous action taken in $s$
        $r \leftarrow R(s, a, .)$
        $s' \leftarrow$ Solving $\mathcal{N}(s)$ Bellman equations
        $MSE \leftarrow \frac{1}{|\mathcal{N}(s)|} \sum_{g \in \mathcal{N}(s)} (\mathbf{Q}(s, g, a) - [\mathbf{R}(s, g, a, s') + \mathbf{V}(s', g)])$
        if $MSE \leq$ threshold then
            for $g' \in \mathcal{G}$ do
                $\bar{r} \leftarrow \mathbf{R}_{\text{MIN}}$ if $g' \neq s$ and $s \in \mathcal{G}$ else $r$
                $\delta \leftarrow \left[\bar{r} + \max_{a'} \mathbf{Q}(s', g', a')\right] - \mathbf{Q}(s, g', a)$
                $\mathbf{Q}(s, g', a) \leftarrow \mathbf{Q}(s, g', a) + \alpha\delta$
        $s \leftarrow s'$
```

5.2 Experiments

To empirically validate the properties of WVFs described above, we use:

1. the bin packing domain introduced in Example 3.1. Here, the non-terminal rewards (rewards for all non-terminal states) are 0 and the goal rewards (rewards for terminal states) range from 0 to 1. We use a discount factor of $\gamma = 0.95$.

2. the Four Rooms domain [Sutton *et al.* 1999], where an agent is required to reach various goal positions similar to the bin domain. The agent can move in any of the four cardinal directions at each timestep (with reward -0.1), but colliding with a wall leaves it in the same state. The agent also has a "done" action that can choose to terminate at any position (with reward of 10 if it is the goal of the current task).

For each of the experiments below, we consider the case where the agent's goals are the entire state space ($\mathcal{G} = \mathcal{S}$).

5.2.1 Learning WVFs

To verify that WVFs can be learned with standard model-free algorithms, we train an agent using Q-learning (Algorithm 6) on the task 🟥 in the bin packing domain. Here, the robot must pack all the red objects into the bin. Figure 5.1 shows the learned WVF. The figure is generated by first plotting the value functions for all goal states, then displaying each of them at their respective position in the gridworld representation of the domain. We can observe from the value gradients of the plots that the learned WVF does indeed have mastery as it encodes how to achieve all desirable goals, demonstrating the results proven in Theorem 5.2. Notice how for the goal states corresponding to no red object in the bin (the rightmost column of the plot), the WVF has zero values everywhere since the agent receives no reward at those goals. As we proved in Theorem 5.1, we can also maximise over goals to obtain the standard state-value function and policy as shown in Figures 5.2.

By learning WVFs, an agent learns a large number of diverse solutions to a single task. However, the upfront cost of learning is likely to be higher since we must learn not only the single value function, but rather the value function with respect to every goal. We investigate the sample complexity of learning WVFs (using Algorithm 6) and learning standard value functions (using standard Q-learning) in Figure 5.3. As expected, we observe that the number of samples required

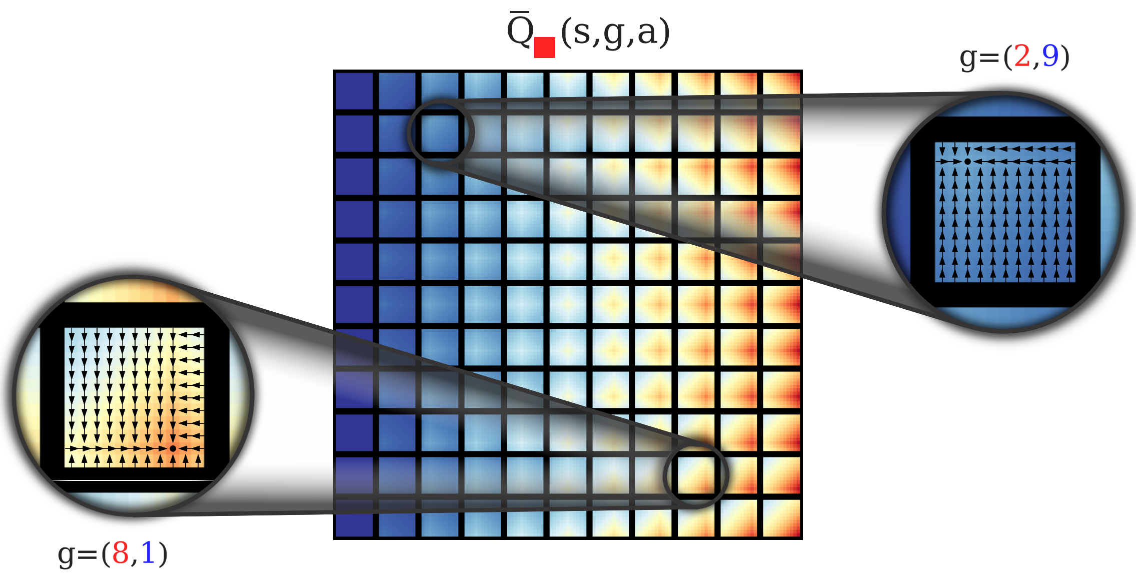


Figure 5.1: Learned WVF for the task of packing all the red objects into the bin. Each square shows the value of all internal states with respect to the goal state at that position.

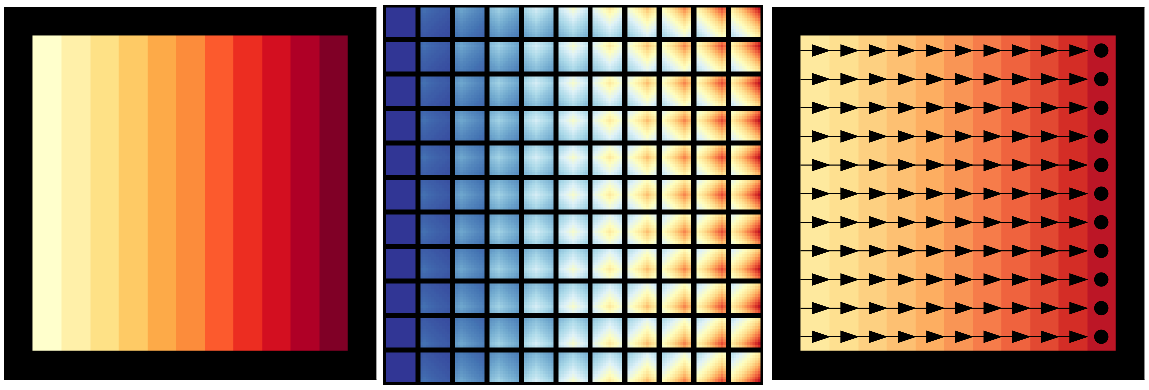
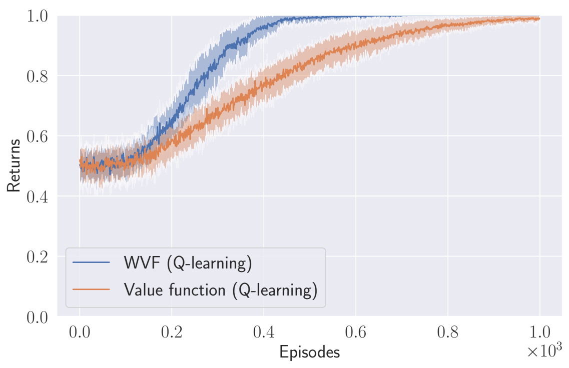
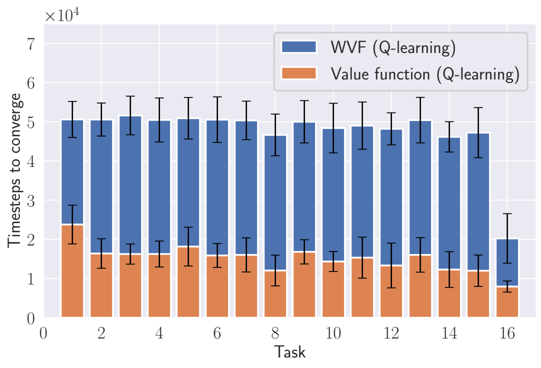


Figure 5.2: Illustrating the standard value functions and policy (right) for the task of packing all the red objects into the bin (left), obtained by maximising over the goal values of the learned WVF $Q^*_{🟥}$ (middle).



(a) Returns during training of the WVF and standard value function for the 🟥 task. Returns are calculated by greedy evaluation at the end of each episode. The shaded regions indicate one standard deviation over 25 random seeds.



(b) Number of samples required to learn optimal WVFs and standard value functions for the 16 tasks in Figure 2.4. Error bars represent standard deviations over 25 random seeds.

Figure 5.3: Sample complexity for training WVFs and standard value functions.

to learn optimal WVFs is greater than that for learning optimal standard value functions (Figure 5.3b). Interestingly, we also observe that it is more sample efficient to learn WVFs if we only care about the performance of the resulting policy instead of the optimality of the Q-values—despite the fact that WVFs have an additional dimension that must be learned (Figure 5.3a). We conjecture that this is due to the induced goal-directed exploration of Algorithm 6, similarly to the results obtained by Kaelbling [1993].

We show in the next section how we can leverage the WVFs to improve transfer in a multi-task setting, which amortises the upfront cost over multiple tasks.

5.2.2 Planning with WVFs

We now use the Four Rooms domain to demonstrate that the transition probabilities can be inferred from the learned WVF and used to improve learning. We compare leveraging the

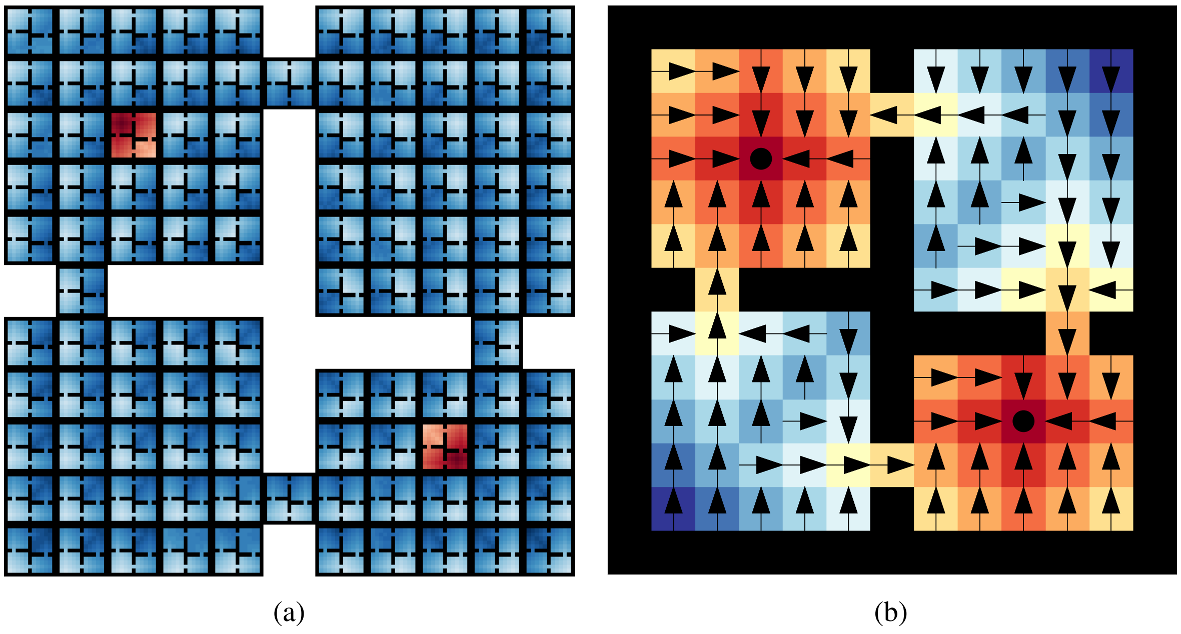


Figure 5.4: (a) Learned WVF. (b) Inferred values and policy for solving the current task.

inferred dynamics while learning (Algorithm 7) to Q-learning for both WVFs and regular value functions, as well as Dyna for regular value functions. We train these four agents on the task where they must learn to navigate to either the middle of the *top-left* or *bottom-right* rooms. Figure 5.4 shows the learned WVF, which is generated by plotting the value functions for every goal position and displaying them at their respective xy positions. The results in Figure 5.5 illustrate that sample efficiency can be greatly improved by integrating the planning capabilities of WVFs.

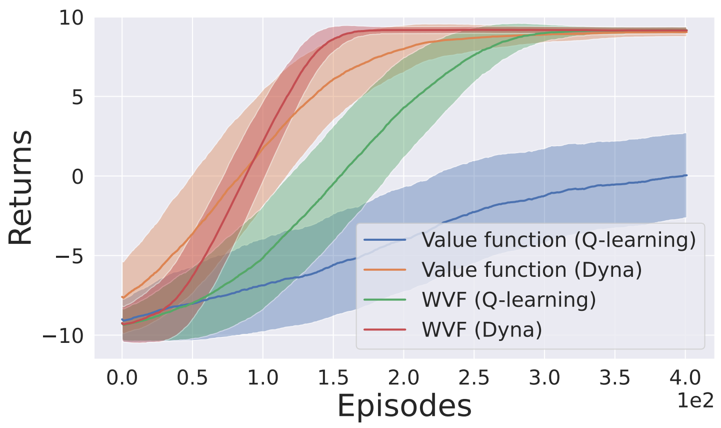


Figure 5.5: Returns during training for both WVFs and regular value functions, with and without planning. The shaded regions indicate one standard deviation over 25 runs.

Figures 5.6 (left) and (middle) respectively show the transitions inferred at the end of Dyna (WVF) Dyna learning. This is done by solving the Bellman equations with $s', g \in \mathcal{S} \times \mathcal{S}$ and $s', g \in \mathcal{N}(s) \times \mathcal{N}(s)$. For each, we infer the next state probabilities for taking each cardinal action at the center of each room, and place the corresponding arrow in the state with highest probability. The red arrows in Figure 5.6a correspond to incorrectly inferred next states, which is a consequence of the learned WVF not being near optimal at all states for all goals. Figure 5.6b shows that in practice, if the WVF is not near-optimal, we can still infer dynamics by using $s', g \in \mathcal{N}(s) \times \mathcal{N}(s)$. Figure 5.6c shows sample trajectories for following the optimal policy using the inferred transition probabilities. The gray-scale color of each arrow corresponds to the normalised value prediction for that state.

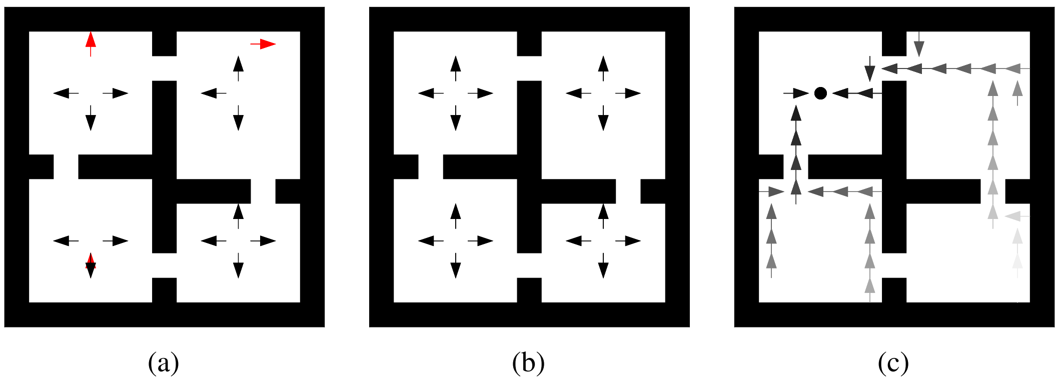


Figure 5.6: (a–b) Inferred one-step transitions. Red arrows indicate incorrect predictions. (c) Imagined rollouts using the learned WVF.

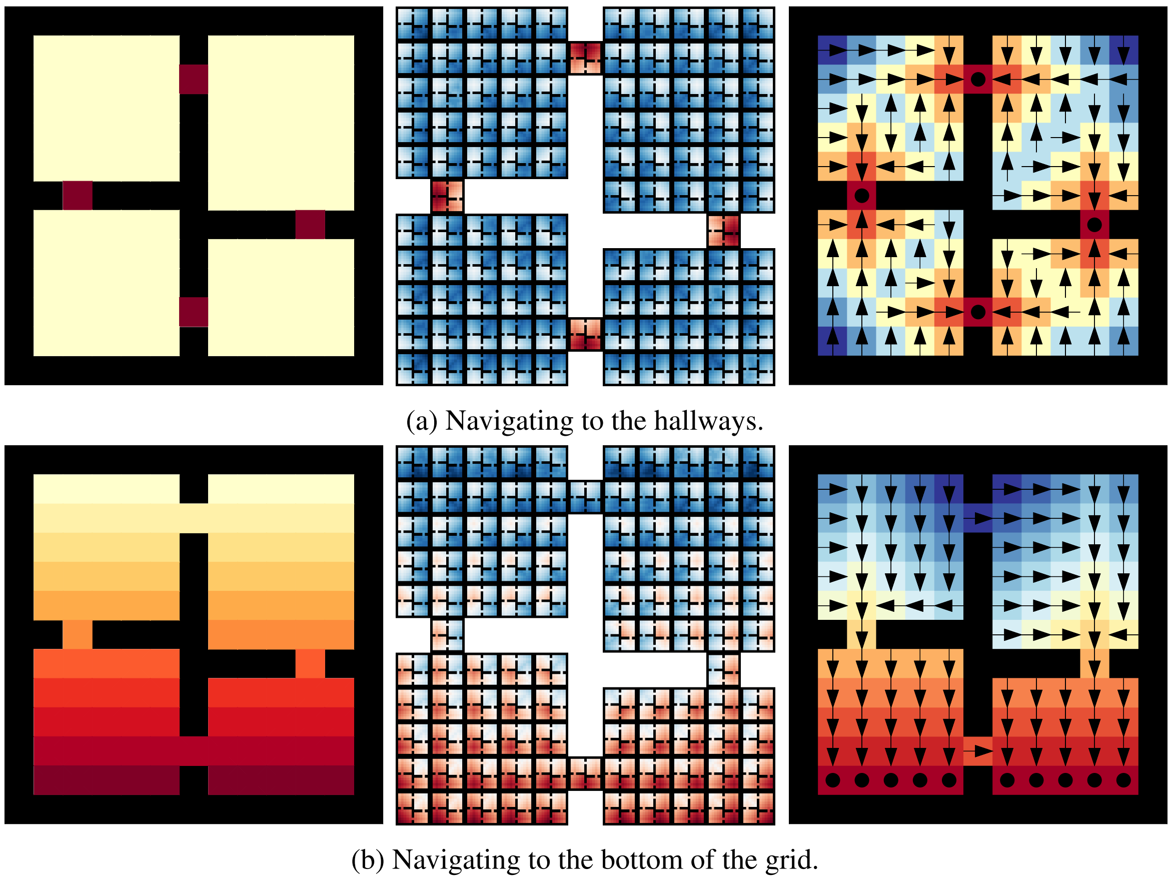


Figure 5.7: Four Rooms domain. From left to right on each figure: The task specific rewards, inferred WVF using Theorem 5.4, and inferred values and policy from maximising over goals.

5.2.3 Multitask transfer with WVFs

Having learned a single WVF in the Four Rooms domain (Figure 5.4), we now show that it can be used to solve subsequent tasks—by combining the learned WVF with the task-specific rewards as per Theorem 5.4. Critically, this means that any new task an agent might face can simply be solved by estimating its reward function, reducing the RL problem to a supervised learning one. Figure 5.7 illustrates the reward functions and subsequent WVFs and policies for two sample tasks. Importantly, given the reward functions (which can be estimated from data), the optimal policies can immediately be computed without further learning.

5.3 Related works

The idea of learning how to achieve all goals in a multi-goal environment from a single stream of experience was first introduced by Kaelbling [1993] with dynamic goal (DG) functions. DG functions encode the distance between states and goals learned by giving an agent minimum rewards at internal states and maximum rewards at boundary states. While lacking in theory, Kaelbling [1993] demonstrated in the tabular case that DG functions learned how to achieve goals significantly faster than regular value functions. Veeriah *et al.* [2018] later used UVFAs to extend these results to the function approximation case, and showed that such goal-oriented value functions were also useful as pre-training and auxiliary knowledge for improving sample efficiency. Another similar work is hindsight experience replay [Andrychowicz *et al.* 2017], where an agent also learns to achieve multiple goals to accelerate the learning of a main task. It is similar to Veeriah *et al.* [2018] in that it also uses UVFAs for function approximation, but differs in that it only considers tasks with a single desired goal per episode which is given upfront to the agent. The agent then learns to achieve the desired goals faster by also learning how to achieve undesirable ones.

In contrast to these approaches, WVFs introduced in this section are a principled generalisation of DG functions to the case of arbitrary task reward functions. Instead of giving an agent minimum rewards at internal states and maximum rewards at boundary states (as in related works [Kaelbling 1993; Veeriah *et al.* 2018]), we use the extended reward function (Equation 5.1), which allows for arbitrary task rewards.

5.4 Conclusion

In this chapter, we addressed the fundamental challenge in reinforcement learning (RL) where traditional approaches, reliant on single reward functions and value functions, limit an agent's ability to generalize across tasks. To tackle this limitation, we introduced world value functions (WVFs) as a novel goal-oriented knowledge representation. Unlike standard value functions, WVFs encode how to achieve not just the current task goals but also those of any other goal-reaching task. Our contributions include the formal introduction of WVFs, demonstrating their capacity for learning from a single stream of experience without additional modifications to the RL framework. We established that agents equipped with WVFs attain mastery, capable of achieving all goals within their environment. Moreover, we showcased WVFs' utility in planning, illustrating their ability to implicitly encode world dynamics and significantly improve sample efficiency in model-based RL scenarios. Through theoretical analysis and empirical validation in environments like the Four Rooms domain, we highlight WVFs' promise in addressing the challenges of task generalization and sample efficiency in RL.

Chapter 6

Logical composition of WVFs

In the previous chapters, we extended the framework of Nangue Tasse *et al.* [2020b] to formalise the logical composition of tasks with arbitrary rewards. We then generalised their extended value functions (EVFs) with world value functions (WVFs), which enabled mastery of arbitrary goal-reaching tasks. Importantly, we showed that WVFs encode sufficient information about learned tasks to solve new ones without further learning, given the reward function of the new tasks to solve. Can a similar result be obtained when only the Boolean expression (instead of the reward function) of a new task is given?

In this chapter we formally show that WVFs of goal-reaching tasks are indeed sufficient to solve the logical composition of tasks zero-shot.[1] We do this by first formalising their composition under the relevant algebraic structures, just as was done with task compositions. This gives us mathematical tools which can be used to formally prove zero-shot composition and further explore some additional properties of the established structures. Hence, we make the following main contributions in this chapter:

(i) **Conjunction and disjunction of WVFs (Section 6.1):** We extend the conjunction and disjunction operators of Nangue Tasse *et al.* [2020b] to the WVFs of arbitrary goal-reaching tasks. We then show that there is a homomorphism between the task and WVF algebra, enabling zero-shot solutions to arbitrary conjunctions and disjunctions of goal-reaching tasks. Even when the WVFs are sub-optimal (due to function approximation for example), these compositions are shown to result in no loss in optimality (Section 6.4).

(ii) **Negation of WVFs (Section 6.2):** Given a De Morgan algebra over goal-reaching tasks, we show that the same negation operator defined by Nangue Tasse *et al.* [2020b] also leads to a De Morgan algebra over WVFs. Similarly to Section 6.1, we show that both algebra are also homomorphic, leading to zero-shot solutions to arbitraty logical compositions of

[1]While we focus on the WVFs of goal-reaching tasks (Definition 5.1), we leave our proofs as general as possible, making it clear which parts actually make use of this constraint. In particular, we note how none of these proofs make use of the deterministic dynamics constraint. This will be particularly useful for our proofs in lifelong RL (Chapter 7), where we show the effect of relaxing this constraint.

goal-reaching tasks. Finally, the results in Section 6.4 show that these compositions only lead to a constant decrease in the sub-optimality of learned WVFs.

(iii) **Propositional logics over WVFs (Section 6.3):** Interestingly, given a Boolean algebra over goal-reaching tasks, we introduce a new definition for the negation of the corresponding WVFs that still leads to a Boolean WVF algebra. Similarly to the Boolean task algebra, we show that this Boolean EVF algebra is isomorphic to the power-set over states (or transitions). This is a significant result, as it will lead to an efficient way of learning WVFs. Finally, and remarkably, Section 6.4 then shows the new negation leads to no loss in optimality when WVFs are sub-optimal.

6.1 WVF lattice

Similarly to how we formalised the disjunction and conjunction of tasks using the lattice algebraic structure, we now formalise the disjunction and conjunction of WVFs. Since WVFs are real valued functions, a natural partial order on them is pointwise $\leq$ (the usual $\leq$ on $\mathbb{R}$). We state the resulting poset formally as follows:

Proposition 6.1. *Let $(\mathcal{M}, \vee, \wedge)$ be a lattice of goal-reaching tasks, and let $\mathcal{Q}^*$ be the set of optimal WVFs of tasks in $\mathcal{M}$. Then $(\mathcal{Q}^*, \leq)$ is a partially ordered set with the relation $\leq$ given by*

$$\mathbf{Q}^*_{M_1} \leq \mathbf{Q}^*_{M_2} \text{ if } \mathbf{Q}^*_{M_1}(s,g,a) \leq \mathbf{Q}^*_{M_2}(s,g,a) \text{ for all } (s,g,a) \in \mathcal{S} \times \mathcal{G} \times \mathcal{A}.$$

Proof. Follows from the usual $\leq$ relation on $\mathbb{R}$. □

Given that all goal-reaching tasks share the same non-terminal rewards, the WVF for each goal state is partially ordered based only on the terminal reward at that goal state (since the terminal reward at the other goal states is set to R_{MIN} by the extended reward function). Hence, since $(\mathcal{M}, \vee, \wedge)$ is a lattice based on the usual $\leq$ over rewards, every pair of optimal WVFs has a supremum and an infimum in $\mathcal{Q}^*$ respectively given by simply applying pointwise sup and inf:

Proposition 6.2. *Let $(\mathcal{M}, \vee, \wedge)$ be a lattice of goal-reaching tasks, and let $\mathcal{Q}^*$ be the set of optimal WVFs of tasks in $\mathcal{M}$. Then for all $M_1, M_2 \in \mathcal{M}$,*

(i) $\sup\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\} = \mathbf{Q}^*_{M_1 \vee M_2} \in \mathcal{Q}^*$, *(ii)* $\inf\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\} = \mathbf{Q}^*_{M_1 \wedge M_2} \in \mathcal{Q}^*$.

Proof. Without loss of generality, let us first assume that

$$\mathbf{R}_{M_1}(g, g, \pi^*_{M_1}(g,g), s') \leq \mathbf{R}_{M_2}(g, g, \pi^*_{M_2}(g,g), s')$$

for some goal $g \in \mathcal{G}$ and next state s'. Then,

$$\mathbf{R}_{M_1}(g, g, \pi^*_{M_1}(g, g), s') \leq \mathbf{R}_{M_2}(g, g, \pi^*_{M_2}(g, g), s') \tag{6.1}$$

$$\implies \mathbf{R}_{M_1}(s, g, \pi^*_{M_1}(s, g), s') \leq \mathbf{R}_{M_2}(s, g, \pi^*_{M_2}(s, g), s') \text{ for all } s \in \mathcal{S}, \tag{6.2}$$

since the rewards are the same at non-terminal states and $\mathbf{R}_{\text{MIN}}$ at terminal states $s \neq g$. (6.3)

$$\implies \mathbb{E}_{\pi^*_{M_1}}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(s_t, g, a_t, s_{t+1})\right] \leq \mathbb{E}_{\pi^*_{M_2}}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{R}(s_t, g, a_t, s_{t+1})\right] \tag{6.4}$$

$$\implies \mathbf{V}^*_{M_1}(s, g) \leq \mathbf{V}^*_{M_2}(s, g). \tag{6.5}$$

Now let $M_1, M_2 \in \mathcal{M}$. Then for all (s, g, a) in $\mathcal{S} \times \mathcal{G} \times \mathcal{A}$,

(i):

$$\mathbf{Q}^*_{sup}(s, g, a) := \sup_{M \in \{M_1, M_2\}} \mathbf{Q}^*_M(s, g, a)$$

$$= \sup_{M \in \{M_1, M_2\}} \left[\mathbf{R}_M(s, g, a, s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)\mathbf{V}^*_M(s', g)\right]$$

$$= \sup_{M \in \{M_1, M_2\}} \mathbf{R}_M(s, g, a, s') + \sup_{M \in \{M_1, M_2\}} \left[\gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)\mathbf{V}^*_M(s', g)\right],$$

since $\mathbf{R}_M(s, g, a, s') = 0 \; \forall s \notin \mathcal{G}$ and $P(\omega|s, a) = 1$ with $\mathbf{V}^*_M(\omega, g) = 0 \; \forall s \in \mathcal{G}$.

$$= \mathbf{R}_{M_1 \vee M_2}(s, g, a, s') + \sup_{M \in \{M_1, M_2\}} \left[\gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)\mathbf{V}^*_M(s', g)\right]$$

$$= \mathbf{R}_{M_1 \vee M_2}(s, g, a, s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \sup_{M \in \{M_1, M_2\}} \mathbf{V}^*_M(s', g) \quad \text{(Using Equation 6.5)}$$

$$= \mathbf{R}_{M_1 \vee M_2}(s, g, a, s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)\mathbf{V}^*_{M_1 \vee M_2}(s', g),$$

Using Equation 6.5 since $\sup_{M \in \{M_1, M_2\}} \mathbf{R}_M(g, g, \pi^*_M(g, g), s')$ defines $\mathbf{V}^*_{M_1 \vee M_2}(s', g)$.

$$= \mathbf{Q}^*_{M_1 \vee M_2}(s, g, a).$$

$$\implies \mathbf{Q}^*_{sup} = \mathbf{Q}^*_{M_1 \vee M_2} \in \mathcal{Q}^*.$$

Since $\mathbf{Q}^*_{sup}$ is in $\mathcal{Q}^*$, it follows from the pointwise $\leq$ on $\mathbb{R}$ that it is the lowest upper bound of $\mathbf{Q}^*_1$ and $\mathbf{Q}^*_2$.

(ii): Follows similarly to (i).

□

This means that the set of WVFs $\mathcal{Q}^*$ forms a lattice $(\mathcal{Q}^*, \vee, \wedge)$ with $\vee$ and $\wedge$ given by $\mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2} := \sup\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\}$ and $\mathbf{Q}^*_{M_1} \wedge \mathbf{Q}^*_{M_2} := \inf\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\}$ respectively. We define these formally as follows:

Definition 6.1. *Let $(\mathcal{M}, \vee, \wedge)$ be a lattice of goal-reaching tasks, and let $\mathcal{Q}^*$ be the set of optimal WVFs of tasks in $\mathcal{M}$. The join $\vee : \mathcal{Q}^* \times \mathcal{Q}^* \to \mathcal{Q}^*$ and meet $\wedge : \mathcal{Q}^* \times \mathcal{Q}^* \to \mathcal{Q}^*$ operators* *are given by the mappings* $(\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}) \mapsto \mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2}$ *and* $(\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}) \mapsto \mathbf{Q}^*_{M_1} \wedge \mathbf{Q}^*_{M_2}$ *respectively, where*

$$\begin{aligned}\mathbf{Q}^*_{M_2} \vee \mathbf{Q}^*_{M_2} : \ & \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R} \\ & (s,g,a) \mapsto \sup\{\mathbf{Q}^*_{M_1}(s,g,a), \mathbf{Q}^*_{M_2}(s,g,a)\},\end{aligned}$$

$$\begin{aligned}\mathbf{Q}^*_{M_1} \wedge \mathbf{Q}^*_{M_2} : \ & \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R} \\ & (s,g,a) \mapsto \inf\{\mathbf{Q}^*_{M_1}(s,g,a), \mathbf{Q}^*_{M_2}(s,g,a)\}.\end{aligned}$$

In fact $(\mathcal{Q}^*, \vee, \wedge)$ forms a distributive lattice, following from the distributivity of inf and sup on real numbers. We state this as follows:

Proposition 6.3. *Let* $(\mathcal{M}, \vee, \wedge)$ *be a lattice of goal-reaching tasks, and let* $\mathcal{Q}^*$ *be the set of optimal WVFs of tasks in* $\mathcal{M}$. *Then* $(\mathcal{Q}^*, \vee, \wedge)$ *is a distributive lattice.*

Proof. Follows from the distributivity of inf and sup. □

Given a non-empty finite set $\mathcal{O}$ of lower bounded subsets of WVFs $\mathcal{N} \subset \mathcal{Q}^*$, the WVF lattice $(\mathcal{Q}^*, \vee, \wedge)$ gives us a principled way of specifying the disjunction of conjunctions:

$$\bigvee_{\mathcal{N} \in \mathcal{O}} \left(\bigwedge_{N \in \mathcal{N}} N \right)(s,g,a) = \sup_{\mathcal{N} \in \mathcal{O}} \left(\inf_{N \in \mathcal{N}} \mathbf{Q}^*_N(s,g,a) \right).$$

Similarly, given a non-empty finite set $\mathcal{O}$ of upper bounded subsets of WVFs $\mathcal{N} \subset \mathcal{Q}^*$, the conjunction of disjunctions is given by

$$\bigwedge_{\mathcal{N} \in \mathcal{O}} \left(\bigvee_{N \in \mathcal{N}} N \right)(s,g,a) = \inf_{\mathcal{N} \in \mathcal{O}} \left(\sup_{N \in \mathcal{N}} \mathbf{Q}^*_N(s,g,a) \right).$$

Having established a lattice algebra over tasks and WVFs, we show that there exists an equivalence between the two. As a result, if we can specify a task under the lattice algebra, we can immediately obtain the optimal WVF for the task. This homomorphism follows from the fact that $\sup\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\} = \mathbf{Q}^*_{M_1 \vee M_2}$ and $\inf\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\} = \mathbf{Q}^*_{M_1 \wedge M_2}$ in Proposition 6.2.

Theorem 6.1. *Let* $(\mathcal{M}, \vee, \wedge)$ *be a lattice of goal-reaching tasks, and let* $(\mathcal{Q}^*, \vee, \wedge)$ *be the corresponding lattice of WVFs. Let* $H : \mathcal{M} \to \mathcal{Q}^*$ *be any map from* $\mathcal{M}$ *to* $\mathcal{Q}^*$ *such that* $H(M) = \mathbf{Q}^*_M$ *for all* M *in* $\mathcal{M}$. *Then* H *is a homomorphism.*

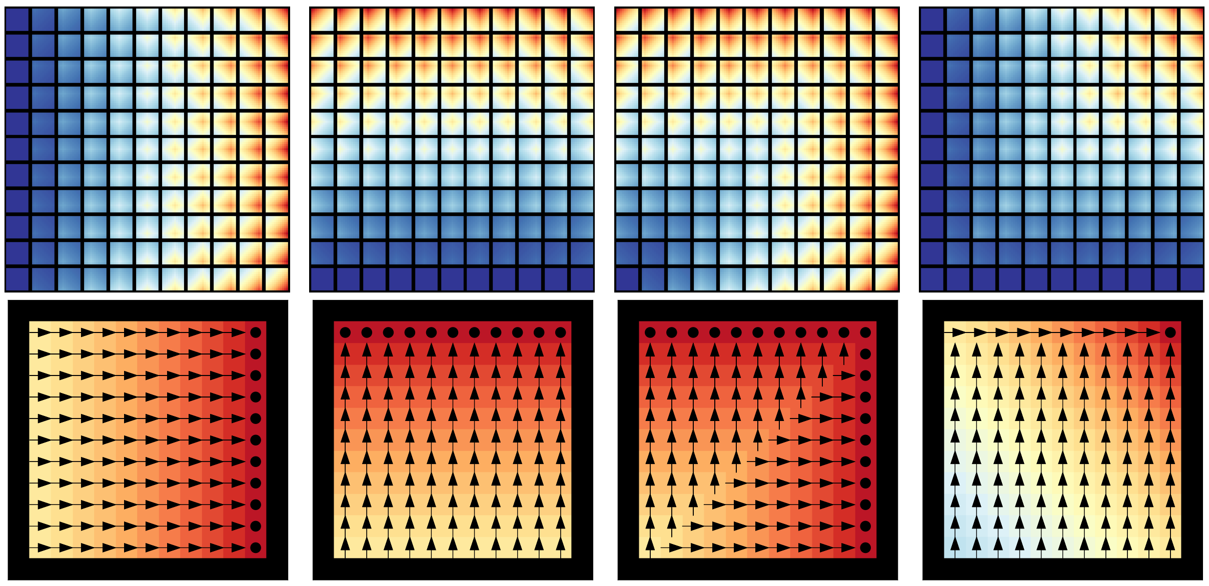
Proof. Follows from the proof of Proposition 6.2, which gives

$$\mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2} = \sup\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\} = \mathbf{Q}^*_{M_1 \vee M_2}$$

and

$$\mathbf{Q}^*_{M_1} \wedge \mathbf{Q}^*_{M_2} = \inf\{\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2}\} = \mathbf{Q}^*_{M_1 \wedge M_2}$$

$\forall M_1, M_2 \in \mathcal{M}$. □



(a) $\mathbf{Q}^*_{🟥}$ (b) $\mathbf{Q}^*_{🟦}$ (c) $\mathbf{Q}^*_{🟥} \vee \mathbf{Q}^*_{🟦}$ (d) $\mathbf{Q}^*_{🟥} \wedge \mathbf{Q}^*_{🟦}$

Figure 6.1: Showing the disjunction and conjunction of the learned WVFs ($\mathbf{Q}^*_{🟥}$, $\mathbf{Q}^*_{🟦}$) in the bin packing domain. The top row shows the WVFs, and the bottom one shows the value functions and policies obtained by acting greedily over their values per goal.

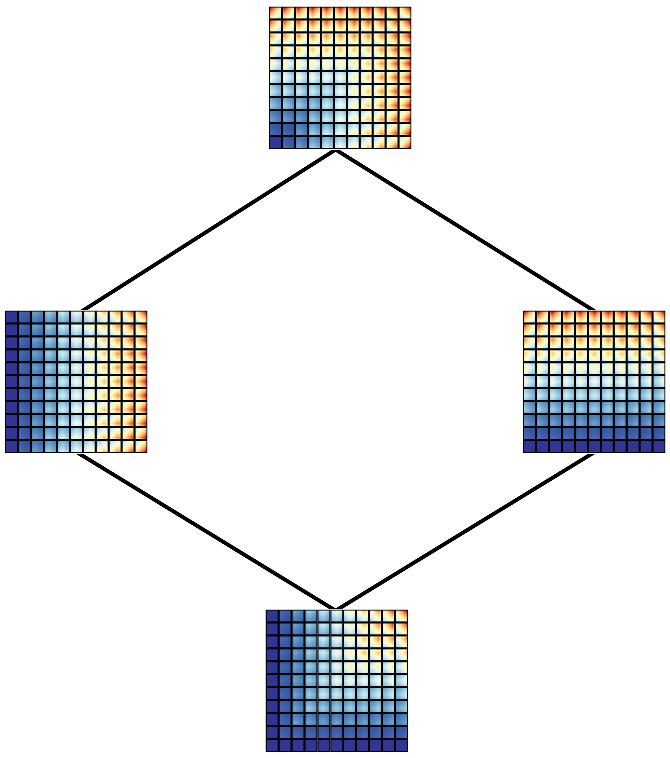


Figure 6.2: Hasse diagram of the WVF lattice generated by $\mathbf{Q}^*_{🟥}$ and $\mathbf{Q}^*_{🟦}$.

Experiment 6.1. *Consider the bin packing domain introduced in Example 3.1 where internal rewards are* 0 *and the goal rewards range from* 0 *to* 1*. The discounting used is* $\gamma = 0.95$.

We train an agent on the tasks 🟥 *and* 🟦*, in which the robot must respectively pack all the red and blue objects into the bin. Figure 6.1 shows the learned WVFs* $\mathbf{Q}^*_{\text{🟥}}$ *and* $\mathbf{Q}^*_{\text{🟦}}$ *together with their disjunction and conjunction. Notice how the composition of WVFs exhibits the same semantics as that of rewards. This highlights the homomorphism proven in Theorem 6.1, showing the structural similarity between the task space and value function space. Finally, Figure 6.2 shows the Hasse diagram of the WVF sub-lattice generated by all combinations of disjunction and conjunction of* $\mathbf{Q}^*_{\text{🟥}}$ *and* $\mathbf{Q}^*_{\text{🟦}}$*.*

6.2 De Morgan WVF algebras

Having formalised the meaning of disjunction and conjunction, we next formalise the meaning of the negation of WVFs. A De Morgan algebra enables us to define this by adding the minimal required properties that encapsulate the desired semantics of negation. Specifically, we assume that the set of tasks is bounded and that the WVF of each task has mastery. We then define the negation of an WVF as follows:

Definition 6.2. *Let* $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *be a De Morgan lattice of goal-reaching tasks, and let* $\mathcal{Q}^*$ *be the set of optimal WVFs of tasks in* $\mathcal{M}$*. Define* $\mathbf{Q}^*_{INF}, \mathbf{Q}^*_{SUP} \in \mathcal{Q}^*$ *to be the optimal* $\bar{Q}$*-functions for the task bounds* $\mathcal{M}_{INF}, \mathcal{M}_{SUP} \in \mathcal{M}$*. Then the negation operator is given by*

$$\begin{aligned}\neg :\ & \mathcal{Q}^* \to \mathcal{Q}^* \\ & \mathbf{Q}^* \mapsto \neg\mathbf{Q}^*, \text{ where } \neg\mathbf{Q}^* :\ \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R} \\ & \qquad (s, g, a) \mapsto (\mathbf{Q}^*_{SUP}(s, g, a) + \mathbf{Q}^*_{INF}(s, g, a)) - \mathbf{Q}^*(s, g, a).\end{aligned}$$

We now show that the negation of $\mathbf{Q}^* \in \mathcal{Q}^*$ is indeed in $\mathcal{Q}^*$ for WVFs with mastery:

Proposition 6.4. *Let* $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *be a De Morgan lattice of goal-reaching tasks, and let* $\mathcal{Q}^*$ *be the set of optimal WVFs of tasks in* $\mathcal{M}$*. Then* $\neg\mathbf{Q}^*_M = \mathbf{Q}^*_{\neg M} \in \mathcal{Q}^*$ *for all* $M \in \mathcal{M}$*.*

Proof. For all (s, g, a) in $\mathcal{S} \times \mathcal{G} \times \mathcal{A}$,

$$\begin{aligned}&[\mathcal{T}\neg\mathbf{Q}^*_M]\,(s, g, a) = \mathbf{R}_{\neg M}(s, g, a, s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \max_{a' \in \mathcal{A}} \neg\mathbf{Q}^*_M(s', g, a') \\ &= [(\mathbf{R}_{\mathcal{M}_{SUP}}(s, g, a, s') + \mathbf{R}_{\mathcal{M}_{INF}}(s, g, a, s')) - \mathbf{R}_M(s, g, a, s')] + \\ &\quad \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \max_{a' \in \mathcal{A}} [(\mathbf{Q}^*_{SUP}(s', g, a') + \mathbf{Q}^*_{INF}(s', g, a')) - \mathbf{Q}^*_M(s', g, a')] \\ &= [(\mathbf{R}_{\mathcal{M}_{SUP}}(s, g, a, s') + \mathbf{R}_{\mathcal{M}_{INF}}(s, g, a, s')) - \mathbf{R}_M(s, g, a, s')] + \\ &\quad \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \left[\left(\max_{a' \in \mathcal{A}} \mathbf{Q}^*_{SUP}(s', g, a') + \max_{a' \in \mathcal{A}} \mathbf{Q}^*_{INF}(s', g, a')\right) - \max_{a' \in \mathcal{A}} \mathbf{Q}^*_M(s', g, a')\right] \\ &\quad \text{(Using Theorem 5.3)} \\ &= \mathbf{R}_{\mathcal{M}_{SUP}}(s, g, a, s') + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) \max_{a' \in \mathcal{A}} \mathbf{Q}^*_{SUP}(s', g, a') +\end{aligned}$$

$$\begin{aligned}
&\mathbf{R}_{\mathcal{M}_{INF}}(s,g,a,s') + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) \max_{a'\in\mathcal{A}} \mathbf{Q}^*_{INF}(s',g,a') - \\
&\mathbf{R}_{M}(s,g,a,s') + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) \max_{a'\in\mathcal{A}} \mathbf{Q}^*_{M}(s',g,a') \\
&= (\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) - \mathbf{Q}^*_{M}(s,g,a) \\
&= \neg\mathbf{Q}^*_{M}(s,g,a).
\end{aligned}$$

Hence $\neg\mathbf{Q}^*_M$ is a fixed point of the Bellman optimality operator.

If $s \in \mathcal{G}$, then

$$\begin{aligned}
\neg\mathbf{Q}^*_{M}(s,g,a) &= (\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) - \mathbf{Q}^*_{M}(s,g,a) \\
&= (\mathbf{R}_{\mathcal{M}_{SUP}}(s,g,a,s') + \mathbf{R}_{\mathcal{M}_{INF}}(s,g,a,s')) - \mathbf{R}_{M}(s,g,a,s') = \mathbf{Q}^*_{\neg M}(s,g,a).
\end{aligned}$$

Since $\neg\mathbf{Q}^*_M = \mathbf{Q}^*_{\neg M}$ holds in $\mathcal{G}$ and $\neg\mathbf{Q}^*_M$ is a fixed point of $\mathcal{T}$, then $\neg\mathbf{Q}^*_M = \mathbf{Q}^*_{\neg M}$ holds everywhere.

□

We now formalise the interaction of the negation of WVFs with the conjunction and disjunction of WVFs as follows:

Proposition 6.5. *Let* $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *be a De Morgan lattice of goal-reaching tasks, and let* $\mathcal{Q}^*$ *be the set of optimal WVFs of tasks in* $\mathcal{M}$*. Then* $(\mathcal{Q}^*, \vee, \wedge, \neg, \mathbf{Q}^*_{SUP}, \mathbf{Q}^*_{INF})$ *is a De Morgan algebra.*

Proof. Let $\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2} \in \mathcal{Q}^*$ be the optimal $\bar{Q}$-value functions for tasks $M_1, M_2 \in \mathcal{M}$ with reward functions r_{M_1} and r_{M_2}. We show that $\neg, \vee, \wedge$ satisfy the De Morgan algebra axioms (i) – (vii).[2]

(i)–(v): These follow from the properties of inf and sup.

(vi): This follows from the bounds $\mathbf{Q}^*_{SUP}, \mathbf{Q}^*_{INF} \in \mathcal{Q}^*$.

(vii): The first condition easily follows from the definition of $\neg$. For the second condition, we have that for all (s,g,a) in $\mathcal{S}\times\mathcal{G}\times\mathcal{A}$,

$$\begin{aligned}
\neg(\mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2})(s,g,a) &= (\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) - \sup_{M\in\{M_1,M_2\}} \mathbf{Q}^*_{M}(s,g,a) \\
&= (\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) + \inf_{M\in\{M_1,M_2\}} -\mathbf{Q}^*_{M}(s,g,a) \\
&= \inf_{M\in\{M_1,M_2\}} (\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) - \mathbf{Q}^*_{M}(s,g,a) \\
&= (\neg\mathbf{Q}^*_{M_1} \wedge \neg\mathbf{Q}^*_{M_2})(s,g,a).
\end{aligned}$$

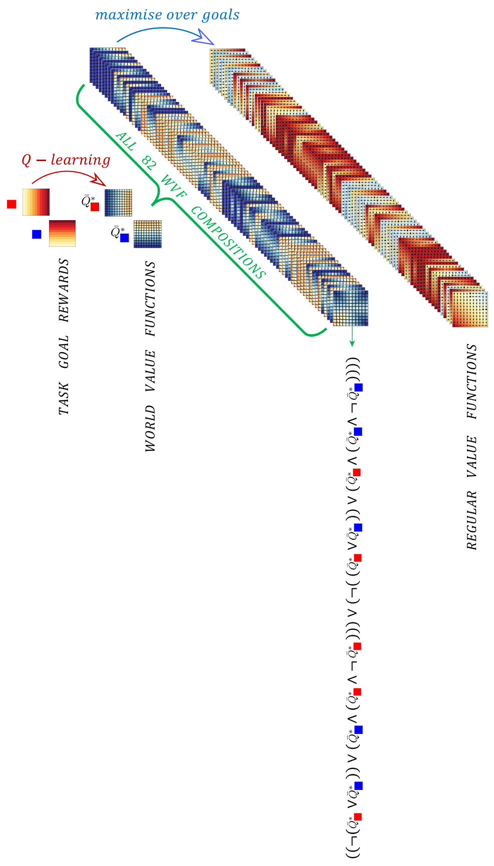
[2]The De Morgan algebra axioms are stated in Definition 2.3.

□

Having established a De Morgan algebra over tasks and WVFs, we show that there exists an equivalence between the two. As a result, if we can specify a task under the De Morgan algebra, we can immediately derive the optimal WVF for the task.

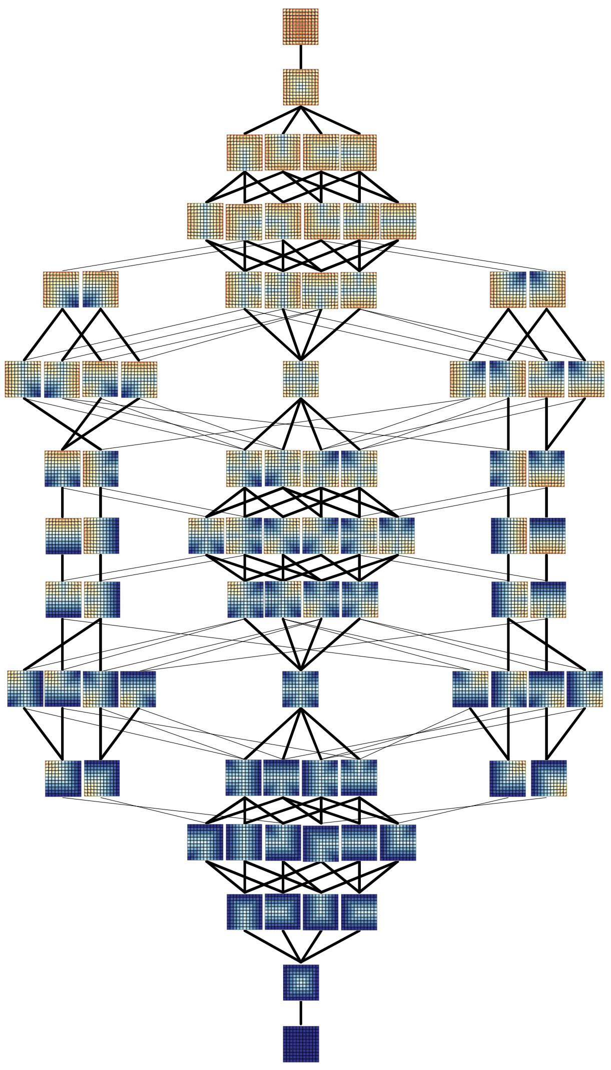
Theorem 6.2. *Let $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ be a De Morgan lattice of goal-reaching tasks, and let $(\mathcal{Q}^*, \vee, \wedge, \neg, \mathbf{Q}^*_{SUP}, \mathbf{Q}^*_{INF})$ be the corresponding De Morgan lattice over WVFs. Let $H : \mathcal{M} \to \mathcal{Q}^*$ be any map from $\mathcal{M}$ to $\mathcal{Q}^*$ such that $H(M) = \mathbf{Q}^*_M$ for all M in $\mathcal{M}$. Then H is a homomorphism.*

Proof. This follows from the proof of Proposition 6.4 and the homomorphism between the task and WVF lattices. □



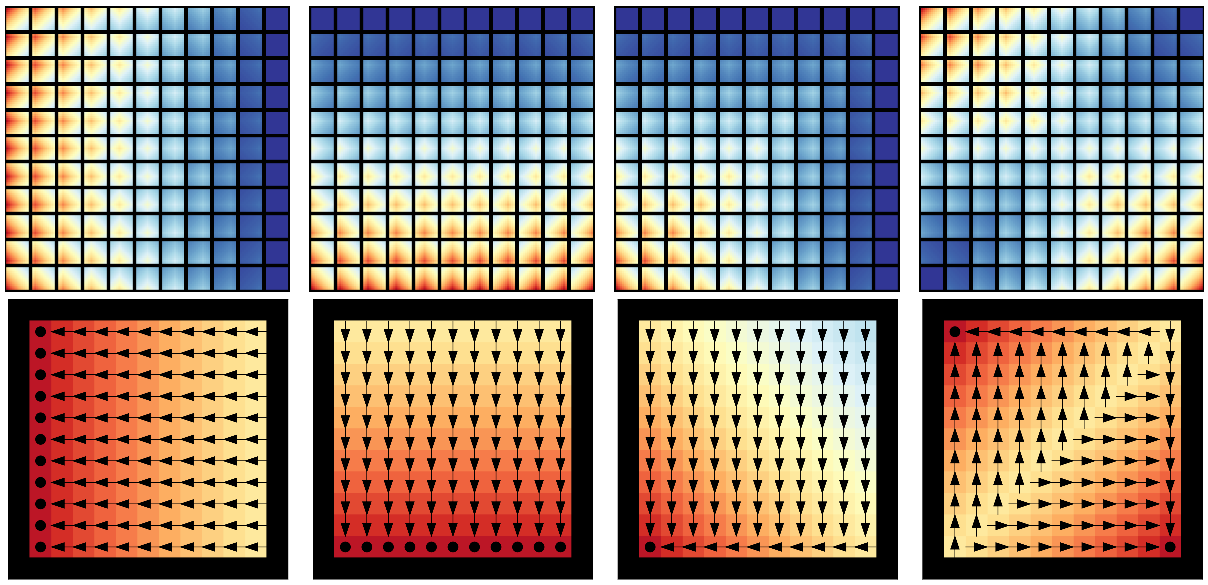
maximise over goals
Q − learning
ALL 82 WVF COMPOSITIONS
TASK GOAL REWARDS
WORLD VALUE FUNCTIONS
REGULAR VALUE FUNCTIONS
$((\neg(\bar{Q}^*_{🟥} \vee \bar{Q}^*_{🟦})) \vee (\bar{Q}^*_{🟦} \wedge (\bar{Q}^*_{🟥} \wedge \neg\bar{Q}^*_{🟥}))) \vee (\neg(\neg(\bar{Q}^*_{🟥} \vee \bar{Q}^*_{🟦})) \wedge (\bar{Q}^*_{🟥} \wedge (\bar{Q}^*_{🟦} \wedge \neg\bar{Q}^*_{🟦})))$

(a) Zero-shot WVF compositions

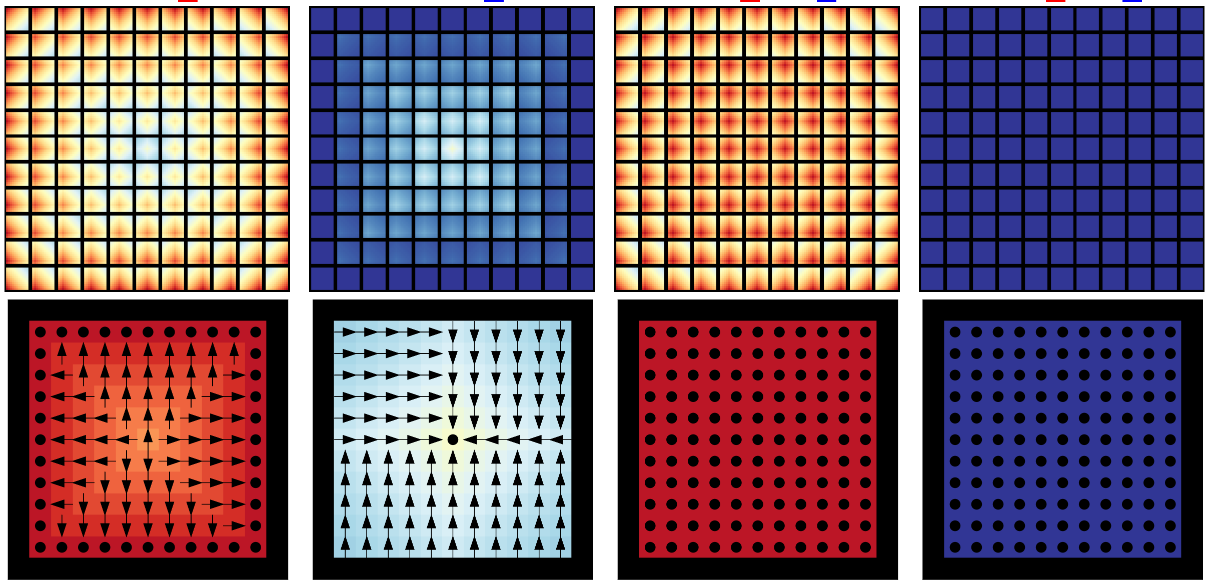


(b) Hasse diagram

Figure 6.3: Illustration of the De Morgan sub-lattice generated by composing $\mathbf{Q}^*_{🟥}$ and $\mathbf{Q}^*_{🟦}$.



(a) $\neg\mathbf{Q}^*_{\text{🟥}}$ (b) $\neg\mathbf{Q}^*_{\text{🟦}}$ (c) $\mathbf{Q}^*_{\text{🟥}} \,\bar{\vee}\, \mathbf{Q}^*_{\text{🟦}}$ (d) $\mathbf{Q}^*_{\text{🟥}} \,\underline{\vee}\, \mathbf{Q}^*_{\text{🟦}}$



(e) $\begin{matrix}(\mathbf{Q}^*_{\text{🟥}} \vee \neg\mathbf{Q}^*_{\text{🟥}}) \\ \vee \\ (\mathbf{Q}^*_{\text{🟦}} \vee \neg\mathbf{Q}^*_{\text{🟦}})\end{matrix}$ (f) $\begin{matrix}(\mathbf{Q}^*_{\text{🟥}} \wedge \neg\mathbf{Q}^*_{\text{🟥}}) \\ \wedge \\ (\mathbf{Q}^*_{\text{🟦}} \wedge \neg\mathbf{Q}^*_{\text{🟦}})\end{matrix}$ (g) $\mathbf{Q}^*_{SUP}$ (h) $\mathbf{Q}^*_{INF}$

Figure 6.4: The learned boundary WVFs ($\mathbf{Q}^*_{SUP}$, $\mathbf{Q}^*_{INF}$) and the composition of learned WVFs ($\mathbf{Q}^*_{\text{🟥}}$, $\mathbf{Q}^*_{\text{🟦}}$). The top rows in each figure show the WVFs and the bottom ones show the value functions and policies obtained by acting greedily over goals.

Experiment 6.2. *Consider the bin packing domain introduced in Example 3.1. We use the WVFs learned in Experiment 6.1 for the tasks* 🟥 *and* 🟦*. Figure 6.3 shows the De Morgan sub-lattice generated by all combinations of their disjunction, conjunction, and negation. Figure 6.4 shows sample WVF compositions, and the optimal policies and optimal value functions obtained from them by maximising over goals. For example, their negations* $\neg\mathbf{Q}^*_{\text{🟥}}$ *and* $\neg\mathbf{Q}^*_{\text{🟦}}$*, result in policies in which the agent determines how to unpack all the red and blue objects from the bin, without further learning. This shows that the negation defined above does indeed have the expected* *semantics. The figure also shows that arbitrary disjunction, conjunction, and negation of WVFs also produce WVFs with the desired semantics.*

6.3 Boolean WVF algebra

While the De Morgan WVF algebra allows us to specify arbitrary disjunction, conjunction, and negation of WVFs, it does not in general represent the full desired properties of logic. In particular, the WVF compositions do not always satisfy the laws of the excluded middle and of non-contradiction. This is because the De Morgan WVF algebra allows for WVFs obtained from non-Boolean tasks—that is, tasks with non-binary rewards. In this section, we show that by restricting the WVFs to those of Boolean tasks, we obtain the full logic on WVFs.

To achieve this, we first redefine $\neg$ over $\mathcal{Q}^*$ as follows:

$$\neg(\mathbf{Q}^*)(.) \coloneqq \begin{cases} \mathbf{Q}^*_{SUP}(.) & \text{if } |\mathbf{Q}^*(.) - \mathbf{Q}^*_{INF}(.)| \leq |\mathbf{Q}^*(.) - \mathbf{Q}^*_{SUP}(.)| \\ \mathbf{Q}^*_{INF}(.) & \text{otherwise,} \end{cases}, \forall (.) \in \mathcal{S} \times \mathcal{G} \times \mathcal{A}.$$

The intuition behind this re-definition of the negation operator is as follows: since each goal is either desirable or not, the optimal world value function $\mathbf{Q}^*(s,g,a)$ is either $\mathbf{Q}^*_{SUP}(s,g,a)$ or $\mathbf{Q}^*_{INF}(s,g,a)$. Hence, if $\mathbf{Q}^*(s,g,a)$ is closer to $\mathbf{Q}^*_{INF}(s,g,a)$, then its negation should be $\mathbf{Q}^*_{SUP}(s,g,a)$, and vice versa. For tasks in $\mathcal{M}$, this is equivalent to the previous definition of $\neg$ for optimal Q-value functions, but it will give us tight bounds when composing ϵ-optimal WVFs (see Theorem 6.5).

Proposition 6.6. *Let $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ be a Boolean algebra of goal-reaching tasks, and let $\mathcal{Q}^*$ be the set of optimal WVFs of tasks in $\mathcal{M}$. Then $(\mathcal{Q}^*, \vee, \wedge, \neg, \mathbf{Q}^*_{SUP}, \mathbf{Q}^*_{INF})$ is a Boolean WVF algebra.*

Proof. Let $\mathbf{Q}^*_{M_1}, \mathbf{Q}^*_{M_2} \in \mathcal{Q}^*$ be the optimal world action-value functions for tasks $M_1, M_2 \in \mathcal{M}$ with reward functions R_{M_1} and R_{M_2}. We show that $\neg, \vee, \wedge$ satisfy the Boolean properties (i) – (vii).

(i)–(v): These follow directly from the properties of the $\inf$ and $\sup$ functions.

(vi): For all (s,g,a) in $\mathcal{S} \times \mathcal{G} \times \mathcal{A}$,

$$\begin{aligned}
&(\mathbf{Q}^*_{SUP} \wedge \mathbf{Q}^*_{M_1})(s,g,a) \\
&= \inf\{\mathbf{Q}^*_{SUP}(s,g,a), \mathbf{Q}^*_{M_1}(s,g,a)\} \\
&= \begin{cases} \inf\{\mathbf{Q}^*_{SUP}(s,g,a), \mathbf{Q}^*_{SUP}(s,g,a)\}, & \text{if } R_{M_1}(g,a',s') = R_{\mathcal{M}_{SUP}}(g,a',s') \; \forall\, a', s' \\ \inf\{\mathbf{Q}^*_{SUP}(s,g,a), \mathbf{Q}^*_{INF}(s,g,a)\}, & \text{otherwise.} \end{cases} \\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s,g,a), & \text{if } R_{M_1}(g,a',s') = R_{\mathcal{M}_{SUP}}(g,a',s') \; \forall\, a', s' \\ \mathbf{Q}^*_{INF}(s,g,a), & \text{otherwise.} \end{cases} \\
&= \mathbf{Q}^*_{M_1}(s,g,a), \text{ since } R_{M_1}(g,a',s') \in \{R_{\mathcal{M}_{INF}}(g,a',s'), R_{\mathcal{M}_{SUP}}(g,a',s')\} \; \forall\, a', s'.
\end{aligned}$$

Similarly, $\mathbf{Q}^*_{SUP} \vee \mathbf{Q}^*_{M_1} = \mathbf{Q}^*_{SUP}$, $\mathbf{Q}^*_{INF} \wedge \mathbf{Q}^*_{M_1} = \mathbf{Q}^*_{INF}$, and $\mathbf{Q}^*_{INF} \vee \mathbf{Q}^*_{M_1} = \mathbf{Q}^*_{M_1}$.

(vii): For all $(.)$ in $\mathcal{S} \times \mathcal{G} \times \mathcal{A}$,

$$\begin{aligned}
&(\mathbf{Q}^*_{M_1} \wedge \neg \mathbf{Q}^*_{M_1})(.) \\
&= \inf\{\mathbf{Q}^*_{M_1}(.), \neg \mathbf{Q}^*_{M_1}(.)\} \\
&= \begin{cases} \inf\{\mathbf{Q}^*_{INF}(.), \mathbf{Q}^*_{SUP}(.)\} & \text{if } |\mathbf{Q}^*(.) - \mathbf{Q}^*_{INF}(.)| \leq |\mathbf{Q}^*(.) - \mathbf{Q}^*_{SUP}(.)| \\ \inf\{\mathbf{Q}^*_{SUP}(.), \mathbf{Q}^*_{INF}(.)\} & \text{otherwise,} \end{cases} \\
&= \mathbf{Q}^*_{INF}(.).
\end{aligned}$$

Similarly, $\mathbf{Q}^*_{M_1} \vee \neg \mathbf{Q}^*_{M_1} = \mathbf{Q}^*_{SUP}$.

□

Having established a Boolean algebra over tasks and WVFs, we show that they are homomorphic. As a result, if we can specify a task under the Boolean algebra, we can immediately obtain the optimal WVF for the task.

Theorem 6.3. *Let $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ be a Boolean algebra of goal-reaching tasks, and let $(\mathcal{Q}^*, \vee, \wedge, \neg, \mathbf{Q}^*_{SUP}, \mathbf{Q}^*_{INF})$ be the corresponding Boolean algebra of WVFs. Let $H : \mathcal{M} \to \mathcal{Q}^*$ be any map from $\mathcal{M}$ to $\mathcal{Q}^*$ such that $H(M) = \mathbf{Q}^*_M$ for all M in $\mathcal{M}$. Then H is a homomorphism.*

Proof. Let $M_1, M_2 \in \mathcal{M}$. Then for all (s, g, a) in $\mathcal{S} \times \mathcal{G} \times \mathcal{A}$,

(i):

$$\begin{aligned}
&\mathbf{Q}^*_{\neg M_1}(s, g, a) \\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s, g, a), & \text{if } R_{\neg M_1}(g, a', s') = R_{\mathcal{M}_{SUP}}(g, a', s') \ \forall\, a', s' \\ \mathbf{Q}^*_{INF}(s, g, a), & \text{otherwise.} \end{cases} \\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s, g, a), & \text{if } R_{M_1}(g, a', s') = R_{\mathcal{M}_{INF}}(g, a', s') \ \forall\, a', s' \\ \mathbf{Q}^*_{INF}(s, g, a), & \text{otherwise.} \end{cases} \\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s, g, a), & \text{if } \mathbf{Q}^*_{M_1}(s, g, a) = \mathbf{Q}^*_{INF}(s, g, a) \\ \mathbf{Q}^*_{INF}(s, g, a), & \text{otherwise.} \end{cases} \\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s, g, a), & \text{if } |\mathbf{Q}^*_{M_1}(s, g, a) - \mathbf{Q}^*_{INF}(s, g, a)| \leq |\mathbf{Q}^*_{M_1}(s, g, a) - \mathbf{Q}^*_{SUP}(s, g, a)| \\ \mathbf{Q}^*_{INF}(s, g, a), & \text{otherwise.} \end{cases} \\
&= \neg \mathbf{Q}^*_{M_1}(s, g, a).
\end{aligned}$$

(ii):

$$\begin{aligned}
&\mathbf{Q}^*_{M_1 \vee M_2}(s,g,a)\\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s,g,a), & \text{if } R_{M_1 \vee M_2}(g,a',s') = R_{\mathcal{M}_{SUP}}(g,a',s') \;\forall\; a',s' \\ \mathbf{Q}^*_{INF}(s,g,a), & \text{otherwise.} \end{cases}\\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s,g,a), & \text{if } \max\{R_{M_1}(g,a',s'), R_{M_2}(g,a',s')\} = R_{\mathcal{M}_{SUP}}(g,a',s') \;\forall\; a',s' \\ \mathbf{Q}^*_{INF}(s,g,a), & \text{otherwise.} \end{cases}\\
&= \begin{cases} \mathbf{Q}^*_{SUP}(s,g,a), & \text{if } \max\{\mathbf{Q}^*_{M_1}(s,g,a), \mathbf{Q}^*_{M_2}(s,g,a)\} = \mathbf{Q}^*_{SUP}(s,g,a) \\ \mathbf{Q}^*_{INF}(s,g,a), & \text{otherwise.} \end{cases}\\
&= \max\{\mathbf{Q}^*_{M_1}(s,g,a), \mathbf{Q}^*_{M_2}(s,g,a)\}\\
&= (\mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2})(s,g,a).
\end{aligned}$$

(iii): Follows similarly to (ii).

□

Given that the algebraic structures of goal-reaching tasks and WVFs are homomorphic, any task that is specified according to the task algebra can be immediately solved according to the corresponding WVF algebra. We now show that for the case of a Boolean WVF algebra, we can immediately construct optimal WVFs directly from a set of desired goals. We do this by showing that the WVF and power set Boolean algebras are in fact isomorphic. Consider the mapping F between the set of WVFs $\boldsymbol{\mathcal{Q}}^*$ and the power-set $P(\mathcal{G})$, given by

$$\begin{aligned}
F :\; & P(\mathcal{G}) \to \boldsymbol{\mathcal{Q}}^*\\
& \mathcal{H} \mapsto \mathbf{Q}^*_{\mathcal{H}}, \text{ where } \mathbf{Q}^*_{\mathcal{H}} :\; \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}\\
& \qquad (s,g,a) \mapsto \begin{cases} \mathbf{Q}^*_{SUP}(s,g,a), & \text{if } g \in \mathcal{H} \\ \mathbf{Q}^*_{INF}(s,g,a), & \text{otherwise.} \end{cases}
\end{aligned}$$

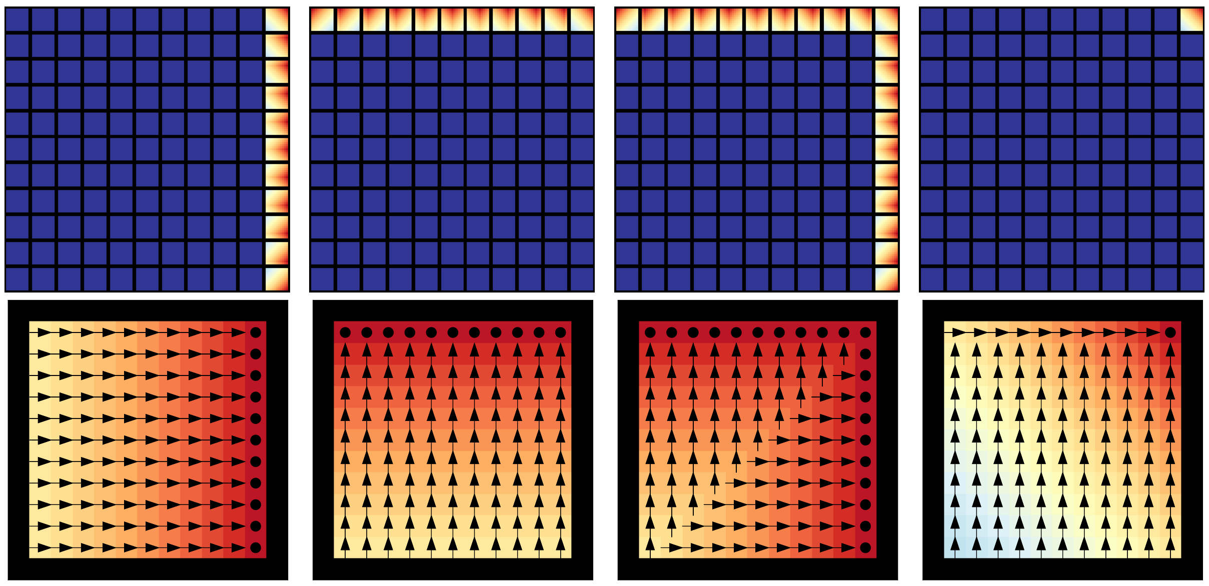
F is clearly an isomorphism since each $g \in \mathcal{H}$ defines, and can be defined by, each $g' \in \mathcal{G}$ that gives $\mathbf{Q}^*_{\mathcal{H}}(s,g',a) = \mathbf{Q}^*_{SUP}(s,g',a) \;\forall\; (s,a) \in \mathcal{S} \times \mathcal{A}$.

The isomorphism between a Boolean WVF algebra and the power-set Boolean algebra gives us the following important result.

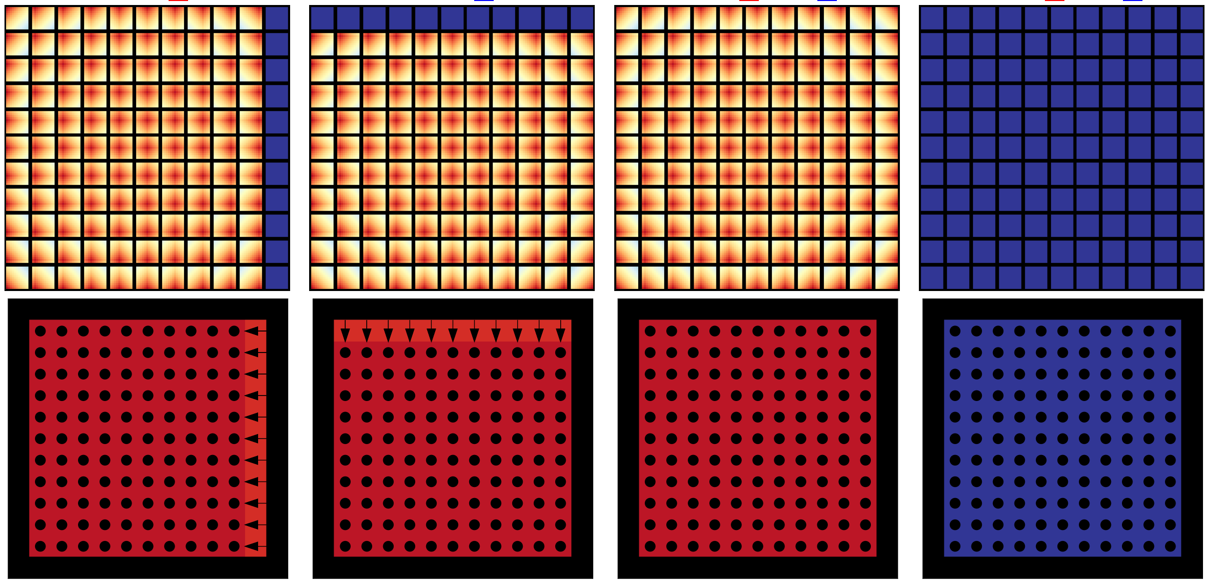
Theorem 6.4. *Let $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ be a Boolean algebra of goal-reaching tasks, and let $\boldsymbol{\mathcal{Q}}^*$ be the set of optimal WVFs of tasks in $\mathcal{M}$. Then the Boolean algebra on $\boldsymbol{\mathcal{Q}}^*$ is isomorphic to the Boolean algebra on $\mathcal{M}$.*

This illustrates how the base knowledge an agent needs to act optimally in an environment for any future goal-reaching task can be constructed rather than learned—if there is an efficient way of doing that construction (for example when $|\mathcal{G}|$ is finite and relatively small). All that is required to be learned are two WVFs: the lower bound WVF $\mathbf{Q}^*_{INF}$ and upper bound WVF $\mathbf{Q}^*_{SUP}$. Precisely, for any task M with desirable goals $\mathcal{H}$ (goals such that $R_M(s,a,g) = R_{SUP}(s,a,g)$), we have for all (s,g,a):

$$\mathbf{Q}^*_M(s,g,a) := \mathbf{Q}^*_{SUP}(s,g,a) \textbf{ if } (g \in \mathcal{H}) \textbf{ else } \mathbf{Q}^*_{INF}(s,g,a)$$



(a) $\mathbf{Q}^*_{\color{red}\blacksquare}$ (b) $\mathbf{Q}^*_{\color{blue}\blacksquare}$ (c) $\mathbf{Q}^*_{\color{red}\blacksquare} \vee \mathbf{Q}^*_{\color{blue}\blacksquare}$ (d) $\mathbf{Q}^*_{\color{red}\blacksquare} \wedge \mathbf{Q}^*_{\color{blue}\blacksquare}$



(e) $\neg\mathbf{Q}^*_{\color{red}\blacksquare}$ (f) $\neg\mathbf{Q}^*_{\color{blue}\blacksquare}$ (g) $\begin{matrix}(\mathbf{Q}^*_{\color{red}\blacksquare} \vee \neg\mathbf{Q}^*_{\color{red}\blacksquare}) \\ \vee \\ (\mathbf{Q}^*_{\color{blue}\blacksquare} \vee \neg\mathbf{Q}^*_{\color{blue}\blacksquare})\end{matrix}$ (h) $\begin{matrix}(\mathbf{Q}^*_{\color{red}\blacksquare} \wedge \neg\mathbf{Q}^*_{\color{red}\blacksquare}) \\ \wedge \\ (\mathbf{Q}^*_{\color{blue}\blacksquare} \wedge \neg\mathbf{Q}^*_{\color{blue}\blacksquare})\end{matrix}$

Figure 6.5: The Boolean composition (top row) of learned WVFs in the bin packing domain. The corresponding value functions and policies in the bottom row are obtained by acting greedily over their Q-values per goal.

Experiment 6.3. *Consider the simple bin packing domain used in Example 3.4 where the goal rewards are restricted to* $R_{MIN} = 0$ *or* $R_{MAX} = 1$*. The internal rewards are* 0 *and the goal rewards are* 0 *for undesired terminal states and* 1 *for desired ones. The discounting used is* $\gamma = 0.95$.

We train an agent on the tasks $\color{red}\blacksquare$ *and* $\color{blue}\blacksquare$ *(as described in Example 3.4), producing the respective WVFs* $\mathbf{Q}^*_{\color{red}\blacksquare}$ *and* $\mathbf{Q}^*_{\color{blue}\blacksquare}$*. We are now able to perform zero-shot composition of any logical combination*

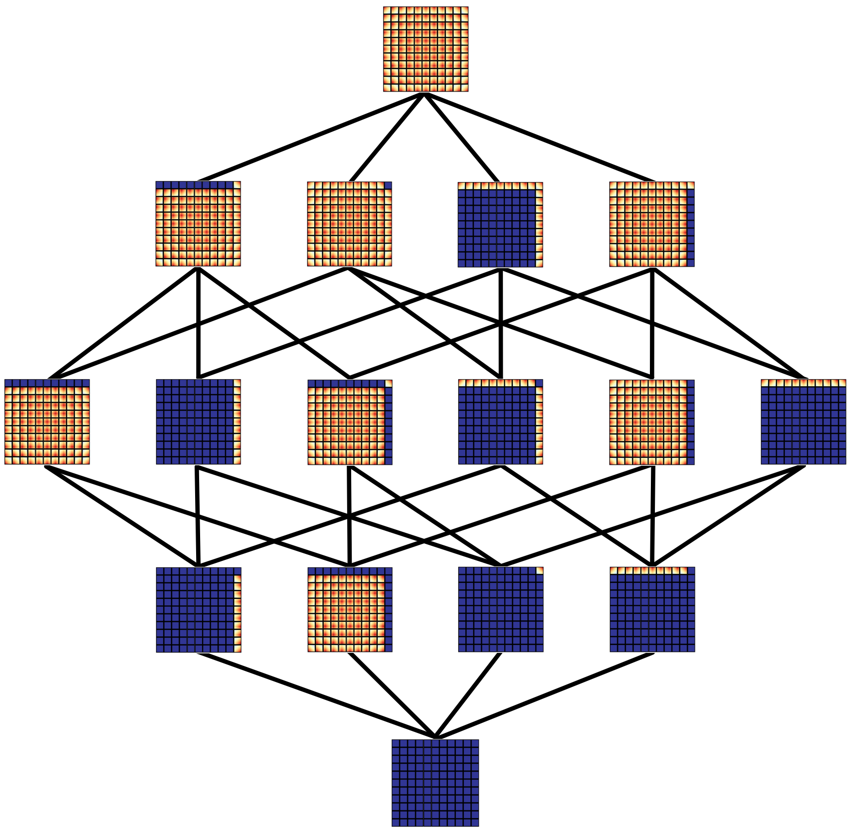


Figure 6.6: Hasse diagram of the Boolean sub-algebra generated by $\mathbf{Q}^*_{🟥}$ and $\mathbf{Q}^*_{🟦}$.

of 🟥 *and* 🟦. *This is demonstrated in Figure* 6.5. *As usual, we observe that the WVFs have the same structure as the rewards in the terminal set and encode how to achieve them. Notice how for the lower bound WVF (the result of meaningless composition), the optimal policy is to achieve any terminal state (Figure* 6.5f*). This is because all terminal states are equally undesirable (they have the lowest values). Figure* 6.6 *shows the Hasse diagram of the Boolean sub-lattice generated by* $\mathbf{Q}^*_{🟥}$ *and* $\mathbf{Q}^*_{🟦}$.

Finally, we show that while optimal WVFs are more expensive to learn than optimal standard value functions, the trade-off with the compositional explosion of skills justifies this cost. We demonstrate this in the bin packing domain where we need to learn only 7 base tasks (Figure 6.8 *shows the learned WVFs), as opposed to 121 for the disjunctive case (since there are 121 goals). Figure* 6.7 *shows results in comparison to the disjunctive composition of* van Niekerk et al. *[*2019*].*

6.4 Composition with function approximation

In this section, we demonstrate that our compositional approach can also be used to tackle domains where function approximation is required. In this setting, it is likely that learned value functions will be suboptimal owing to generalisation error. However, we show in Theorem 6.5 that the simplicity of the lattice algebra operators—*supremum* for disjunction and *infimum* for conjunction—enables disjunctions and conjunctions of ϵ-optimal WVFs without any decrease

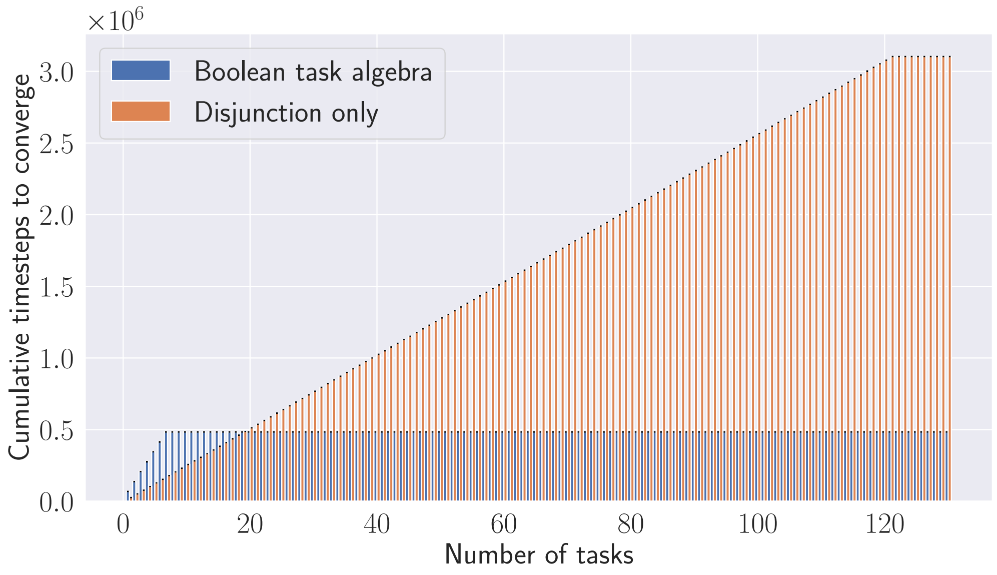


Figure 6.7: Cumulative number of samples required to solve tasks in the bin packing domain. Error bars represent standard deviations over 100 runs.

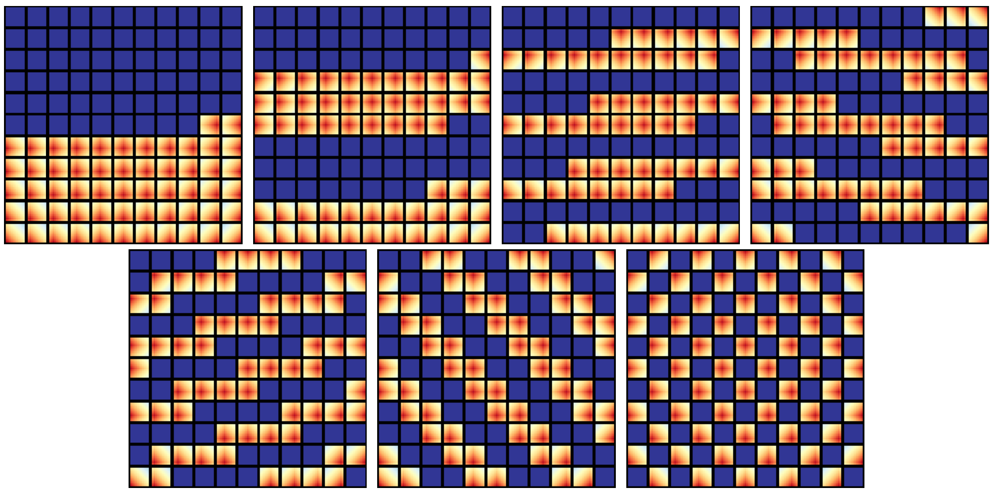


Figure 6.8: Base WVFs (generators) for the full Boolean algebra WVF space

in optimality. Furthermore, we show that the negation of an ϵ-optimal WVF leads to only a constant decrease in optimality for a De Morgan algebra, and no decrease in optimality for a Boolean algebra.

Theorem 6.5. *Let $\mathcal{M}$ be a set of tasks and $\mathcal{Q}^*$ the set of optimal WVFs for tasks in $\mathcal{M}$. Denote $\tilde{\mathbf{Q}}^*_M$ as the ϵ-optimal WVF for a task $M \in \mathcal{M}$ such that*

$$|\mathbf{Q}^*_M(s,g,a) - \tilde{\mathbf{Q}}^*_M(s,g,a)| \leq \epsilon \text{ for all } (s,g,a) \in \mathcal{S} \times \mathcal{G} \times \mathcal{A}.$$

Then for all M_1, M_2 in $\mathcal{M}$ and (s,g,a) in $\mathcal{S} \times \mathcal{G} \times \mathcal{A}$,

(i) $\left|[\mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2}](s,g,a) - [\tilde{\mathbf{Q}}^*_{M_1} \vee \tilde{\mathbf{Q}}^*_{M_2}](s,g,a)\right| \leq \epsilon$

(ii) $\left|[\mathbf{Q}^*_{M_1} \wedge \mathbf{Q}^*_{M_2}](s,g,a) - [\tilde{\mathbf{Q}}^*_{M_1} \wedge \tilde{\mathbf{Q}}^*_{M_2}](s,g,a)\right| \leq \epsilon$

(iii) if $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *is a De Morgan algebra,*

$$\left|\neg\mathbf{Q}^*_{M_1}(s,g,a) - \neg\tilde{\mathbf{Q}}^*_{M_1}(s,g,a)\right| \leq 3\epsilon$$

(iv) if $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ *is a Boolean algebra,*

$$\left|\neg\mathbf{Q}^*_{M_1}(s,g,a) - \neg\tilde{\mathbf{Q}}^*_{M_1}(s,g,a)\right| \leq \epsilon$$

Proof. **(i):**

$$\begin{aligned}
&\left|[\mathbf{Q}^*_{M_1} \vee \mathbf{Q}^*_{M_2}](s,g,a) - [\tilde{\mathbf{Q}}^*_{M_1} \vee \tilde{\mathbf{Q}}^*_{M_2}](s,g,a)\right| \\
&= \left|\sup_{M \in \{M_1,M_2\}} \mathbf{Q}^*_M(s,g,a) - \sup_{M \in \{M_1,M_2\}} \tilde{\mathbf{Q}}^*_M(s,g,a)\right| \\
&\leq \sup_{M \in \{M_1,M_2\}} \left|\mathbf{Q}^*_M(s,g,a) - \tilde{\mathbf{Q}}^*_M(s,g,a)\right| \\
&\leq \epsilon
\end{aligned}$$

(ii):

$$\begin{aligned}
&\left|[\mathbf{Q}^*_{M_1} \wedge \mathbf{Q}^*_{M_2}](s,g,a) - [\tilde{\mathbf{Q}}^*_{M_1} \wedge \tilde{\mathbf{Q}}^*_{M_2}](s,g,a)\right| \\
&= \left|\inf_{M \in \{M_1,M_2\}} \mathbf{Q}^*_M(s,g,a) - \inf_{M \in \{M_1,M_2\}} \tilde{\mathbf{Q}}^*_M(s,g,a)\right| \\
&\leq \inf_{M \in \{M_1,M_2\}} \left|\mathbf{Q}^*_M(s,g,a) - \tilde{\mathbf{Q}}^*_M(s,g,a)\right| \\
&\leq \epsilon
\end{aligned}$$

(iii): Let $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ be a De Morgan algebra. Then,

$$\begin{aligned}
&\left|\neg\mathbf{Q}^*_{M_1}(s,g,a) - \neg\tilde{\mathbf{Q}}^*_{M_1}(s,g,a)\right| \\
&= |\,(\mathbf{Q}^*_{SUP}(s,g,a) + \mathbf{Q}^*_{INF}(s,g,a)) - \mathbf{Q}^*_{M_1}(s,g,a) - \\
&\quad \left(\tilde{\mathbf{Q}}^*_{SUP}(s,g,a) + \tilde{\mathbf{Q}}^*_{INF}(s,g,a)\right) - \tilde{\mathbf{Q}}^*_{M_1}(s,g,a)| \\
&= |\left(\mathbf{Q}^*_{SUP}(s,g,a) - \tilde{\mathbf{Q}}^*_{SUP}(s,g,a)\right) + \left(\mathbf{Q}^*_{INF}(s,g,a) - \tilde{\mathbf{Q}}^*_{INF}(s,g,a)\right) + \\
&\quad \left(\tilde{\mathbf{Q}}^*_{M_1}(s,g,a) - \mathbf{Q}^*_{M_1}(s,g,a)\right)| \\
&\leq \left|\mathbf{Q}^*_{SUP}(s,g,a) - \tilde{\mathbf{Q}}^*_{SUP}(s,g,a)\right| + \left|\mathbf{Q}^*_{INF}(s,g,a) - \tilde{\mathbf{Q}}^*_{INF}(s,g,a)\right| + \\
&\quad \left|\tilde{\mathbf{Q}}^*_{M_1}(s,g,a) - \mathbf{Q}^*_{M_1}(s,g,a)\right| \\
&\leq 3\epsilon
\end{aligned}$$

(iv): Let $(\mathcal{M}, \vee, \wedge, \neg, \mathcal{M}_{SUP}, \mathcal{M}_{INF})$ be a Boollean algebra.

$$
\begin{aligned}
&\left|\neg \mathbf{Q}^*_{M_1}(s,g,a) - \neg \tilde{\mathbf{Q}}^*_{M_1}(s,g,a)\right| \\
&= \begin{cases} |\mathbf{Q}^*_{SUP}(s,g,a) - \neg \tilde{\mathbf{Q}}^*_{INF}(s,g,a)|, & \text{if } \mathbf{Q}^*_{M_1} = \mathbf{Q}^*_{INF}(s,g,a) \\ |\mathbf{Q}^*_{INF}(s,g,a) - \neg \tilde{\mathbf{Q}}^*_{SUP}(s,g,a)|, & \text{otherwise.} \end{cases} \\
&= \begin{cases} |\mathbf{Q}^*_{SUP}(s,g,a) - \tilde{\mathbf{Q}}^*_{SUP}(s,g,a)|, & \text{if } \mathbf{Q}^*_{M_1} = \mathbf{Q}^*_{INF}(s,g,a) \\ |\mathbf{Q}^*_{INF}(s,g,a) - \tilde{\mathbf{Q}}^*_{INF}(s,g,a)|, & \text{otherwise.} \end{cases} \\
&\leq \epsilon.
\end{aligned}
$$

□

Experiment 6.4. *We consider a continuous 3D Four Rooms environment where the ant robot of* Duan et al. [2016] *must navigate to the center of specific rooms. The environment is simulated in MuJoCo* [Todorov et al. 2012] *with a 29-dimensional continuous state space (representing the position and velocity of the ant's joints) and an 8-dimensional continuous action space. Figure* 6.9 *shows a rendered view of the environment.*

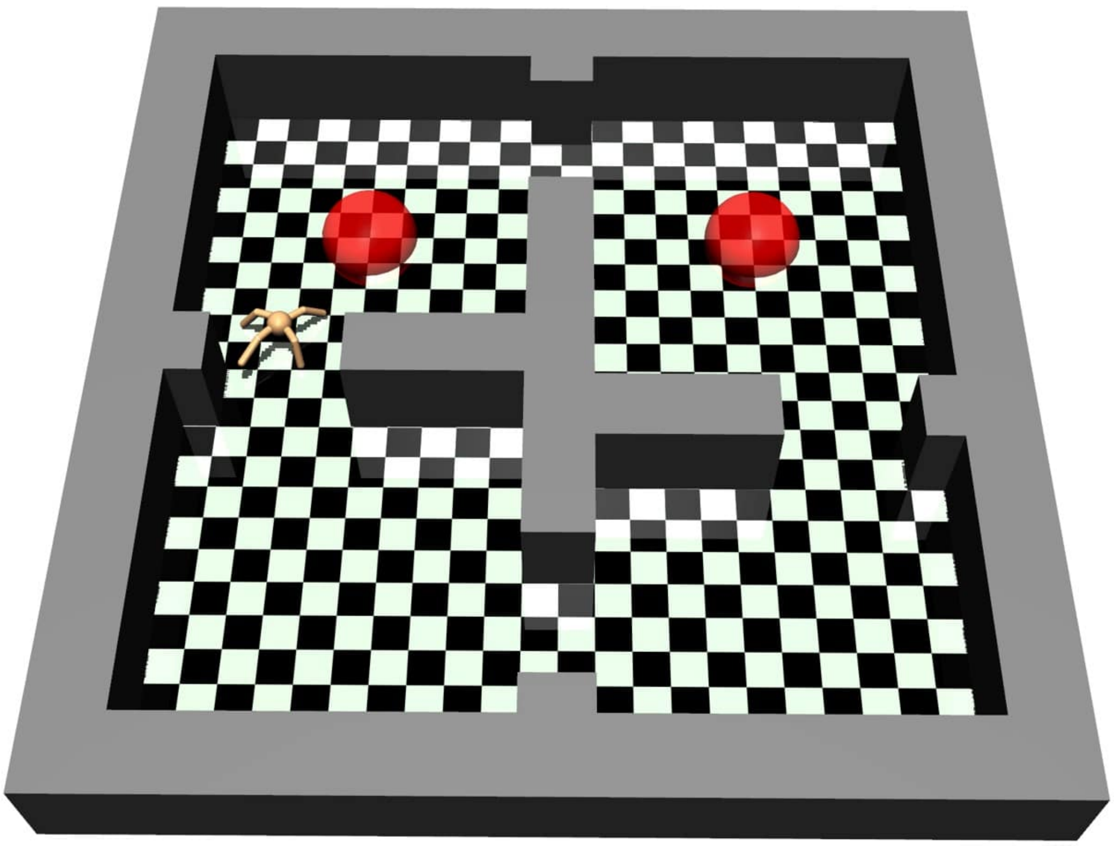


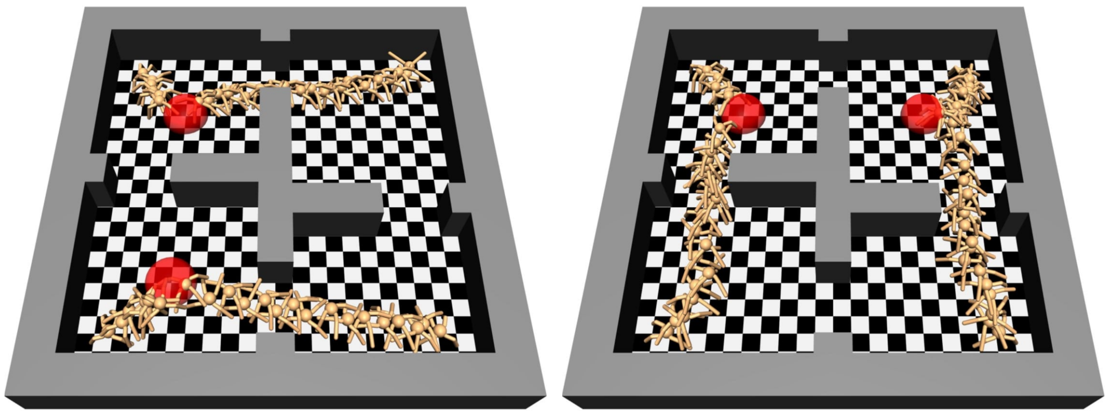
Figure 6.9: The layout of the MuJoCo Four Rooms domain with a quadruped ant robot. The spheres indicate goals the agent must reach (center of the two top rooms in this case).

Since this environment is particularly challenging (as a result of the large state space and the ant's non-linear and highly unstable dynamics), we use dense rewards on internal states (similarly to Peng et al. [2019]) to facilitate learning, and limit the terminal states of each task to only the desired ones (that is, the environment terminates only when the ant is ϵ-close to the center of a desired room). We use soft actor-critic with automated entropy adjustments (SAC-AEA) [Haarnoja et al. 2018b] as the off-policy RL algorithm to learn the world values for each goal. The architectures used for the policy and action-value networks are as follows:

1. *Three fully-connected linear layers for the policy networks: (a) Layer 1 has input size 29 and output size 256 and uses a ReLU activation function. (b) Layer 2 has input size 256 and output size 256 and uses a ReLU activation function. (c) Layer 3 has input size 256 and output size 16, representing the mean and standard deviation of the Gaussian distribution over actions.*

2. *Three fully-connected linear layers for the action-value networks: (a) Layer 1 has input size 37 and output size 256 and uses a ReLU activation function. (b) Layer 2 has input size 256 and output size 256 and uses a ReLU activation function. (c) Layer 3 has input size 256 and output size 1 with no activation function.*

We first learn to solve the two base tasks: navigating to the top rooms, and navigating to the left rooms. The resulting trajectories from the learned WVFs are illustrated by Figure 6.10. We then demonstrate that zero-shot compositions do indeed still hold by showing compositions characterised by disjunction, conjunction and exclusive-or. Figure 6.11 shows the resulting trajectories.

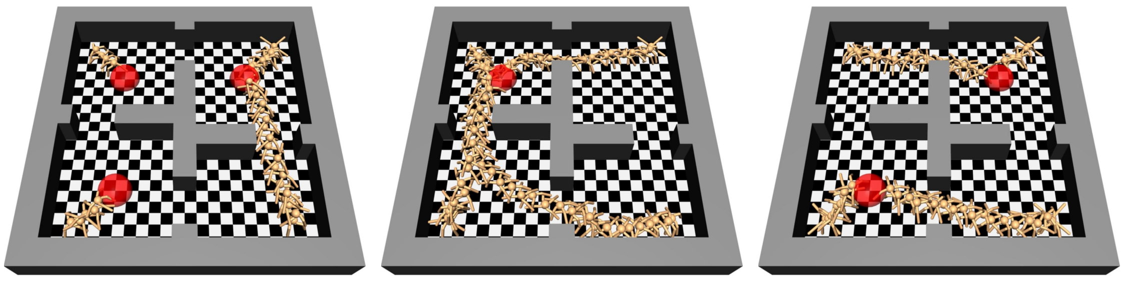


(a) $\mathbf{Q}_L^*$ (b) $\mathbf{Q}_T^*$

Figure 6.10: Learned base tasks for the MuJoCo environment. We show the trajectories obtained from the learned WVFs for each base task from different starting states.

6.5 Related works

Skill composition is a promising form of transfer learning that has garnered significant attention in recent years [Mendez and Eaton 2023]. It involves combining optimal policies or value functions from previously learned tasks to derive solutions for new tasks. Broadly, these methods fall into two categories in the litterature:



(a) $\mathbf{Q}^*_L \vee \mathbf{Q}^*_T$ (b) $\mathbf{Q}^*_L \wedge \mathbf{Q}^*_T$ (c) $\mathbf{Q}^*_L \veebar \mathbf{Q}^*_T$

Figure 6.11: An example of Boolean algebraic composition using the learned WVFs with dense rewards. We show the trajectories obtained from different starting states.

Zero-shot Composition: Todorov [2009] demonstrated that for Linearly-solvable Markov Decision Processes (LMDPs), composed value functions can be obtained without further learning through weighted addition of other tasks. However, this approach is confined to tabular cases with known dynamics. van Niekerk *et al.* [2019] extended this work by introducing OR composition, enabling the derivation of optimal action-value functions without further learning. Haarnoja *et al.* [2018a] and van Niekerk *et al.* [2019] also explored AND compositions, approximating composite task solutions by averaging the value functions of constituent tasks. However, all these approaches do not consider at full set of logical compositions simultaneously.

Few-shot Composition: Sahni *et al.* [2017] proposed ComposeNets, leveraging neural networks to learn linear temporal logic operators for composite skills, facilitating zero-shot generalization and subsequent optimal task solving after few iterations. Other approaches leverage successor features (SF) and generalized policy improvement (GPI) for few-shot task solving with convergence guarantees. Additionally, Hunt *et al.* [2019] demonstrated zero-shot AND composition, allowing the composition of policies without additional exploration in max-entropy-RL by introducing divergence correction terms. Peng et al. (2019) introduced a few-shot learning approach, enabling policy composition via multiplication of base skills, although lacking theoretical underpinnings.

Finally, recent advancements by Nangue Tasse *et al.* [2020b] extended zero-shot optimal composition to include all logical operators OR, AND, and NOT, paving the way for broader application in transfer learning. In this chapter, we have extended this work to more general goal-reaching tasks, with new results on the sub-optimality of the composition operators when the learned WVFs are also sub-optimal.

6.6 Conclusion

In this chapter, we have made significant strides in formalising the logical composition of world value functions (WVFs) for goal-reaching tasks. Building upon the foundation laid out in previous chapters, where we extended the framework of Nangue Tasse *et al.* [2020b] to formalise the logical composition of tasks with arbitrary rewards, we have shown that WVFs encode enough information to solve new goal-reaching tasks without further learning. Importantly, we also showed that they are homomorphic to the ones over tasks. This enables WVFs to be treated algebraically in a similar way to sets in set theory, and propositions in propositional logic. Remarkably, we also showed that these compositions work in the function approximation setting with little to no loss in optimality even when WVFs are sub-optimal. These findings pave the way for more efficient and effective approaches to mastering arbitrary goal-reaching tasks in reinforcement learning settings. In the next chapters, we will leverage this ability to improve sample efficiency and generalisation in RL.

Chapter 7

Generalisation in lifelong learning

This chapter is based on the published work
"Generalisation in Lifelong Reinforcement Learning through Logical Composition"
[Nangue Tasse et al. 2023a], in collaboration with Steven James, and Benjamin Rosman.

As we discussed in the introduction (Chapter 1), a major challenge in RL is building general-purpose agents that can use existing knowledge to quickly solve new tasks in their environment. However, the previous chapters have only looked at solving single tasks (Chapters 3 and 4) or augmenting an agent's ability to solve multiple tasks—given instructions on exactly how to combine learned skills to solve the current task (Chapters 5 and 6). This is ultimately a fatal flaw, since learning to solve complex, real-world tasks from scratch for *every* task of interest is typically infeasible. Similarly, it is also impractical to assume that an agent is always equipped with sufficient skills to solve new tasks, let alone instruct the agent on the correct composition of skills to use for every single task. Hence, the question of interest is then: after learning n tasks sampled from some distribution, how can an agent transfer or leverage the skills learned from those n tasks to improve its starting performance or learning speed in task $n + 1$?

In this chapter, we answer this question by leveraging logical composition (Chapter 6) to create a framework that enables an agent to autonomously determine whether a new task can be immediately solved using its existing abilities, or whether a task-specific skill should be learned. In the latter case, the proposed algorithm also enables the agent to learn the new task faster by generating an estimate of the optimal policy. We make the following main contributions:

(i) **Sample efficiency (Section 7.1.1):** We propose a transfer learning method where the agent first infers the specification of the current task based on the skills it currently has, then uses it to estimate the optimal WVF. Importantly, we bound the performance of the transferred policy on a new task, and also compare it to previous work in the discounted setting.

(ii) **Generalisation (Section 7.1.1):** We propose a lifelong RL algorithm named SOPGOL (*Sum Of Products with Goal-Oriented Learning*). SOPGOL enables agents to iteratively solve tasks as they are given, while at the same time constructing a *library* of skills that can be composed to obtain behaviours for solving future tasks faster, or even without further reinforcement learning. Importantly, we give bounds on the necessary and sufficient number of tasks that need to be learned throughout an agent's lifetime to generalise over a distribution.

(iii) **Empirical evaluations:** We verify our approach in a series of experiments, where we perform transfer learning both after learning a set of base tasks, and after learning an arbitrary set of tasks. We also demonstrate that, as a side effect of our transfer learning approach, an agent can produce an interpretable Boolean expression of its understanding of the current task. Finally, we demonstrate our approach in the full lifelong setting where an agent receives tasks from an unknown distribution. Starting from scratch, an agent is able to quickly generalise over the task distribution after learning only a few tasks, which are sub-logarithmic in the size of the task space.

7.1 Lifelong transfer through composition

In lifelong RL, an agent is presented with a series of tasks sampled from some distribution $\mathcal{D}$. The agent then needs to not only transfer knowledge learned from previous tasks to solve new but related tasks quickly, but it also should not forget learned knowledge in the process. We formalise this lifelong learning problem as follows:

Definition 7.1. *Let $\mathcal{D}$ be an unknown, possibly non-stationary, distribution over a set of tasks $\mathcal{M}(\mathcal{S}, \mathcal{A}, P, \gamma, R_0)$. The lifelong learning problem consists of the repetition of the following steps for $t \in \mathbb{N}$:*

1. *The agent is given a task $M_t \sim \mathcal{D}(t)$,*
2. *The agent interacts with the MDP M_t until it is ϵ-optimal in $M_0, ..., M_t$.*

This formulation of lifelong RL is similar to that of Abel *et al.* [2018]; the main difference is that we do not assume that $\mathcal{D}$ is stationary, and we explicitly require an agent to retain learned skills. Since we aim to investigate how logical composition can be leveraged to address the lifelong learning problem, we focus on tasks where an agent is required to reach a set of desirable goals in a goal space $\mathcal{G} \subseteq \mathcal{S}$ (the environment terminates once a goal state is reached). We hence consider the set of tasks $\mathcal{M}$ such that the tasks are in the same environment—described by a background MDP $(\mathcal{S}, \mathcal{A}, P, \gamma, R_0)$—and each task can be uniquely specified by a set of desirable and undesirable goals:

$$\begin{aligned}\mathcal{M}(\mathcal{S}, \mathcal{A}, P, \gamma, R_0) := \{(\mathcal{S}, \mathcal{A}, P, \gamma, R) \mid \forall a \in \mathcal{A},\ & R(s,a) = R_0(s,a)\ \forall s \in \mathcal{S} \setminus \mathcal{G};\\ & R(g,a) = R_g \in \{R_{\text{MIN}}, R_{\text{MAX}}\}\ \forall g \in \mathcal{G}\}.\end{aligned}$$

As discussed in Chapter 1, one of the main goals in the lifelong setting is that of transfer [Taylor and Stone 2009]. We add an important question to this setting: how many tasks should an agent learn during its lifetime in order to generalise over the task distribution? In other words, how many tasks should it learn to be able to solve any new task immediately? While most approaches focus on the goal of transfer, the question of the number of tasks is often neglected by simply assuming the case where the agent has already learned n tasks [Abel *et al.* 2018; Barreto *et al.* 2018]. Consider, for example, a task space with only $|\mathcal{G}| = 40$ goals. Then, given the combination of all possible goals, the size of the task space is $|\mathcal{M}| = 2^{|\mathcal{G}|} \approx 10^{12}$. If $\mathcal{D}$ is a uniform distribution over $|\mathcal{M}|$, then for most transfer learning methods an agent will have to learn most of the tasks it is presented with, since the probability of observing the same task will be approximately zero. This is clearly impractical for a setting like RL, where learning methods often have a high sample complexity even with transfer learning. It is also extremely memory inefficient, since the learned skills of most tasks must be stored.

7.1.1 Transfer between tasks

In this section, we leverage the logical composition results to address the following question of interest: given an arbitrary set of learned tasks, can we transfer their skills to solve new tasks faster? As we will show in Theorem 7.1, we answer this question in the affirmative. To achieve this, we first note that each task $M \in \mathcal{M}$ can be associated with a binary vector $T \in \{0,1\}^{|\mathcal{G}|}$ which represents its set of desirable goals, as illustrated by the tasks in Table 7.1. The approximation $\tilde{T}$ of this task representation can be learned just from task rewards ($R_M(s,a)$) by simply computing $\tilde{T}(s) = \mathbf{1}_{R_M(s,a)=R_{\text{MAX}}}$ at each terminal state s that the agent reaches. We can then use any generic method, such as the *sum-of-products* (*SOP*), to determine a candidate Boolean expression ($\mathcal{B}_{EXP}$) in terms of the learned binary representations $\tilde{\mathcal{T}}_n = \{\tilde{T}_1, ..., \tilde{T}_n\}$ of a set of past tasks $\hat{\mathcal{M}} = \{M_1, ..., M_n\} \subseteq \mathcal{M}$. An estimate of the optimal $\mathbf{Q}$-value function of M can then be obtained by composing the learned $\mathbf{Q}$-value functions $\tilde{\mathcal{Q}}^*_n = \{\tilde{\mathbf{Q}}^*_1, ..., \tilde{\mathbf{Q}}^*_n\}$ according to $\mathcal{B}_{EXP}$. Theorem 7.1 shows the optimality of this process.

Theorem 7.1. *Let $M \in \mathcal{M}$ be a task with reward function R, binary representation T and optimal world action-value function $\mathbf{Q}^*$. Given ϵ-approximations of the binary representations $\tilde{\mathcal{T}}_n = \{\tilde{T}_1, ..., \tilde{T}_n\}$ and optimal $\mathbf{Q}$-functions $\tilde{\mathcal{Q}}^*_n = \{\tilde{\mathbf{Q}}^*_1, ..., \tilde{\mathbf{Q}}^*_n\}$ for n tasks $\hat{\mathcal{M}} = \{M_1, ..., M_n\} \subseteq \mathcal{M}$, let*

$$T_{\mathcal{F}} = \mathcal{B}_{EXP}(\tilde{\mathcal{T}}_n) \text{ and } \mathbf{Q}_{\mathcal{F}} = \mathcal{B}_{EXP}(\tilde{\mathcal{Q}}^*_n),$$

where $\mathcal{B}_{EXP}$ is derived from $\tilde{\mathcal{T}}_n$ and $\tilde{T}$ using a generic method $\mathcal{F}$.

Define $\pi(s) \in \arg\max_{a\in\mathcal{A}} Q_{\mathcal{F}}$ where $Q_{\mathcal{F}} := \max_{g\in\mathcal{G}} \mathbf{Q}_{\mathcal{F}}(s,g,a)$. Then,

(i) $\|Q^* - Q^\pi\|_\infty \leq \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}} + \mathbf{1}_{R\notin\{R_g\}_{|\mathcal{G}|}})R_\Delta + \epsilon)$,

(ii) if the dynamics are deterministic,

$$\|Q^* - Q_{\mathcal{F}}\|_\infty \leq (\mathbf{1}_{T\neq T_{\mathcal{F}}})R_\Delta + \epsilon,$$

where $\mathbf{1}$ is the indicator function, $R_g(s,a) := \mathbf{R}(s,g,a)$, $R_\Delta := R_{MAX} - R_{MIN}$, and $\|f-h\|_\infty := \max_{s,g,a}|f(s,g,a) - h(s,g,a)|$.

Proof. We first show that when the inferred task representation $T_{\mathcal{F}}$ is not the same as the true task representation T, the resulting estimate of the optimal world action-value function can be bounded by R_Δ. That is, $\|\mathbf{Q}^* - \mathbf{Q}_{\mathcal{F}}\|_\infty \leq (\mathbf{1}_{T\neq T_{\mathcal{F}}})R_\Delta + \epsilon$.

$$\begin{aligned}|\mathbf{Q}^*(s,g,a) - \mathbf{Q}_{\mathcal{F}}(s,g,a)| &= |\mathbf{Q}^*(s,g,a) - \mathbf{Q}^*_{\mathcal{F}}(s,g,a) + \mathbf{Q}^*_{\mathcal{F}}(s,g,a) - \mathbf{Q}_{\mathcal{F}}(s,g,a)| \\ &\leq |\mathbf{Q}^*(s,g,a) - \mathbf{Q}^*_{\mathcal{F}}(s,g,a)| + |\mathbf{Q}^*_{\mathcal{F}}(s,g,a) - \mathbf{Q}_{\mathcal{F}}(s,g,a)| \\ &\leq |\mathbf{Q}^*(s,g,a) - \mathbf{Q}^*_{\mathcal{F}}(s,g,a)| + \epsilon.\end{aligned}$$

If $T = T_{\mathcal{F}}$, then $\mathbf{Q}^*(s,g,a) = \mathbf{Q}^*_{\mathcal{F}}(s,g,a)$, and we are done. Let $T \neq T_{\mathcal{F}}$. Without loss of generality, let $\mathbf{Q}^*(s,g,a) = \mathbf{Q}^*_{SUP}(s,g,a)$ and $\mathbf{Q}^*_{\mathcal{F}}(s,g,a) = \mathbf{Q}^*_{INF}(s,g,a)$. Then,

$$\begin{aligned}|\mathbf{Q}^*(s,g,a) - \mathbf{Q}^*_{\mathcal{F}}(s,g,a)| &\leq |\mathbf{Q}^*_{SUP}(s,g,a) - \mathbf{Q}^*_{INF}(s,g,a)| \\ &\leq R_\Delta.\end{aligned}$$

(i): Now note that each g in $\mathcal{G}$ can be thought of as defining an MDP $M_g := (\mathcal{S}, \mathcal{A}, P, R_g, \gamma)$ with reward function $R_g(s,a) := \mathbf{R}(s,g,a)$, optimal policy $\pi^*_g(s) = \bar{\pi}^*(s,g)$ and optimal Q-value function $Q^{\pi^*_g}(s,a) = \mathbf{Q}^*(s,g,a)$. Then this proof follows similarly to that of Theorem 2 in Barreto *et al.* [2017],

$$\begin{aligned}
&Q^*(s,a) - Q^\pi(s,a) \\
&\leq Q^*(s,a) - Q^{\pi^*_g}(s,a) + \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}})R_\Delta + \epsilon) \quad \text{([Barreto \textit{et al.} 2017, Theorem 1])} \\
&\leq \frac{2}{1-\gamma}\max_{s,a}|R(s,a) - R_g(s,a)| + \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}})R_\Delta + \epsilon) \\
&\quad \text{(using Lemma 1 in Barreto \textit{et al.} [2017])} \\
&\leq \frac{2}{1-\gamma}(\mathbf{1}_{R\neq R_g})R_\Delta + \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}})R_\Delta + \epsilon) \\
&\quad \text{(Since rewards only differ in } \mathcal{G} \text{ where } R(s,a), R_g(s,a) \in \{R_{\text{MIN}}, R_{\text{MAX}}\} \text{ for } s \in \mathcal{G}) \\
&\leq \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}} + \mathbf{1}_{R\neq R_g})R_\Delta + \epsilon).
\end{aligned}$$

Hence,

$$\begin{aligned}
\|Q^* - Q^\pi\|_\infty &\leq \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}} + \min_g \mathbf{1}_{R\neq R_g})R_\Delta + \epsilon) \\
&\leq \frac{2}{1-\gamma}((\mathbf{1}_{T\neq T_{\mathcal{F}}} + \mathbf{1}_{R\notin\{R_g\}_{|\mathcal{G}|}})R_\Delta + \epsilon) \\
&\quad \text{(Since } \min_g \mathbf{1}_{R\neq R_g} = 0 \text{ only when } R \in \{R_g\}_{|\mathcal{G}|}\text{).}
\end{aligned}$$

(ii):

$$\begin{aligned}
|Q^*(s,a) - Q_{\mathcal{F}}(s,a)| &= |\max_g \mathbf{Q}^*(s,g,a) - \max_g \mathbf{Q}_{\mathcal{F}}(s,g,a)| \\
&\leq \max_g |\mathbf{Q}^*(s,g,a) - \mathbf{Q}_{\mathcal{F}}(s,g,a)| \\
&\leq (\mathbf{1}_{T\neq T_{\mathcal{F}}})R_\Delta + \epsilon.
\end{aligned}$$

□

Theorem 7.1(i) states that if $\mathbf{Q}_{\mathcal{F}}$ is close to optimal, then acting greedily with respect to it is also close to optimal. Interestingly, this is similar to the bound obtained by Barreto *et al.* [2018] (Proposition 1) for transfer learning using generalised policy improvement (GPI), but stronger.[1] This is unsurprising, since $\pi(s) \in \arg\max_{a\in\mathcal{A}} \max_{g\in\mathcal{G}} \mathbf{Q}_{\mathcal{F}}(s,g,a)$ can be interpreted

[1]See Section 1.4 of the appendix for a detailed discussion of this with the simplification of the bound in Proposition 1 [Barreto *et al.* 2018] to the same form as Theorem 7.1(i).

as generalised policy improvement on the set of goal policies of the world value function $\mathbf{Q}_{\mathcal{F}}$. Importantly, if the environment is deterministic, then we obtain a strong bound on the composed value functions (Theorem 7.1(ii)). This bound shows that transfer learning using logical composition is ϵ-optimal—that is, there is no loss in optimality—when the new task is expressible as a logical combination of past ones. With the exponential nature of logical combination, this gives agents a strong generalisation ability over the task space—and hence over any task distribution—as we will show in Theorem 7.2.

7.1.2 Comparison with GPI

We first restate Proposition 1 of Barreto *et al.* [2018] here.

Proposition 7.1 ([Barreto *et al.* 2018]). *Let $M \in \mathcal{M}$ and let $Q_i^{\pi_j^*}$ be the action value function of an optimal policy of $M_j \in \mathcal{M}$ when executed in $M_i \in \mathcal{M}$. Given approximations $\{\tilde{Q}_i^{\pi_1}, ..., \tilde{Q}_i^{\pi_n}\}$ such that $|Q_i^{\pi_j} - \tilde{Q}_i^{\pi_j}| \leq \epsilon$ for all $s, a \in \mathcal{S} \times \mathcal{A}$, and $j \in \{1, ..., n\}$, let*

$$\pi(s) \in \arg\max_a \max_j \tilde{Q}_i^{\pi_j}(s, a).$$

then,

$$\|Q^* - Q^\pi\|_\infty \leq \frac{2}{1-\gamma}(\|R - R_i\|_\infty + \min_j \|R_i - R_j\|_\infty + \epsilon),$$

where Q^* is the optimal value function of M, Q^π is the value function of π in M, and $\|f - h\|_\infty :=$ $\max_{s,g,a} |f(s, g, a) - h(s, g, a)|$.

We can simplify the bound in Proposition 7.1 as follows:

$$\begin{aligned}
\|Q^* - Q^\pi\|_\infty &\leq \frac{2}{1-\gamma}(\|R - R_i\|_\infty + \min_j \|R_i - R_j\|_\infty + \epsilon) \\
&\leq \frac{2}{1-\gamma}((\mathbf{1}_{R \neq R_i})R_\Delta + \min_j \|R_i - R_j\|_\infty + \epsilon) \\
&\quad \text{(Since rewards only differ at } s \in \mathcal{G} \text{ where } R(s,a), R_i(s,a) \in \{R_{\text{MIN}}, R_{\text{MAX}}\} \\
&\leq \frac{2}{1-\gamma}((\mathbf{1}_{R \neq R_i})R_\Delta + (\min_j \mathbf{1}_{R_i \neq R_j})R_\Delta + \epsilon) \\
&\leq \frac{2}{1-\gamma}((\mathbf{1}_{r \neq R_i})R_\Delta + (\mathbf{1}_{R_i \notin \{R_j\}_n})R_\Delta + \epsilon) \\
&\quad \text{(Since } \min_j \mathbf{1}_{R_i \neq R_j} = 0 \text{ only when } R_i \in \{R_j\}_n \text{)} \\
&\leq \frac{2}{1-\gamma}((\mathbf{1}_{R \neq R_i} + \mathbf{1}_{R_i \notin \{R_j\}_n})R_\Delta + \epsilon).
\end{aligned}$$

where $\mathbf{1}$ is the indicator function, and $R_\Delta := R_{\text{MAX}} - R_{\text{MIN}}$. We can see that this bound is similar to that of Theorem 7.1(i) but weaker. This because:

(i) The first term of this bound ($\mathbf{1}_{R \neq R_i}$) requires that reward function of the current task (R) be identical to that of a reference task (R_i). In Barreto *et al.* [2018], R_i is taken as the best linear approximation of R. In contrast, the first term of Theorem 7.1(i) ($\mathbf{1}_{T \neq T_{\mathcal{F}}}$) only requires the current task to be expressible as a Boolean composition of past tasks.

(ii) The second term of this bound ($\mathbf{1}_{R_i \notin \{R_j\}_n}$) requires that the reference task (the best linear approximation to the current task) is exactly one of the past tasks. In contrast, the second term of Theorem 7.1(i) ($\mathbf{1}_{R \notin \{R_g\}_{|\mathcal{G}|}}$) only requires the current task to have a single desirable goal.

This suggests that we can can think of the logical composition approach as an efficient way of doing GPI, one which leads to tight performance bounds on the transferred policy (Theorem 7.1(ii)).

7.1.3 Generalisation over a task distribution

We leverage Theorem 7.1 to design an algorithm that combines the SOP approach with goal-oriented learning to achieve fast transfer in lifelong RL. Given an off-policy RL algorithm $\mathscr{A}$, the agent initializes its world value function $\tilde{\mathbf{Q}}$, the task binary vector $\tilde{T}$, and a goal buffer. At the beginning of each episode, the agent computes T_{SOP} and Q_{SOP} for $\tilde{T}$ using the SOP method and its library of learned task vectors and world Q-functions. It then acts using the behaviour policy (ϵ-greedy for example) of $\mathscr{A}$ with $\mathbf{Q}_{SOP}$ for the action-value function if $T_{SOP} = \tilde{T}$, and $\mathbf{Q}_{SOP} \vee \tilde{\mathbf{Q}}$ otherwise.[2] If $T_{SOP} \neq \tilde{T}$, the agent also updates $\tilde{\mathbf{Q}}$ for each goal in the goal buffer using $\mathscr{A}$. Additionally, when the agent reaches a terminal state s, it adds it to the goal buffer and updates $\tilde{T}(s)$ using the reward it receives ($\tilde{T}(s) = \mathbf{1}_{R_M(s,a)=R_{\text{MAX}}}$). Training stops when the agent has reached the desired level of optimality (or after n episodes in practice), after which the agent adds the learned $\tilde{T}$ and $\tilde{\mathbf{Q}}$ to its library if $T_{SOP} \neq \tilde{T}$. We refer to this algorithm as SOPGOL (*Sum Of Products with Goal-Oriented Learning*).

Algorithm 8 shows the full pseudo-code for SOPGOL. Here, $SOP(\tilde{\mathcal{T}}, \tilde{T})$ is the classical sum of products method in Boolean logic. Given a list of binary vectors $\mathcal{T} = [\tilde{T}_1, ..., \tilde{T}_n]$, a Boolean expression for a new binary vector $\tilde{T}$ is obtained as follows:

1. Identify all rows of $\tilde{T}$ with a 1.
2. For each such row: Make a product (conjunction) of all the input variables and make the negation of each variable with a 0 in this row.
3. Take the sum (disjunction) of all these product terms.

The output of SOP is a function B_{EXP} that takes in $|\tilde{\mathcal{T}}|$ variables, and applies disjunctions, conjunctions and negations to them according to the Boolean expression obtained above.

When $\mathcal{G}$ is finite, we show in Theorem 7.2 that SOPGOL generalises over any unknown task distribution after learning only a number of tasks logarithmic in the size of the task space. The lower bound is $\lceil log|\mathcal{G}|\rceil$, since this is the minimum number of tasks that span the task space, as can be seen in Table 7.1 (top) for example. The upper bound is $|\mathcal{G}|$ because that is the dimensionality of the task binary representations $\{0,1\}^{|\mathcal{G}|}$. Since the number of tasks is

[2]Since $\mathbf{Q}_{SOP} \vee \tilde{\mathbf{Q}} = \max\{\mathbf{Q}_{SOP}, \tilde{\mathbf{Q}}\}$, it is equivalent to GPI and hence is guaranteed to be equal or more optimal than the individual value functions. Hence using $\mathbf{Q}_{SOP} \vee \tilde{\mathbf{Q}}$ in the behaviour policy gives a straightforward way of leveraging $\mathbf{Q}_{SOP}$ to learn $\tilde{\mathbf{Q}}$ faster.

Algorithm 8: SOPGOL

```
Input      : off-policy RL algorithm $\mathscr{A}$,                              /* e.g DQN */
             task MDP $M$,
             set of $\epsilon$-optimal task binary representations $\tilde{\mathcal{T}}$,
             set of $\epsilon$-optimal Q-value functions $\tilde{\mathcal{Q}}$.
Initialise : $\tilde{T} : \mathcal{G} \to \{0,1\}$, $\tilde{\mathbf{Q}} : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ according to $\mathscr{A}$, goal buffer $\tilde{\mathcal{G}}$ with
             terminal states observed from a random policy
while $\tilde{\mathbf{Q}}$ is not converged do
    Initialise state $s$ from $M$
    $\mathcal{B}_{EXP} \leftarrow SOP(\tilde{\mathcal{T}}, \tilde{T})$
    $T_{SOP}, \mathbf{Q}_{SOP} \leftarrow \mathcal{B}_{EXP}(\tilde{\mathcal{T}}), \mathcal{B}_{EXP}(\tilde{\mathcal{Q}}^*)$
    $\mathbf{Q} \leftarrow \mathbf{Q}_{SOP}$ if $\tilde{T} = T_{SOP}$ else $\tilde{\mathbf{Q}} \vee \mathbf{Q}_{SOP}$
    $g \leftarrow \arg\max_{g' \in \tilde{\mathcal{G}}} \left( \max_{a \in \mathcal{A}} \mathbf{Q}(s, g', a) \right)$
    while $s$ is not terminal do
        Select action $a$ using the behaviour policy from $\mathscr{A}$: $a \leftarrow \bar{\pi}(s, g)$          /* e.g
         $\epsilon$-greedy */
        Take action $a$, observe reward $r$ and next state $s'$ in $M$
        if $\tilde{T} \neq T_{SOP}$ then
            foreach $g' \in \tilde{\mathcal{G}}$ do
                $\mathbf{r} \leftarrow R_{\text{MIN}}$ if $g' \neq s \in \tilde{\mathcal{G}}$ else $r$
                Update $\tilde{\mathbf{Q}}$ with $(s, g', a, \mathbf{r}, s')$ according to $\mathscr{A}$
        if $s$ is terminal then
            $\tilde{T}(s) \leftarrow \mathbf{1}_{r=R_{\text{MAX}}}$
            $\tilde{\mathcal{G}} \leftarrow \tilde{\mathcal{G}} \cup \{s\}$
        else
            $s \leftarrow s'$
$\mathcal{B}_{EXP} \leftarrow SOP(\tilde{\mathcal{T}}, \tilde{T})$
$\tilde{\mathcal{T}}, \tilde{\mathcal{Q}} \leftarrow (\tilde{\mathcal{T}}, \tilde{\mathcal{Q}})$ if $\tilde{T} = \mathcal{B}_{EXP}(\tilde{\mathcal{T}})$ else $(\tilde{\mathcal{T}} \cup \{\tilde{T}\}, \tilde{\mathcal{Q}} \cup \{\tilde{\mathbf{Q}}\})$
return $\tilde{\mathcal{T}}, \tilde{\mathcal{Q}}$
```

$|\mathcal{M}| = 2^{|\mathcal{G}|}$, we have that the upper bound $|\mathcal{G}| = log|\mathcal{M}|$ is logarithmic in the size of the task space.

Theorem 7.2. *Let $\mathcal{D}$ be an unknown, possibly non-stationary, distribution over a set of tasks $\mathcal{M}(\mathcal{S}, \mathcal{A}, P, \gamma, R_0)$ with finite $\mathcal{G}$. Let $\mathscr{A} : \mathcal{M} \to \mathcal{Q}^*$ be any map from $\mathcal{M}$ to $\mathcal{Q}^*$ such that $\mathscr{A}(M) = \mathbf{Q}^* R_M$ for all M in $\mathcal{M}$. Let*

$$\tilde{\mathcal{T}}_{t+1}, \tilde{\mathcal{Q}}^*_{t+1} = SOPGOL(\mathscr{A}, M_t, \tilde{\mathcal{T}}_t, \tilde{\mathcal{Q}}^*_t) \text{ where } M_t \sim \mathcal{D}(t) \text{ and } \tilde{\mathcal{T}}_0 = \tilde{\mathcal{Q}}^*_0 = \varnothing \; \forall t \in \mathbb{N}.$$

Then,

$$\lceil \log |\mathcal{G}| \rceil \leq \lim_{t \to \infty} N_t \leq |\mathcal{G}| \quad \text{where } N_t := |\tilde{\mathcal{T}}_t| = |\tilde{\mathcal{Q}}^*_t|.$$

Proof. Let $\tilde{T}_t$ be the approximate binary representation of task M_t learned by SOPGOL. We first note that SOPGOL returns $\tilde{\mathcal{T}}_t \cup \{\tilde{T}_t\}$ only if $\tilde{T}_t$ is not in the span of $\tilde{\mathcal{T}}_t$. That is,

$$\tilde{\mathcal{T}}_{t+1} = \tilde{\mathcal{T}}_t \cup \{\tilde{T}_t\} \text{ } iff \text{ } \tilde{T}_t \neq \mathcal{B}_{EXP}(\tilde{\mathcal{T}}_t) \text{ where } \mathcal{B}_{EXP} = SOP(\tilde{\mathcal{T}}_t, \tilde{T}_t).$$

Hence, it is sufficient to show that the number, N, of linearly independent binary vectors, $\tilde{T} \in \{0,1\}^{|\mathcal{G}|}$, that span the Boolean vector space [Subrahmanyam 1964], $GF(2)^{|\mathcal{G}|}$,[3] is bounded by

$$\lceil \log |\mathcal{G}| \rceil \leq N \leq |\mathcal{G}|.$$

This follows from the fact that $\lceil \log |\mathcal{G}| \rceil$ is the size of a minimal set of generators for $GF(2)^{|\mathcal{G}|}$ (as can easily be seen with a Boolean table), and $|\mathcal{G}|$ is its dimensionality.

$\square$

Interestingly, Theorem 7.2 holds even in the case where a new task is expressible in terms of past tasks ($T_{SOP} = \tilde{T}$), but we wish to solve it to a higher degree of optimality than past tasks. In this case, we can pretend $T_{SOP} \neq \tilde{T}$ and learn a new **Q**-function to the desired degree of optimality. We can then add it to our library, and remove any other skill from our library (the least optimal for example).

7.2 Experiments

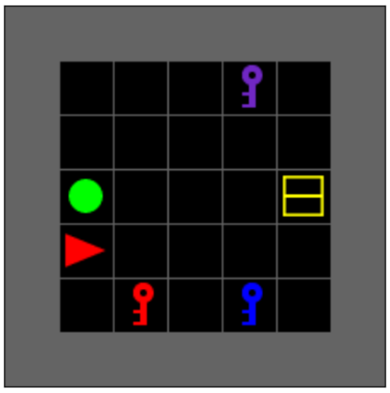


Figure 7.1: PICKUPOBJ domain. The red triangle represents the agent.

Goals	🔑	🔴	🟥	🔑	🔵	🟦	🔑	🟢	🟩	🔑	🟣	🟪	🔑	🟡	🟨
T_a	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
T_b	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
T_c	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
T_d	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Goals	🔑	🔴	🟥	🔑	🔵	🟦	🔑	🟢	🟩	🔑	🟣	🟪	🔑	🟡	🟨
T_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
T_2	0	0	1	0	1	1	1	0	1	1	0	0	1	0	0
T_3	1	1	0	1	0	0	0	1	0	1	1	0	0	1	1

Table 7.1: Binary representation for base (top) and test (bottom) tasks. **0** or **1** corresponds to a goal reward of R_{MIN} or R_{MAX}.

7.2.1 Transfer after pertaining on a set of tasks

In this section, we compare SOPGOL to SOPGOL-transfer, or to SOPGOL-continual. SOPGOL-transfer refers to when no new skill is learned and SOPGOL-continual refers to when

[3]GF(2) is the Galois field with two elements, $(\{0,1\}, +, \cdot)$, where $+ := XOR$ and $\cdot := AND$.

a new skill is always learned using the $SOP\ Q$ estimate to speed up learning, even if the new task could be solved zero shot. Since SOPGOL determines automatically which one to use, we compare whichever one it chooses with the other one in each of our experiments. We note that the results in this section are only demonstrative of our theoretical results, given that they will be averaged over only 4 runs.

Environment

We consider the PICKUPOBJ domain from the MINIGRID environment [Chevalier-Boisvert *et al.* 2018], illustrated by Figure 7.1, where an agent must navigate in a 2D room to pick up objects of various shapes and colours from pixel observations. This type of domain is prototypical in the literature [Nangue Tasse *et al.* 2020a; Barreto *et al.* 2020; van Niekerk *et al.* 2019; Abel *et al.* 2018], because it allows for easy demonstration of transfer learning in many-goal tasks.

The PICKUPOBJ environment is fully observable, where each state observation is a $56 \times 56 \times 3$ RGB image (Figure 1). The agent has 7 actions it can take in this environment corresponding to: 1 - rotate left, 2 - rotate right, 3 - move one step forward if there is no wall or object in front, 4 - pickup object if there is an object in front and no object has been picked, 5 - drop the object in front if an object has been picked and there is no wall or object in front, 6 - open the door in front if there is a closed-door in front, and 7 - close the door in front if there is an opened door in front.

In this domain, there are $|\mathcal{G}| = 15$ goals each corresponding to picking up objects of 3 possible types—box, ball, key—and 5 possible colours—red, blue, green, purple, and yellow (illustrated in Table 7.1). Hence a set of $\lceil \log_2 |\mathcal{G}| \rceil = 4$ base tasks can be selected that can be used to solve all $2^{|\mathcal{G}|} = 32768$ possible tasks under a Boolean composition of goals. For each task, each episode starts with 1 desirable object and 4 other randomly chosen objects placed randomly in the environment. The agent is also placed at a random position with a random orientation at the start of each episode. The agent receives a reward of -0.1 at every timestep, and a reward of 2 when it picks up a desirable object. The environment transitions to a terminal state once the agent picks up any object and the agent observes the picked object. The agent receives a reward of 2 when it picks up desired objects, and -0.1 otherwise.

For all of our experiments in this section, we use deep Q-learning [Mnih *et al.* 2015] as the RL method for SOPGOL and as the performance baseline.

Network architecture and hyperparameters

In our function approximation experiments, we represent each world value function $\tilde{\mathbf{Q}}^*$ with a list of $|\mathcal{G}|$ DQNs, such that the value function for each goal $\tilde{Q}^*_g(s,a) := \tilde{\mathbf{Q}}^*(s,g,a)$ is approximated with a separate DQN. The DQNs used have the following architecture, with the CNN part being identical to that used by Mnih *et al.* [2015]:

1. Three convolutional layers:
   (a) Layer 1 has 3 input channels, 32 output channels, a kernel size of 8 and a stride of 4.
   (b) Layer 2 has 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2.

(c) Layer 3 has 64 input channels, 64 output channels, a kernel size of 3 and a stride of 1.

2. Two fully-connected linear layers:

   (a) Layer 1 has input size 3136 and output size 512 and uses a ReLU activation function.

   (b) Layer 2 has input size 512 and output size 7 with no activation function.

We used the ADAM optimiser with batch size 256 and a learning rate of 10^{-3}. We started training after 1000 steps of random exploration and updated the target Q-network every 1000 steps. Finally, we used ϵ-greedy exploration, annealing ϵ from 0.5 to 0.05 over 100000 timesteps.

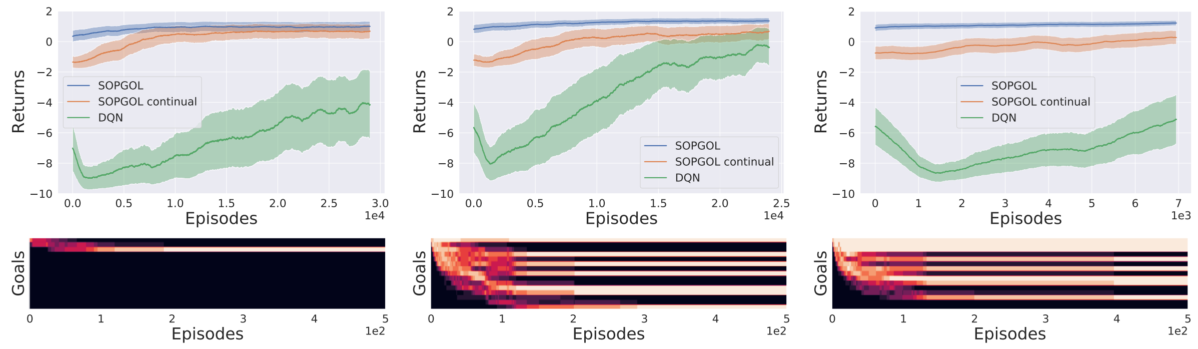
Finally, we used the same DQN architecture and training hyperparameters for the baseline in all experiments.

Setup and results

We first demonstrate transfer learning after pretraining on a set of base tasks—a minimal set of tasks that span the task space. This can be done if the set of goals is known upfront, by first assigning a Boolean label to each goal in a table and then using the rows of the table as base tasks. These are illustrated in Table 7.1 (top). Having learned the ϵ-optimal world value functions for our base tasks, we can now leverage logical composition for transfer learning on test tasks. We consider the three test tasks shown in Table 7.1 (bottom). For each, we run SOPGOL, SOPGOL-continual, and a standard DQN. Figure 7.2 illustrates the results where, as predicted by our theoretical results in Section 7.1.1, SOPGOL correctly determines that the current test tasks are solvable from the logical combinations of the learned base tasks. Its performance from the start of training is hence the best.

Now that we have demonstrated how SOPGOL enables an agent to solve any new task in an environment after training on base tasks, we consider the more practical case where new tasks are not fully expressible as a Boolean expression of previously learned tasks. The agent in this case is pretrained on a set of tasks that do not span the task space, {🟩, 🟪, 🟨, 🔑}, corresponding to the tasks of picking up green objects, blue objects, yellow objects, and keys. We then train the agent with SOPGOL, SOPGOL-transfer, and a standard DQN on the same set of test tasks considered previously (Table 7.1 (bottom)). The results in Figure 7.3 demonstrate how SOPGOL now chooses to learn a task-specific skill after transfer, and hence outperforms SOPGOL-transfer since the test tasks are not entirely expressible in terms of the pretrained ones. Consider Figure 7.3a, for example. The test task is to pick up a yellow box, but the agent has only learned how to pick up red objects, blue objects, yellow objects, and keys. It has not learned how to pick up boxes. However, we note from the inferred Boolean expression ($\widetilde{M}_1$) that the agent correctly identifies that the desired objects are, at the very least, yellow. Without further improvements to this transferred policy (SOPGOL-transfer), we can see that this approach outperforms DQN from the start. This is due to two main factors: (i) the transferred policy navigates to objects more reliably, so takes fewer random actions; and (ii) although the transferred policy does not have a complete understanding of which are the desirable objects, it at least navigates to yellow objects, which are sometimes yellow boxes.

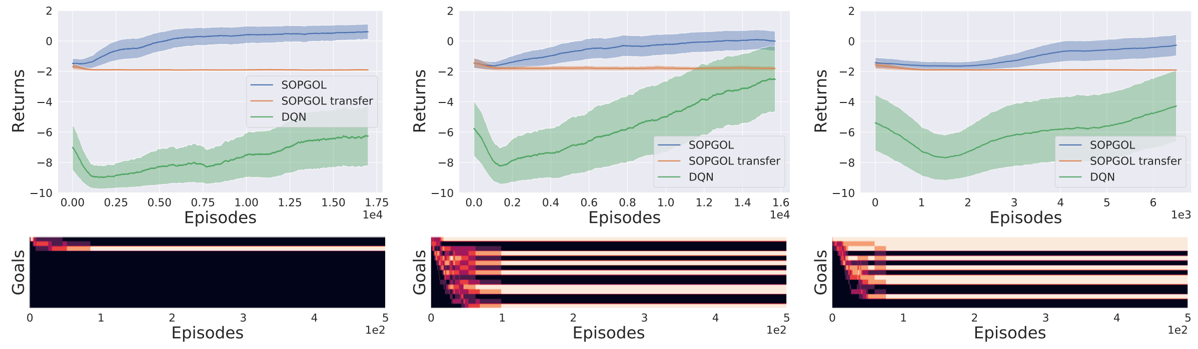
Finally, since SOPGOL is able to determine that the current task is not entirely expressible in terms of its previous tasks (by checking whether $T_{SOP} = \tilde{T}$), it is able to learn a new Q-value



(a) M_1 (b) M_2 (c) M_3

Figure 7.2: Episodic returns (top) and learned binary representations (bottom) for test tasks M_1, M_2 and M_3 after pretraining on the base set of tasks M_a, M_b, M_c and M_d. The shaded regions on the episodic returns indicate one standard deviation over 4 runs. The learned binary representations are similarly averaged over 4 runs, and reported for the first 500 episodes. The initial drop in DQN performance is as a result of the initial exploration phase where the exploration constant decays from 0.5 to 0.05. The Boolean expressions generated by SOPGOL during training for the respective test tasks are:

$$\begin{aligned}
M_1 &= M_a \wedge M_b \wedge M_c \wedge M_d, \\
M_2 &= (M_a \wedge \neg M_b \wedge \neg M_d) \vee (M_a \wedge M_c \wedge M_d) \vee (\neg M_a \wedge M_b \wedge \neg M_c \wedge \neg M_d) \vee \\
&\quad (\neg M_a \wedge \neg M_b \wedge \neg M_c \wedge M_d), \\
M_3 &= (M_a \wedge M_b \wedge M_c) \vee (M_a \wedge \neg M_b \wedge \neg M_d) \vee (M_a \wedge M_c \wedge M_d) \vee (\neg M_a \wedge \\
&\quad M_b \wedge \neg M_c \wedge \neg M_d) \vee (\neg M_a \wedge \neg M_b \wedge \neg M_c \wedge M_d) \vee (\neg M_b \wedge M_c \wedge \neg M_d).
\end{aligned}$$



(a) M_1 (b) M_2 (c) M_3

Figure 7.3: Episodic returns (top) and learned binary representations (bottom) for test tasks M_1, M_2 and M_3 after pretraining on the non-base set of tasks 🟩,🟪,🟨 and 🔑. The shaded regions on the episodic returns indicate one standard deviation over 4 runs. The learned binary representations are similarly averaged over 4 runs, and reported for the first 500 episodes. The initial drop in DQN performance is a result of the initial exploration phase where the exploration constant decays from 0.5 to 0.05. The Boolean expressions generated by SOPGOL for the respective test tasks are:

$$\begin{aligned}
\widetilde{M_1} &= \neg\text{🟩} \wedge \neg\text{🟪} \wedge \text{🟨} \wedge \neg\text{🔑}, \\
\widetilde{M_2} &= (\text{🟩} \wedge \neg\text{🟪} \wedge \neg\text{🟨}) \vee (\neg\text{🟩} \wedge \text{🟪} \wedge \neg\text{🟨} \wedge \text{🔑}) \vee (\neg\text{🟩} \wedge \neg\text{🟪} \wedge \text{🟨} \wedge \text{🔑}) \vee (\neg\text{🟪} \wedge \neg\text{🟨} \wedge \neg\text{🔑}), \\
\widetilde{M_3} &= (\neg\text{🟩} \wedge \neg\text{🟪} \wedge \neg\text{🔑}) \vee (\neg\text{🟩} \wedge \neg\text{🟨}) \vee (\neg\text{🟪} \wedge \neg\text{🟨} \wedge \neg\text{🔑}).
\end{aligned}$$

function that improves on the transferred policy. Additionally, its returns are strictly higher than those of SOPGOL-transfer because SOPGOL learns the new $\mathbf{Q}$-value function faster by using $\mathbf{Q}_{SOP} \vee \tilde{\mathbf{Q}}$ in the behaviour policy.

7.2.2 Transfer during lifelong learning

In this section, we consider the more general setting where the agent is not necessarily given pretrained skills upfront, but is rather presented with tasks sampled from some unknown distribution. We revisit the example given in Section 7.1, but now more concretely by using a stochastic without function approximation.

Environment

We use the Four Rooms domain [Sutton *et al.* 1999], where an agent must navigate in a grid world to particular locations. The goal locations are placed along the sides of the walls and at the centre of rooms (Figure 7.4). This gives a goal space of size $|\mathcal{G}| = 40$ and a task space of size $|\mathcal{M}| = 2^{|\mathcal{G}|} \approx 10^{12}$. The agent can move in any of the four cardinal directions at each timestep, but colliding with a wall leaves the agent in the same location. We add a 5th action for "stay" that the agent chooses to achieve goals. A goal location only becomes terminal if the agent chooses to stay in it. All rewards are 0 at non-terminal states, and 1 at the desirable goals. The transition dynamics are stochastic with a slip probability ($sp = 0.1$). That is, with probability *1-sp* the agent moves in the direction it chooses, and with probability *sp* it moves in one of the other three chosen uniformly at random.

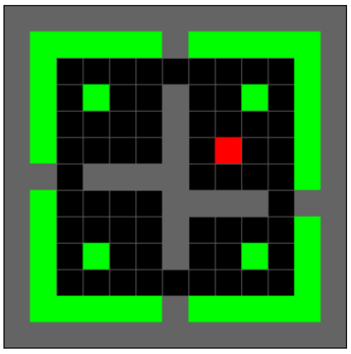
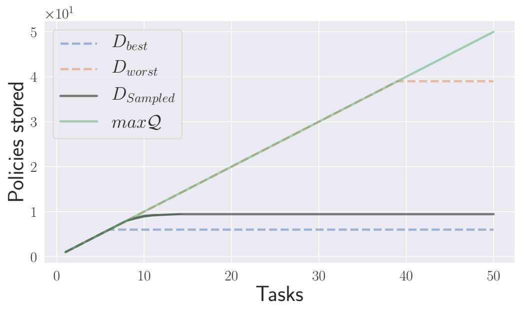


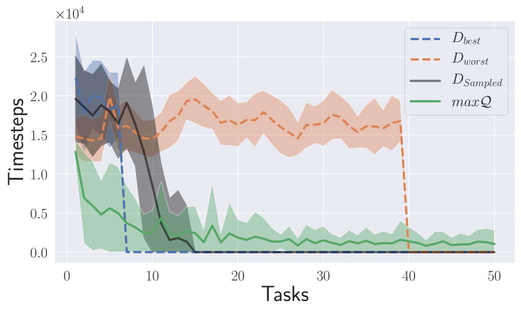
Figure 7.4: 40 goals Four Rooms domain with goals in green and the agent in red.

Setup and results

We demonstrate the ability of SOPGOL to generalise over task distributions by evaluating the approach with the following distributions: (i) $\mathcal{D}_{sampled}$: the goals for each task are chosen uniformly at random over $\mathcal{G}$; (ii) $\mathcal{D}_{best}$: the first $\lceil \log_2 |\mathcal{G}| \rceil$ tasks are the base tasks, while the rest follow $\mathcal{D}_{sampled}$. This distribution gives the agent the minimum number of tasks to learn and store, since the agent learns the base tasks first before being presented with any other task. (iii) $\mathcal{D}_{worst}$: the first $|\mathcal{G}|$ tasks are each defined by a single goal that differs from the previous tasks, while the rest follow $\mathcal{D}_{sampled}$. This distribution forces the agent to learn and store the maximum number of tasks, since none of the $|\mathcal{G}|$ tasks can be expressed as a logical combination



(a) Number of policies learned and stored after solving n tasks.



(b) Number of samples required to learn ϵ-optimal policies for each task.

Figure 7.5: Number of policies learned and samples required for the first 50 tasks of an agent's lifetime in the Four Rooms domain. The shaded regions represent standard deviations over 25 runs.

of the others. We use Q-learning [Watkins 1989] as the RL method for SOPGOL, and Q-learning with $maxQ$ initialisation as a baseline. This has been shown by previous work [Abel *et al.* 2018] to be a practical method of initialising value functions with a theoretically optimal optimism criterion that speeds-up convergence during training. Our results (Figure 7.5) show that SOPGOL enables a lifelong agent to quickly generalise over an unknown task distribution. Interestingly, both graphs show that the convergence speed during a randomly sampled task distribution $\mathcal{D}_{sampled}$ is very close to that of the best task distribution $\mathcal{D}_{best}$. This suggests that there is room to make the bound in Theorem 7.2 even tighter by making some assumptions on the task distribution—an interesting avenue for future work.

7.3 Related Works

There have been several approaches in recent years for tackling the problem of transfer in lifelong RL. Most closely related is the line of work on concurrent skill composition [Todorov 2009; Saxe *et al.* 2017; van Niekerk *et al.* 2019; Hunt *et al.* 2019]. These methods usually focus on multi-goal tasks, where they address the combinatorial amount of desirable goals by composing learned skills to create new ones. Given a reward function that is well approximated by a linear function, Barreto *et al.* [2020] propose a scheme for few-shot transfer in RL by combining GPI and successor features (SF) [Barreto *et al.* 2017]. In general, approaches based on GPI with SFs [Barreto *et al.* 2021] are suitable for tasks defined by linear preferences over features (latent goal states). Given the set of features for an environment, Alver and Precup [2022b] shows that a base set of successor features can be learned, which is sufficient to span the task space. While these approaches also support tasks where goals are not terminal, the smallest number of successor features that must be learned to span the task space is $|\mathcal{G}|$ (the upper-bound in Theorem 7.2). Our work is similar to these approaches in that it can be interpreted as performing GPI with the logical composition of world value functions, which leads to stronger theoretical bounds than GPI with the linear composition of successor features. Finally, none of these works consider the lifelong RL setting where an agent starts with no skill and receives tasks sampled from an unknown distribution (without additional knowledge like base features or true task representations). In contrast, SOPGOL is able to handle this setting with logarithmic bounds on the number of skills needed to generalise over the task distribution (Theorem 7.2).

Other approaches like options [Sutton *et al.* 1999] and hierarchical RL [Barto and Mahadevan 2003] address the lifelong RL problem via temporal compositions. These methods are usually focused on single-goal tasks, where they address the potentially long trajectories needed to reach a desired goal by composing sub-goal skills sequentially [Levy *et al.* 2017; Bagaria and Konidaris 2019]. While they do not consider the multi-goal setting, they can be used in conjunction with concurrent composition to learn how to achieve a combinatorial amount of desirable long horizon goals. Finally, there are also non-compositional approaches [Finn *et al.* 2017; Abel *et al.* 2018; Singh *et al.* 2021], which usually aim to learn the policy for a new task faster by initializing the networks with some pre-training procedure. These can be used in combination with SOPGOL to learn new skills faster.

7.4 Conclusion

In this chapter, we proposed an approach for efficient transfer learning in RL. Our framework, SOPGOL, leverages the Boolean algebra framework developed in Chapters 3 and 6 to determine which skills should be reused in a new task. We demonstrated that, if a new task is solvable using existing skills, an agent is able to solve it with no further learning. However, even if this is not the case, an estimate of the optimal value function can still be obtained to speed up training. This allows agents in a lifelong learning setting to quickly generalise over any unknown (possibly non-stationary) task distribution.

The main limitation of this work is that it only considers tasks with binary goal rewards—where goals are either desirable or not. Although this covers a vast number of many-goal tasks, combining our framework with works on weighted composition [van Niekerk *et al.* 2019; Barreto *et al.* 2020] could enable a similar level of generalisation over tasks with arbitrary goal rewards. Another exciting avenue for future work would be to extend our transfer learning and generalisation results to include temporal tasks by leveraging temporal composition approaches like options. Finally, we note that just like previous work, we rely on the existence of an off-the-shelf RL method that can learn goal-reaching tasks in a given environment. Since that is traditionally very sample inefficient, our framework can be complemented with other transfer learning methods like MaxQInit [Abel *et al.* 2018] to speed up the learning of new skills (over and above the transfer learning and task space generalisation shown here). Our approach is a step towards the goal of truly general, long-lived agents, which are able to generalise both within tasks, as well as over the distribution of possible tasks it may encounter.

Part III

Reliable Agents

In Chapter 8, we demonstrate how the ability to understand task compositions and solve them without further learning can then be leveraged by agents to reliably follow natural language instructions. However, the ambiguity in natural language makes it hard to guarantee that the task specification and resulting compositional behaviours are correct. One solution to this is to use formal languages, such as LTL, which can be verified. In Chapter 9, we propose *skill machines*, a method for agents to provably solve formal languages that are regular (like fragments of LTL). We then show in Chapter 10 that this compositional ability can be leveraged by robots to safely act in the real world.

Chapter 8

Natural language instruction following

This chapter is based on the peer-reviewed works "Learning to Follow Language Instructions with Compositional Policies" [Cohen et al. 2021] and "End-to-End Learning to Follow Language Instructions with Compositional Policies" [Cohen et al. 2022], jointly lead with Vanya Cohen, in collaboration with Nakul Gopalan, Steven James, Matthew Gombolay, Raymond Mooney, and Benjamin Rosman.

In the previous chapters (in Part II), we demonstrated that RL agents equipped with WVFs (Chapter 5) are able to solve a variety tasks in their environment when given explicit instructions as Boolean expressions (Chapter 6) or after infering the Boolean expressions from rewards (Chapter 7). However, it is desirable for an agent to be able to reliably solve a rich variety of problems that can be specified through natural language—for easier human robot interactions.

Prior work uses natural language to specify tasks and also uses that language to enhance generalisation to unseen tasks through imitation and reinforcement learning [Ahn *et al.* 2023; Blukis *et al.* 2020]. However, the high sample complexity of RL-based methods presents difficulties, as agents must map numerous language instructions to corresponding behaviours. Such methods are also tough to train in lifelong settings. The agents lack ways to capitalise on individual previously learned tasks.

One generalisation strategy involves composing novel tasks based on previous ones [Todorov 2009]. Language instructions can guide these compositions, but there is no current mechanism for value function composition guided by language instructions that allows continued learning as novel tasks are introduced. Furthermore many methods do not offer a fully differentiable method for updating both WVFs and language representations as novel tasks arise in a continual learning setting.

To overcome the generalisation issue, we exploit the compositional nature of task specifications and their solutions (in the form of WVFs). This approach is possible as both natural language and the learned WVFs exhibit the property of compositionality. The constituent expressions are possible atomic goal specifications. These goal specifications have WVFs which define the agent's behaviour. We use machine translation approaches to map linguistic task specifications onto corresponding logical expressions, which are then used to combine WVFs to solve the specified tasks.

We make the following main contributions in this Chapter:

1. **Learning to translate:** We develop and evaluate two approaches for learning to follow natural language instructions from weak supervision. We first develop sampling based approach. Here, pretrained compositional policies are connected to a translation model capable of mapping natural language statements to logical expressions specifying compositions of those policies. The language model then samples sequences of tokens to select candidate Boolean expressions to follow, using the cumulative rewards from the composed policies as learning feedback. We then propose a fully differentiable model that learns to compose value functions to solve novel tasks specified using natural language. Here, the model learns to output the correct expressions via soft attention.

2. **Emperical results for the sampling approach:** We demonstrate empirically that pretraining of the translation model on non-task-specific data is sufficient to generate compositional expressions. We found the T5 [Raffel *et al.* 2020] language model sufficient to generate novel Boolean expressions given language commands. This ability to generate novel expressions leads to a significant reduction in samples from the environment after the pretrained model learns to solve just a single task. We detail learning results for 18 tasks in the BabyAI showing that compositional policies, along with a pretrained model, lead to substantial savings in the number of samples required to learn novel tasks. Finally, we also provide ablation results with and without a pretrained language model, and with and without our compositional policies evaluated in the environment.

3. **Emperical results for the end-to-end:** We develop an experimental pipeline to test the efficacy of language-guided compositional reinforcement learning for 18 tasks in the BabyAI domain. We present sample-efficiency results demonstrating that our language-guided compositional RL approach requires 40.6% fewer learning steps than a strong non-compositional pretrained baseline agent.

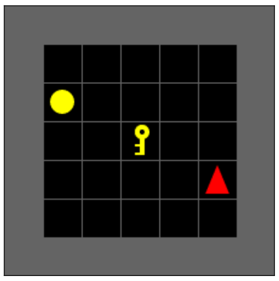
8.1 Translation with transformer models

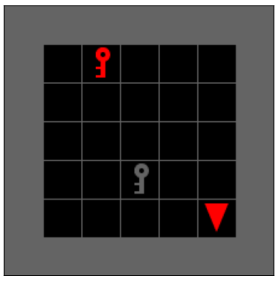
Recent progress in natural language processing (NLP) has demonstrated the effectiveness of large-scale generative pretraining and subsequent fine-tuning on downstream tasks, such as translation, question answering, and classification [Devlin *et al.* 2019; Peters *et al.* 2018; Radford *et al.* 2018]. Subsequent work has shown that scaling both model parameters and pretraining corpus size leads to better transfer learning and generalisation Radford *et al.* [2019].

To map between natural language instructions and Boolean expressions specifying policy compositions, we utilise the T5 sequence-to-sequence model Raffel *et al.* [2020] based on the Transformer architecture Vaswani *et al.* [2017]. The model is pretrained using an unsupervised learning objective on the Colossal Clean Crawled Corpus (C4) Raffel *et al.* [2020], a filtered version of the Common Crawl.[1] The C4 corpus contains 750GB of text, the vast majority of which is fluent English. Raffel *et al.* perform exhaustive ablation studies to develop their pretrained models, which offer good performance on a variety of NLP tasks including translation.

[1]https://commoncrawl.org

Transformer-based models use the self-attention mechanism Vaswani *et al.* [2017] to build sequence representations of text inputs, and to transform those representations into probability distributions over text outputs. As with the original Transformer architecture, the T5 model is composed of both an encoder and decoder stack of self-attention layers to map input sequences to output sequences. Self-attention layers receive input embeddings from lower layers and compose them to form higher-level embeddings.


(a) Language command "pick up the yellow ball" with corresponding logical expression `pickup_yellow` ∧ `pickup_ball`.


(b) Language command "pick up the red key" with corresponding logical expression `pickup_red` ∧ `pickup_key`.

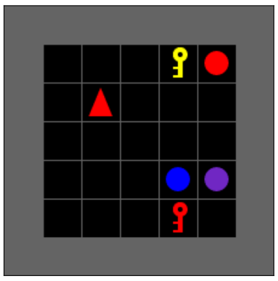

(c) Language command "pick up the red key" with corresponding logical expression `pickup_red` ∧ `pickup_key`.

Figure 8.1: Examples of tasks in the BabyAI `PickUpObj` environment [Chevalier-Boisvert *et al.* 2019]. For each task, there is a target and distractor object. The agent is represented by the red triangle. We also investigate performance when four distrator objects are present. a) The agent must pick up the yellow ball but not the yellow key. To solve this level, the agent must use the intersection of the "pickup" value functions for "yellow" and "ball". b) The agent must pick up the red key while not picking up the grey key. Solving this level requires using the intersection of the "pickup" value functions for "red" and "key". c) The agent must pick up the red key while not being distracted by the yellow key and red ball. Solving this level requires using the intersection of the "pickup" value functions for "red" and "key".

8.2 Language understanding via sampling

Our agent learns to combine pretrained compositional policies by translating BabyAI "mission" statements (e.g. "pick up the blue box") into Boolean algebraic expressions which specify compositions of policies (see domain in Figure 8.1). We limit our investigation to intersections of policies, although the Boolean compositional policies also allow for disjunction and negation. Training begins with training a compositional policy to solve each of the task primitives. The agent can navigate to objects in the BabyAI domain described by three type attributes $\{box, ball, key\}$ and six color attributes $\{red, blue, green, grey, purple, yellow\}$, which yields eighteen possible navigation tasks.

8.2.1 Translating missions to Boolean expressions

We select the smallest of the publicly released T5 models as the pretrained model for our experiments (Table 8.1): the T5-small model which has 60 million parameters and is sufficient for our tasks based upon our empirical exploration.

T5-small model parameters	
Embedding dimension	512
Fully-connected dimension	2048
Attention-heads	8
Encoder, Decoder Layers	6

Table 8.1: The parameters of the T5-small model used in our experiments. To train the model we use the AdamW Optimiser [Loshchilov and Hutter 2019] and a learning rate of 1e-4.

We translate natural language task instructions to Boolean algebraic expressions that represent the task's value function. The Boolean algebraic expressions have tokens and operators. The legal operators are union (disjunction), intersection (conjunction) and negation. The tokens in the Boolean algebraic expressions represent goal value functions that can be composed to create richer tasks. For the BabyAI domain, these tokens represent value functions for picking up objects by type $\{pickup\_box, pickup\_ball, pickup\_key\}$, picking up objects by color $\{pickup\_red, pickup\_blue, pickup\_green, pickup\_grey, pickup\_purple, pickup\_yellow\}$, the logical operators and end-of-sentence tokens $\{and, < s >\}$.

Both the input and output tokens are byte-pair encoding (BPE) subword units [Sennrich *et al.* 2016] learned from the C4 training corpus. For example, the Boolean task algebra token "$pickup\_purple$" is represented by the subwords "$pickup$", "$\_$", "pur", and "ple". Each of the tokens in the Boolean task algebra is represented by one or more BPE subword units. Instead of sampling from the BPE subword units directly, continuations are sampled from the distribution of tokens in the Boolean task algebra. If BPE subwords were sampled directly, at the beginning of training the probability of outputting valid tokens from the Boolean task algebra would be vanishingly small. Decoding stops when the stop token is produced or more than three Boolean algebra tokens have been sampled. We use temperature-based sampling [Ackley *et al.* 1985] to produce translated sequences during training, and greedy sampling during evaluation. For translation model details see Table 8.1.

Given an input mission to the T5-small model (e.g. "pick up the red ball") we can sample a Boolean expression from its output distribution over tokens (e.g. `pickup_red and pickup_ball`). This expression is then parsed and validated for syntactic correctness by a Boolean algebra expression parser. The corresponding WVF is obtained as follows (we omit the value functions' parameters for readability):

$$\mathbf{Q}_{pickup\_red \wedge pickup\_ball} = \min\{\mathbf{Q}_{pickup\_red}, \mathbf{Q}_{pickup\_ball}\}$$

The full process for generating policies from task instructions is illustrated in Figure 8.2. Finally, the agent can maximise over the composed value function to act in the environment: $\pi(s) \in \arg\max_{a \in \mathcal{A}} \max_{g \in \mathcal{G}} \mathbf{Q}(s, g, a)$.

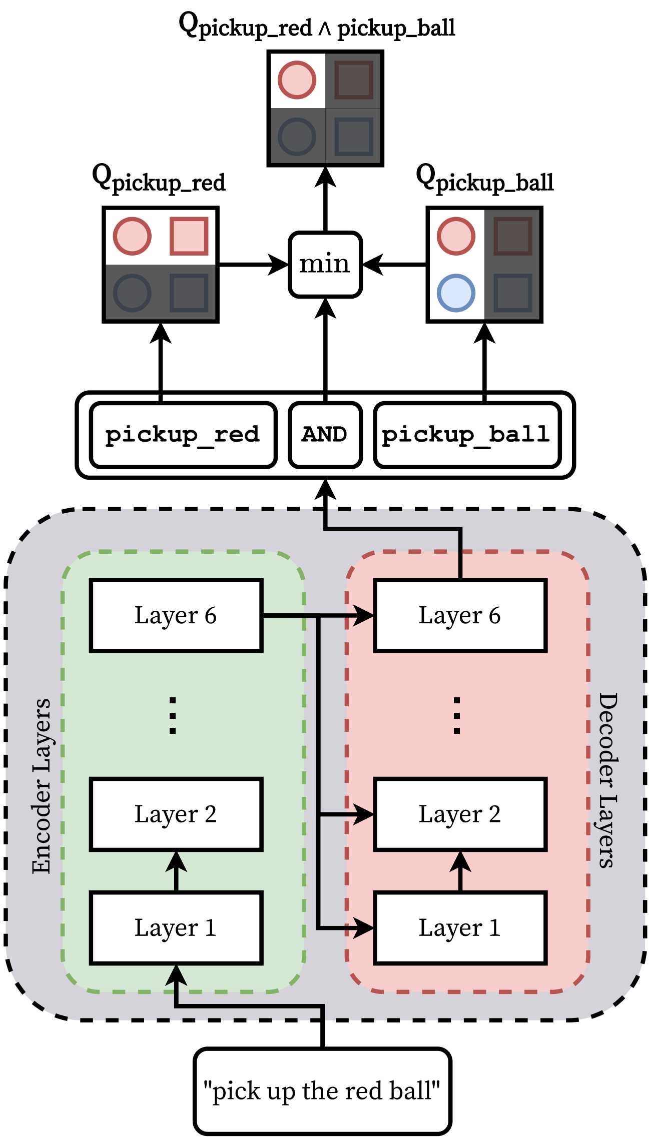


Figure 8.2: The T5-small model first translates the input mission command “pick up the red ball” into a Boolean expression, with variables representing the vocabulary of possible WVFs. Then the intersection of the value functions is computed, resulting in a value function for picking up a red ball in the environment.

8.2.2 Baseline model

The baseline is a non-compositional CNN-DQN [Mnih *et al.* 2015] conditioned on the input mission language. The model is a simplified version of the baseline used by Chevalier-Boisvert *et al.* [2019], and uses a CNN to extract image features and a Gated Recurrent Unit (GRU) [Cho *et al.* 2014] that takes the mission as input and outputs text features. The image and text features are then concatenated and passed through two fully-connected layers to compute the output Q-values.

In contrast to our method, the baseline is a joint model which learns a Q-function conditioned on both image state and language features. As such, its component value function and language representations are not pretrained. Our method learns both tasks and language separately and then learns to combine them compositionally. Likewise, the baseline model does not have explicit compositional structure and must instead learn to condition output Q-values on a combination of image and language features.

8.2.3 Experiments

To assess the agent's ability to generalise compositionally, we evaluate the agent as it learns to solve all available tasks in sequence. In each of ten trials, we randomly shuffle the order in which the 18 tasks are introduced and then train the agent to solve each task one at a time. At iteration 0 of each new task, and every 100 training steps thereafter, the performance of the agent is evaluated using returns from 100 policy roll-outs. A task is considered solved if the agent successfully reaches the goal object in 95 out of 100 roll-outs, at which point the agent is presented with the next task in the sequence. During training, Boolean policy expressions are sampled from the translation model using temperature-based sampling with a temperature of 1.0 to inject randomness in the sampling process. However, when evaluating whether the agent has successfully solved the task, expressions are generated through greedy sampling to only select the most likely continuation tokens without noise.

Learning tasks in series

We adopt four experimental settings to investigate the impact of the different components in our overall system.

First, we consider two strategies for initialising the translation model: we use either 1) the pretrained T5-small model, or 2) its randomly initialised instantiation provided by Wolf *et al.* [2020]. We also consider the effect of the pretrained policies on the overall performance of the agent by 1) using the returns of the policies executed in the environment as a learning signal for the language model, or 2) directly comparing the output of the language model to the ground-truth logical expression. The combinations of language model pretraining and feedback type form the four experiments presented.

During training the environment provides noisy feedback from randomisation of object and agent positions and imperfections in the trained compositional policies. Further, each of the environments has one or four distractor objects sampled uniformly at random from the 18 object types. These objects may have the same type attributes as the target object, in which case the mission command changes from using the definite to the indefinite article (e.g. "pick up a red ball," instead of "pick up the red ball").

During inference the mission statement for the current task is translated to a Boolean expression, which is passed to a Boolean expression parser to determine syntactic correctness. If the expression is not syntactically valid, the agent receives a reward of -1.0. If the expression is valid, the corresponding compositional policy is instantiated and executed in the environment 50 times.

The agent receives the mean reward from these 50 roll-outs. Each episode receives a reward of $+1.0$ for successfully picking up the target object, and -2.0 for failing to pickup the target object within 50 time-steps. The 50 roll-out parameter was chosen by imperially establishing that 50 offers a robust estimation of the mean reward for the instantiated policy.

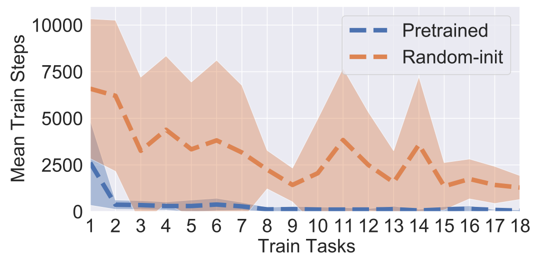
The rewards of $+1.0$ and -2.0 were determined empirically to incentivise the production of optimal Boolean expressions. Without asymmetrically discouragement for picking up the wrong objects, the agent can learn to simply rely on color or object type (rather than both) to act in the environment and attain reward. This behaviour represents a local minimum, where the agent will attain reward in cases where color or object type distinguishes the correct object from a distractor object. By asymmetrically discouraging failure, the agent is incentivised to utilise both the color and object type information to execute more precise policies. The reward scale and sign determines whether the output Boolean expressions are made more or less likely by the cross-entropy loss.

Additionally, we evaluate the effect of environment noise on the translation model's learning. The translation model is separately trained using feedback from logically comparing sampled Boolean expressions to the known true Boolean expressions for those tasks. In this setting, the translation model receives a reward of $+1.0$ for outputting equivalent Boolean expressions and -1.0 for non-equivalent expressions. This removes sources of noise in training the language model: the environmental randomisation, distractor objects, and noise from imperfect policies. However, it differs from purely supervised learning in that learning only occurs on samples from the translation model, produced through temperature sampling.

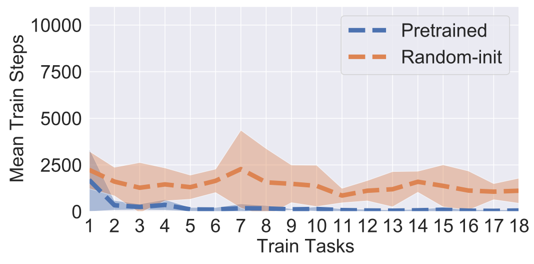
Figure 8.3 depicts the effects of pretraining versus randomly initialising the translation model, when training with feedback from the environment, and with feedback from the equivalence of output logical expressions. The results indicate that whether acting in the real environment or with "perfect" feedback based on logical equivalence, using the pretrained model vastly outperforms the randomly-initialised translation model. While both models see a decrease in the mean train steps across the randomly shuffled 18 tasks, the number of samples required by the pretrained model drops precipitously after learning the first task. Further, in both the pretrained and randomly-initialised cases, learning in the environment is detrimental to model performance, with both the mean number of training steps, and the standard deviation higher for the agent when learning from environmental feedback. The greater number of training steps demonstrates the negative impact of the distractor objects and the agent's imperfect policies on translation model learning. In Figure 8.3e the addition of more distractor objects initially requires more training steps on average to learn new tasks, but with substantially higher variance. However, as more tasks are learned, the pretrained model outperforms the randomly initialised model (as in the single distractor setup). We speculate that this is due to the higher variance in the rewards when using more distractor objects.

Baseline comparison

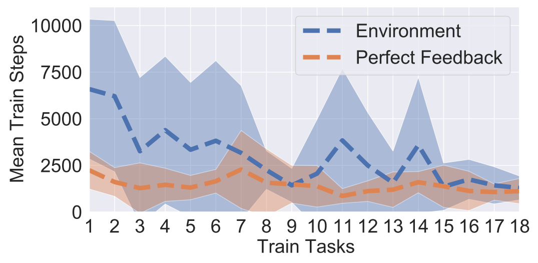
Figure 8.3f compares the number of training steps needed by the BabyAI baseline model to our compositional model. As with the other experiments, results are reported over 10 trials, where the agent learns to solve each task in the task set sequentially. The task order is shuffled between trials. In this experiment the number of training steps is capped at $20,000$ and a single distractor object is present in the environment. Initially, the baseline model succeeds in learning the first



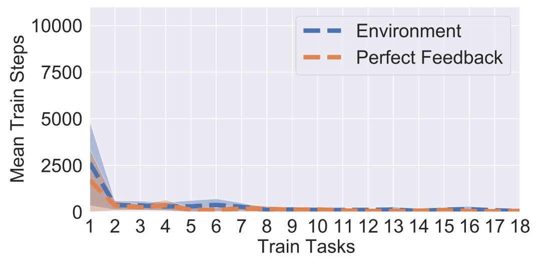
(a) With feedback from evaluating the compositional policies in the environment with one distractor, compares the effect of pretraining on the number of training steps needed to learn each translation from each mission to each Boolean expression.



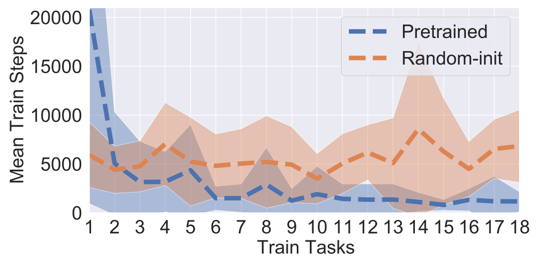
(b) With "perfect" feedback based only on the output Boolean expression, compares the effect of pretraining on the number of training steps needed to learn each translation from each mission to each Boolean expression.



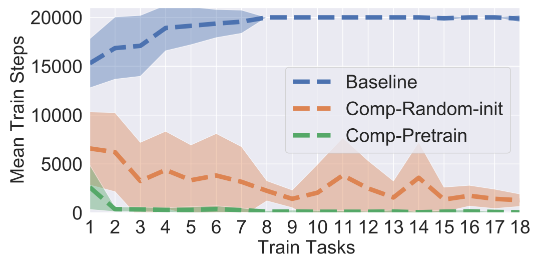
(c) With a randomly-initialised T5 model, compares the learning of the agent from policies evaluated in the environment with one distractor to "perfect" feedback based only on the output Boolean expression.



(d) With a pretrained T5 model, compares the learning of the agent from policies evaluated in the environment with one distractor to "perfect" feedback based only on the output Boolean expression.



(e) With four distractor objects and feedback from evaluating the compositional policies in the environment, compares the effect of pretraining on the number of training steps needed to learn each translation from each mission to each Boolean expression.



(f) Performance of the Baseline model versus Compositional models (both with and without pretraining). All three models are trained using feedback from the environment with a single distractor object.

Figure 8.3: Number of training steps required by various agents to solve each task in a random sequence of tasks. The translation model used is the T5-small model with and without pretraining. Tasks are learned in series, with the same model used across tasks. Task order is randomised across trials. The shaded regions represent the standard deviations over 10 runs.

several tasks. However, this model eventually begins to overfit, and reaches the training step limit for the remaining tasks in the set. Despite a much larger parameter count, neither of the

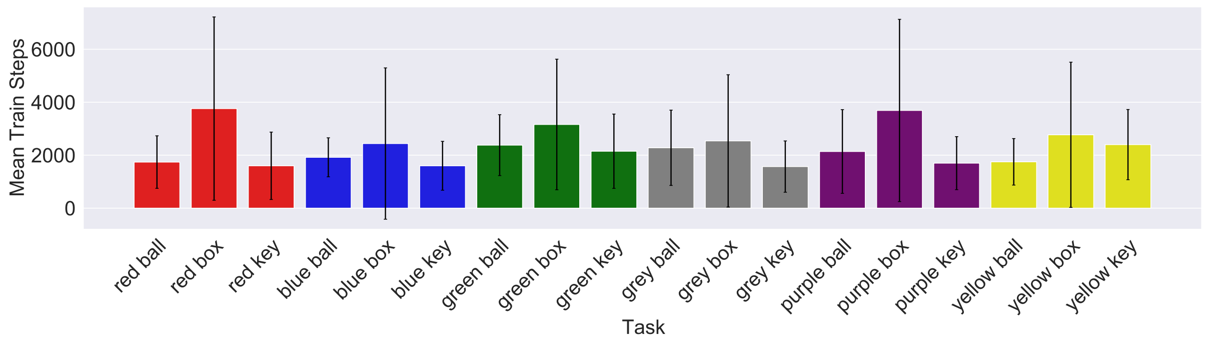


Figure 8.4: The mean training steps needed to learn the translation from each mission to each Boolean expression for each of the 18 potential tasks, when the translation model has perfect feedback from the environment. Rewards are $+1.0$ for equivalent output Boolean expressions and -1.0 for incorrect expressions. Means and Standard Deviations computed over 10 trials.

compositional models overfit, and the compositional model with language model pretraining needs close to zero additional samples to learn the later tasks.

Difficulty of translation tasks

In this experiment, we fine-tune the pretrained translation model using reinforcement learning individually on each of the 18 tasks to compare the relative difficulty of the underlying translations. Unlike in the serial task learning experiment, the translation model learns each task individually with no transfer between tasks. The mean train steps and standard deviations are plotted for 10 trials for each task. The purpose is to determine if learning any of the translations for the tasks are significantly more challenging to learn than the others, which would lead to differential performance when learning certain sequences of tasks.

Figure 8.4 shows the mean train steps needed to learn each translation task based on the logical equivalence of the output expression to the ground-truth expression. The figure shows a similar range of difficulty in translating from each mission statement to each logical expression, indicating that no tasks are overwhelmingly more difficult than the others. However, there are differences in the translation difficulty between certain tasks. Translations for picking up “box” objects consistently require more training samples to learn. The unequal difficulty could be due to differences between the pretrained features for box objects.

8.2.4 Discussion

First, we trained task-specific WVFs (as described in Chapter 5) for a set of preselected atomic tasks from the BabyAI environment [Chevalier-Boisvert *et al.* 2019]. Additionally, we fine-tune a T5 model [Raffel *et al.* 2020] using reinforcement learning to translate natural language instructions into logical expressions that specify the compositions of the task-specific WVFs. The WVFs are then used by the agent to form policies for acting in the environment. Finally, the agent’s collected environment rewards are used as a signal to improve the translation model.

We evaluate the agent by learning a set of compositional tasks in series and observe the number of training steps needed to learn each additional task in the series. Further, we perform ablation studies to understand the effect of model pretraining on web-scale corpora and the stochastic nature of feedback from the environment on sample complexity. With a pretrained T5 model [Raffel *et al.* 2020], the mean number of training steps needed to learn an additional task drops by 86% after learning just one task. Without model pretraining, the mean training steps drops by only 6%, although the number of training steps continues to drop as more tasks are learned.

When learning all available tasks in the environment, the number of training steps needed to learn the final task decreases by 98% for the pretrained model, compared to only 80% for the randomly initialised model. In terms of the fractional improvement in the training steps needed to learn the final task, the pretrained model provides a $10\times$ improvement (2% versus 20%) over the randomly-initialised model.

8.3 Language understanding end-to-end

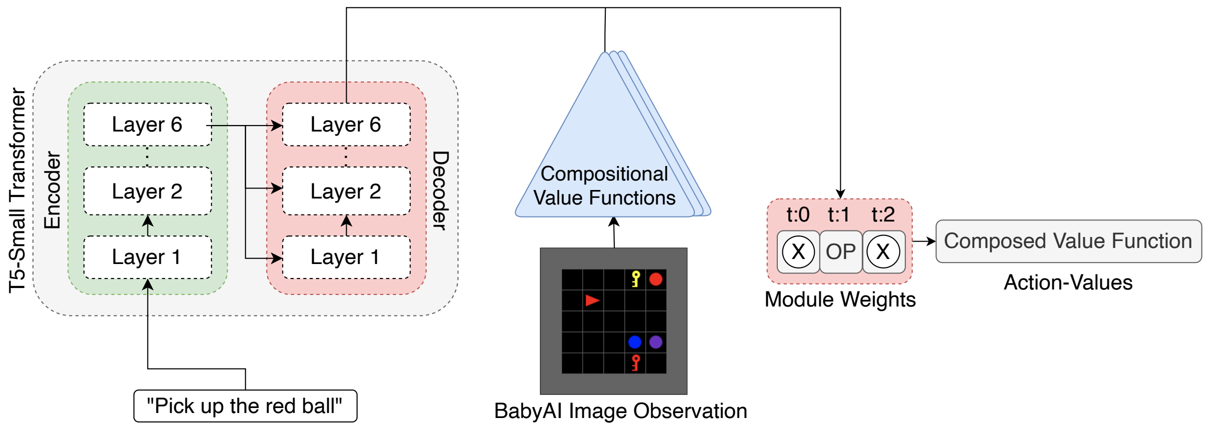


Figure 8.5: The NMVN architecture used to learn WVFs. The network maps image observation inputs and text BabyAI missions to action values by composing the pretrained WVFs using a differentiable attention mechanism. The model learns using the same DQN objective that was used to pretrain the WVFs.

We now propose the *Neural Module Value Network* (NMVN) to enable fully differentiable compositional learning of language-instruction following tasks without demonstrations or imitation learning. We build on the Boolean WVF representations of Section 6.3 and propose a system for learning compositional policies for following language instructions. Such language-conditioned compositional RL policies can be treated as pretrained general-purpose policies to which novel behaviours can be added as needed when solving different novel tasks. Our insight is that language commands implicitly specify the compositional structure of the environment, but without compositional RL representations, this structure cannot be used effectively. Likewise it would be challenging to learn how best to compose the RL representations in the absence of this information. Language, therefore, provides the information required to unlock the utility of compositional RL.

The Neural Module Value Network (NMVN) in Figure 8.5 maps input BabyAI observations and text mission statements to action values. The BabyAI environment observations are image observations of the whole environment and mission commands. The image observations are 54 by 54 pixels and contain 3 color channels . The commands take the form “pick up [the/a] [object]” where [object] contains a composition of type and color attributes (e.g. red box). If more than one valid goal object is present in the environment, the indefinite article “a” is used. T5 produces a sequence of embeddings of a fixed output length conditioned on the input mission command. In the NMVN, these output representations are transformed into attention values over the pretrained WVFs and operations.

As there are two object attribute classes available in BabyAI (color and object type) the model allows for Boolean conjunction expressions with two arguments. At the operator position, to approximate the “intersection” min operation from Chapter 6, we utilise softmin. At each argument position of the decoder sub-module, attention weights are calculated over a vocabulary of tokens describing the object type attributes $\{box, ball, key\}$, object colors $\{red, blue, green, grey, purple, yellow\}$. Each of these tokens corresponds to a pretrained WVF. Attentions are calculated as the softmax over the logits of these tokens.

At each argument position, a hard attention is calculated from the soft-attention by taking the max over attention positions. The attention mechanism learns to select the appropriate arguments to the conjunction expression. However by utilising a hard attention mechanism the model would no longer be differentiable. To remedy this, we approximate the gradient with respect to the hard attention using the straight-through estimator of Jang *et al.* [2017]. In the forward pass, all argument attentions are hard, but the gradients for the backward pass are calculated with respect to a softmax distribution. We find this works well in practice and produces policies which attain our success threshold of 90%, which was not true for the soft attention-based arguments. Composed value function outputs are calculated using a simple recurrence mechanism. We note that this restricts the space of compositions expressible by the model and some Boolean expressions would require additional attention and memory modules to handle operation precedence. Nonetheless this model is sufficient to express the compositions needed to solve the BabyAI pickup tasks.

8.3.1 Training the NMVN

For each episode during training, the NMVN receives an input mission command and environment image observations. At each environment step, the mission command is passed to the T5 model and the image observations are passed to the CNN-DQNs that model the WVFs. These WVFs are then combined using the learned attention mechanism to produce action values for the task. The agent collects experience from the environment based on these WVFs. As in DQN, at each step a random batch of experience tuples is sampled and the NMVN is updated using the DQN temporal difference (TD) loss [Mnih *et al.* 2015; Sutton 1988]. Optimising DQN poses significant challenges and issues with stable learning are outlined in other works [Haarnoja *et al.* 2018c; Maei *et al.* 2009; Mnih *et al.* 2015; Van Hasselt *et al.* 2016]. Through a rigorous hyperparameter search we find hyperparameters that lead to effective learning. A full list of relevant hyperparameters are available in Table 8.2.

NMVN Hyperparameters	
Optimiser	AdamW
Learning rate	5e-7
Batch Size	32
Softmax Temp	0.5
Replay Buffer Size	1e3
ϵ init	0.5
ϵ final	0.1

Table 8.2: The model hyperparameters were determined empirically through grid-search over a set of held-out tasks. The AdamW Optimiser was introduced by Loshchilov and Hutter [2019].

8.3.2 Baselines

CNN-DQN

The naive baseline is a joint language and vision model which learns a single Q-function from scratch for all tasks. The baseline architecture is based on a CNN-DQN [Mnih *et al.* 2015] with a GRU [Cho *et al.* 2014] implementing the BabyAI mission language encoder. The final GRU hidden state is used as the representation for the input mission command. Like the NMVN this network maps image observation inputs and text BabyAI missions to action values, but utilises a FiLM layer [Perez *et al.* 2018] to condition the action values on the inputs. Like the pretrained WVFs and NMVN, this model also learns using the DQN loss. Its component value function and language representations are not pretrained.

Non-compositional pretrained NMVN

We also present results for a more competitive baseline that leverages WVF pretraining. It is based on an ablated NMVN architecture that removes the NMVN's ability to generalise compositionally. This model is initialised with the pretrained basis task WVFs and uses T5 for its language encoder. Unlike the NMVN that generates differentiable compositions of two WVFs, this model simply takes a linear combination of the pretrained WVFs, using an attention mechanism conditioned on the mission language. This baseline does not have the “and” operator or multiple variables. It is equivalent to the NMVN with only one variable. Also unlike the NMVN, the pretrained WVFs are not frozen and are instead finetuned during training to better model the true value function. As in the NMVN, the T5 model is also finetuned. Each WVF is labeled only with a unique ID. In this setting, the underlying compositional-semantic structure of the task attributes is hidden and the model must discover the relevant language-WVF mappings.

To assess the comparative advantage of understanding the ground-truth semantics of the WVFs, we propose a variant of the non-compositional pretrained baseline, labeled as ”with structure”. Analogous to the full NMVN, this model is supplied with labels for WVFs, granting it equivalent access to the fundamental semantics and structure of the object attributes. Given that the mission text and WVFs share a common vocabulary, this change increases performance by allowing the model to learn attentions in fewer samples.

8.3.3 Experiments

We evaluate the NMVN and baselines in a 7×7 BabyAI environment. We assess the agent's ability to perform tasks that involve intersections of attributes. For example,"pick up the red ball" requires the intersection of the basis "pick up the red object" and "pick up the ball" WVFs. In each episode, there are four distractor objects randomly sampled and placed in the environment. The agent must correctly navigate to the object specified by the mission and pick it up with 100 environment steps. All experiments were run on NVIDIA RTX A5000 GPUs with 24GB of VRAM.

Four-attribute composition We show results for learning sequences of tasks that test the agent's ability to generalise to novel compositions of attributes and entirely novel attributes. From the eighteen intersection tasks that can be formed by composing the three object and six color attributes, we select sets of four tasks that are each composed of four attributes, two color attributes and two object type attributes. These are then learned in series, with a shared model learning all four tasks sequentially. For example, the attributes present in the tasks "pick up the red ball" and "pick up the green key" can be combined to form two new tasks: "pick up the green ball" and "pick up the red key". We evaluate the agent by plotting the number of training steps required to attain a 90% success rate on each task, up to a maximum of one thousand steps. Once a task reaches the success rate threshold, the next task in the sequence starts training. The reported training steps do not include the warm-up steps required to fill the replay buffer with environment experience tuples before training starts.

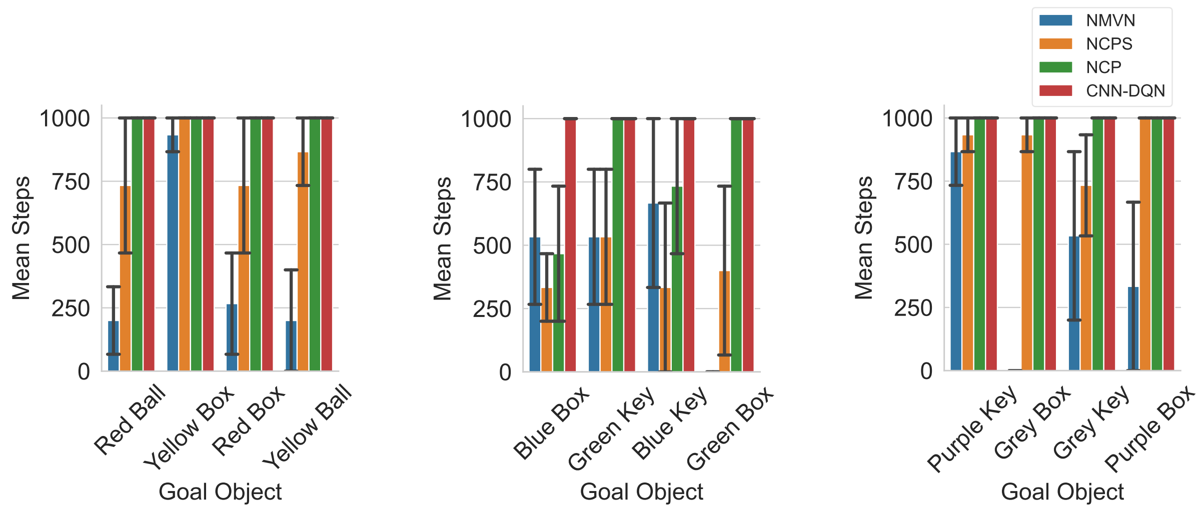
To succeed in this setting, the agent must learn the correct mapping between the mission language and the WVFs (which correspond to environment attributes) and be able to generalise this mapping to the held-out combinations of attributes. If the agent learns a compositional mapping between these spaces, then it should be able to generalise to novel combinations of attributes with few additional learning samples. Further, by using the transfer-learning capabilities of the language model, the agent can in principle generalise to entirely unseen attributes.

Intersection task series While the previous experiment assesses the agent's ability to generalise to chosen combinations of attributes, this experiment investigates agent performance across a larger number of tasks. The agent learns each of the eighteen intersection tasks in series. The tasks are randomly shuffled across seeds. The agent has a budget of 100 training steps to Optimise each task before it starts learning the next task in the series. While the agent only trains on a single task at a time, it is evaluated every 50 timesteps on all of the intersection tasks. This experiment assesses the agent's ability to effectively learn series of tasks from very few samples and how well the agent generalises its experience from learning single tasks to performing the rest of the tasks. One advantage of compositional representations is that they should generalise better than non-compositional ones and be less likely to overfit. If the agent suffers catastrophic forgetting [Kirkpatrick *et al.* 2017] or otherwise overfits to the individual tasks, then the mean success rate across all the tasks will drop. If the agent generalises properly, the agent should achieve high average performance across the tasks. After learning a few tasks, learning more tasks should not significantly increase the agent's performance as the agent should have already generalised systematically to unseen combinations of attributes.

Results

We present results for our two experimental settings, the four-attribute series tasks, and the intersection series tasks, across four models: our proposed model (NMVN), the non-compositional pretrained baseline with access to the ground-truth WVF semantics (NCPS), the non-compositional pretrained baseline without this information (NCP), and the randomly initialised deep Q-learning baseline model (CNN-DQN). All results are averaged over three random seeds.

Four-attribute composition As shown in Figure 8.6, the NMVN learns to complete each sequence of tasks to a 90% success rate in fewer than $1{,}000$ steps on average. For two tasks, the NMVN required no additional training steps to solve those tasks. By contrast, the non-compositional baselines require many more steps to solve the tasks overall, and in some cases do not solve the tasks before the $1{,}000$ step timeout. For some tasks such as "blue box" and "blue key" the non-compositional baselines perform competitively to the NMVN. The non-compositional pretrained baselines solve these tasks in fewer steps than the NMVN. However, both models fail to generalise to the other tasks in the series. In the case of the "green box" task, the NMVN solves this task with no additional training steps after training on the other tasks in the series. Neither baseline demonstrates effective transfer across the tasks or generalises in zero additional training steps to any of the tasks. As expected the CNN-DQN agent does not solve any of the tasks within the timeout and the NCP agent requires many more steps on average to learn the tasks, and reaches the timeout threshold on most tasks.

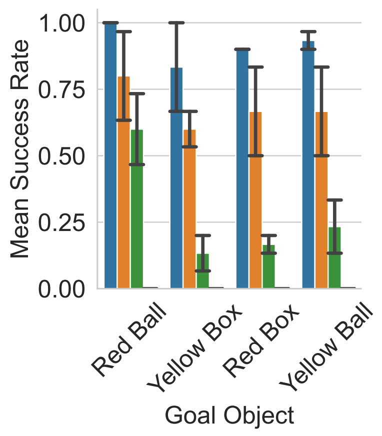


(a) Task series for the attributes $\{box, ball, yellow, red\}$.

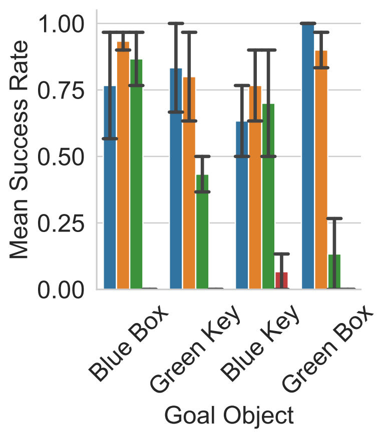
(b) Task series for the attributes $\{box, key, green, blue\}$.

(c) Task series for the attributes $\{key, box, purple, grey\}$.

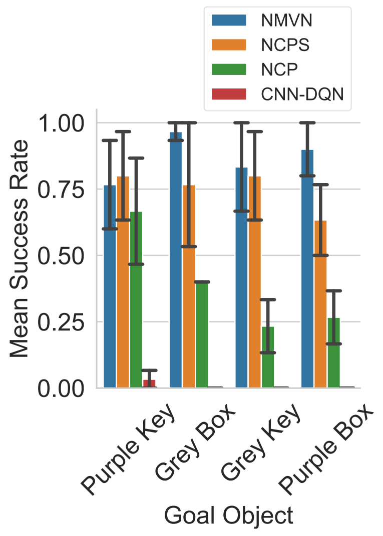
Figure 8.6: In each plot, the pickup task is learned for each object in series from left to right using a shared model. Mean environment steps needed to attain a 90% success rate on each task and 80% confidence intervals are computed across three trials. Lower is better. The two non-compositional pretrained baselines are denoted NCPS and NCP. Some tasks require no additional training steps for the agent to succeed. The NMVN model achieved a 40.6% average reduction in the number of steps required to complete all twelve tasks over the NCPS baseline. Note that in some cases the NMVN requires no additional training steps to solve the tasks.



(a) Task set for the attributes $\{box, ball, yellow, red\}$.



(b) Task set for the attributes $\{box, key, green, blue\}$.



(c) Task set for the attributes $\{key, box, purple, grey\}$.

Figure 8.7: In each plot, the pickup task is learned for each object in series from left to right using a shared model. Mean final success rates for each task and 80% confidence intervals are computed across three trials. Higher is better. While the NMVN does not outperform the NCPS baseline for every task, relative to the baselines its success rate tends to increase as it trains on more tasks. Because the tasks time out after $1{,}000$ time steps, the success rates at completion are not always 90%.

Figure 8.7 plots the success rate attained by each agent when it finished training on each task. The NMVN attains a higher success rate than the baselines on most of the tasks. There are some tasks where the baselines perform equivalently to the NMVN or better, but the NMVN generalises more effectively to the following tasks in those series. For example, in Figure 8.7a, the NMVN outperforms the baselines on the final task in the series. Across the 12 tasks examined, between the NMVN and the non-compositional pretrained baseline with access to the environment attributes, there was a 40.6% average reduction in the number of steps required to complete each task.

Intersection task series Models that generalise compositionally should transfer learn between combinations of attributes, allowing them to perform better on held-out tasks with fewer samples. Figure 8.8 plots the performance of the NMVN and baselines as they sequentially learn all 18 intersection tasks. While each model is trained on one task at a time, its overall performance on all tasks is evaluated. Despite being trained on each task for only 100 steps, the NMVN demonstrates transfer learning between the tasks, attaining a high level of success on the overall set of tasks. The NCPS baseline can finetune its value functions and has access to the underlying WVF semantics. However, it only reaches the same success rate as the NMVN after training on almost all of the intersection tasks. This indicates that there is value in having access to the semantics of the WVFs. The NMVN appears to reach a higher level of performance before experiencing a slight drop around the seventh task. This decline may be attributable to overfitting or noise, especially considering that the sequence of tasks is shuffled randomly. The NCP

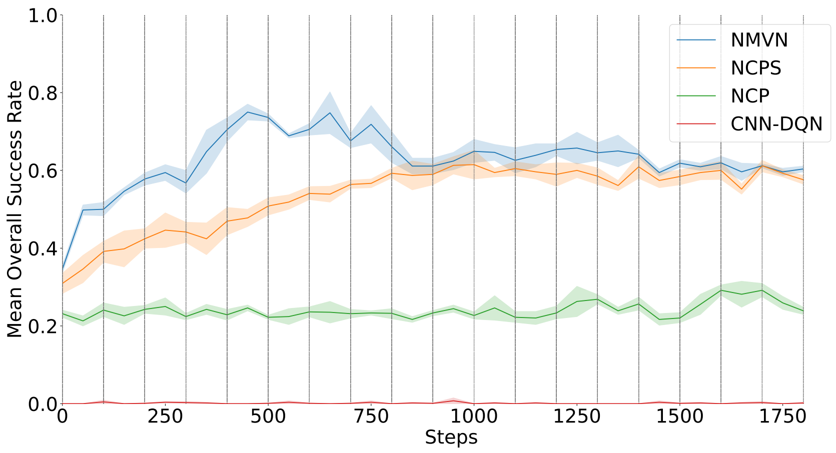


Figure 8.8: Mean success rates for training on 18 intersection tasks sequentially and 80% confidence intervals across three trials. Higher is better. The vertical lines indicate the start and end of training on each task. The model is trained on each task for 100 steps and evaluated on all tasks every 50 steps. The NMVN model outperforms the NCPS model when both are trained on a limited number of tasks. As expected, once both models have been trained on all tasks, their overall success rates across the 18 tasks converge. The NMVN appears to peak in performance before decreasing slightly around the seventh task. This could be due to overfitting or noise as the task order is randomly shuffled. The two non-compositional pretrained baselines are denoted NCPS and NCP.

baseline does not attain a high success rate or improve as it is trained on more tasks. Without this information, the learning problem is significantly more difficult, which limits the agent's ability to generalise from a few samples.

8.3.4 Discussion

We demonstrated a fully differentiable method to ground language instructions to compositional RL policies that demonstrates zero-shot task solving. This represents an improvement over the state-of-the-art language to value function models such as [Ahn *et al.* 2023; Driess *et al.* 2023] as these require demonstrations to learn their policies and have no mechanisms to update the policy and the LLMs at the same time. Our model provides the ability to update the right value functions and an LLM for any given task while utilising the compositional task structure. We believe such ability is critical if we desire agents capable of continually learning language and behaviour over time in scenarios such as lifelong learning.

In this work, we use pretrained WVFs and learned to ground to these functions with finetuning. While our method leverages the known compositional structure of the environment, this structure could be discovered while learning the basis WVFs. We would like to demonstrate the learning of value functions and language representations simultaneously from scratch. In principle our model supports this, but in practice we have found it challenging for the model to attain a high success rate on all but the simplest domains. To make the problem easier, a curriculum could be leveraged to provide implicit information about the environment structure.

8.4 Related works

Our work is situated within the paradigm of reinforcement learning, where novel tasks are specified using natural language and the agent is required to solve the task in the fewest possible steps. Previous approaches have solved this problem using end-to-end architectures that are learned or improved using reinforcement learning and a set of demonstrations [Anderson *et al.* 2018; Blukis *et al.* 2020; Chaplot *et al.* 2018]. A problem with such approaches is a lack of compositionality in the learned representations. For example, learning to navigate to a red ball does not help the agent to learn to identify and navigate to a blue ball. Moreover, demonstrations are hard to come by from users especially when the user does not know the desired behaviour. Approaches that translate language commands to a symbolic representation and then plan to reach a goal can demonstrate compositionality due to the pre-specified symbolic representations [Dzifcak *et al.* 2009; Williams *et al.* 2018; Gopalan *et al.* 2018]. However, these works do not allow the agent to learn policies, and use pre-specified symbols and a model for planning.

Compositional representation learning has been demonstrated in the computer vision and language processing tasks using Neural Module Networks (NMN) [Andreas *et al.* 2016; Hu *et al.* 2018], but we explicitly desire compositional representations both for the reinforcement learning policies and the language command. Kuo *et al.* [2021] do demonstrate compositional representations for policies, but they depend on a pre-trained parser to learn this representation. On the other hand, we use large language models [Raffel *et al.* 2020] and compositional policy representations to demonstrate compositionality in our representations and the ability to solve novel unseen instruction combinations.

Compositional policy representations have been demonstrated using value function compositions, which were first demonstrated by Todorov [2007] using the linearly solvable MDP framework. Moreover, zero-shot disjunction [van Niekerk *et al.* 2019] and approximate conjunction [Haarnoja *et al.* 2018a; van Niekerk *et al.* 2019; Hunt *et al.* 2019] have been shown using entropy regularised RL. Nangue Tasse *et al.* [2020a] demonstrate zero-shot optimal composition for all three logical operators—disjunction, conjunction, and negation—in the stochastic shortest path problems. In contrast, our approach extends ideas from Chapter 6 to solve novel commands specified using natural language.

Recent works like *SayCan* use language models and pretrained language-conditioned value functions to solve language specified tasks few-shot and zero-shot [Ahn *et al.* 2023]. Shridhar *et al.* [2021] uses pretrained image-text representations to perform pick-and-place tasks on a robot. Other work incorporates learning from demonstration and language with large-scale pretraining to solve robotics tasks Driess *et al.* [2023]; Brohan *et al.* [2022]. However, these works use learning from demonstration as opposed to reinforcement learning. Moreover, their methodology is unsuitable for continual learning settings where both the RL value functions and language embeddings are improved over time as novel tasks are introduced.

8.5 Conclusion

In this chapter, we investigated two approaches for instruction following that leverages the compositional representations present in both the Boolean Task algebra value functions and in large, pretrained language models. Since regular value functions cannot in general be optimally combined to produce desired behaviours [Todorov 2009; van Niekerk *et al.* 2019], we leveraged WVFs since they admit composability. By ensuring that both the language and control aspects of the agent are compositional, we demonstrated that an agent can use its existing knowledge to quickly solve new tasks using very few samples. Such sample efficiency is critical in developing long-lived agents that are required to learn and act in the real world.

However, the proposed approach assumes given basis WVFs. We would like to create a fully differentiable model that learns to create a space of compositional goals and to map language to the space of Boolean algebra over the learned compostional goals.

To adress this goal, we develop the Neural Module Value Network (NMVN), a fully differentiable model that learns to follow language instructions using WVFs. We leverage compositionality to tackle the complex challenges of reinforcement learning (RL) methods, providing a pathway to more generalisable and efficient RL agents. We present results for our model and non-compositional baselines on challenging tasks that require the agent to generalise to novel tasks from few samples. Our approach is able to solve novel tasks to a high success rate, often in hundreds of time steps. Some tasks are even solved zero-shot. Not only does our model represent a substantial improvement over standard CNN-DQN agents, but it also outperforms a competitive non-compositional pretrained baseline, requiring 40.6% fewer learning steps across a series of tasks. Overall these results demonstrate the utility of combining language representations with compositional RL.

In summary, the findings presented underscore the significance of integrating language representations with compositional RL, highlighting the potential for creating more robust and versatile agents capable of mastering a wide array of tasks and solving them reliably.

Chapter 9

Skill machines: Temporal logic composition

This chapter is based on the published work
"Skill Machines: Temporal Logic Skill Composition in Reinforcement Learning" [Nangue Tasse et al. 2024], in collaboration with Devon Jarvis, Steven James, and Benjamin Rosman.

In the previous chapter, we demonstrated how agents can leverage their zero-shot logical composition abilities to follow natural language instructions. We achieved this by decomposing the problem into two steps: (i) *language understanding* where the agent learns from trial and error how to translate from natural language instructions to formal language Boolean expressions; then (ii) *instruction following* where the agent composes its learned basis skills according to his language understanding to solve the task. However, we only demonstrated this for intersection instructions over two attributes (shape and color). Even then, this proved challenging due to the difficulty of language understanding.

If we ultimately want agents that are trully reliable with guarantees on their language understanding and instruction following, then one promising approach is using formal languages directly to specify instructions (Chapter 2.3). However, as we described in the introduction and background chapters, this is particularly challenging because of the need for simultaneous spatial (logics) and temporal composition abilities.

In this chapter, we aim to address the highlighted problem by combining the logical composition framework with reward machines to develop an agent capable of both zero-shot spatial *and* temporal composition. We particularly focus on temporal logic composition, such as linear temporal logic (LTL) [Pnueli 1977], allowing agents to sequentially chain and order their skills while ensuring certain conditions are always or never met. We make the following main contributions:

1. **Skill machines:** We propose *skill machines (SM)*, which are finite state machines (FSM) that encode the solution to any task specified using any given regular language (such as regular fragments of LTL) as a series of Boolean compositions of *skill primitives*—composable sub-skills for achieving high-level goals in the environment. An SM is defined by translating the regular language task specification into an FSM, and defining the skill to use per FSM state as a Boolean composition of pretrained skill primitives.

2. **Zero-shot and few-shot learning using skill machines:** By leveraging reward machines (RM) [Icarte *et al.* 2018]—finite state machines that encode the reward structure of a task—we show how an SM can be obtained directly from an LTL task specification, and prove that these SMs are *satisficing*—given a task specification and regular reachability assumptions, an agent can successfully solve the task while adhering to any constraints. We further show how standard off-policy RL algorithms can be used to improve the resulting behaviours when optimality is desired. This is achieved with no new assumption in RL.

3. **Emperical and qualitative results:** We demonstrate our approach in several environments, including a high-dimensional video game and a continuous control environment. Our results indicate that our method is capable of producing near-optimal to optimal behaviour for a variety of long-horizon tasks without further learning, including empirical results that far surpass all the representative state-of-the-art baselines.

9.1 Skill composition for temporal logic tasks

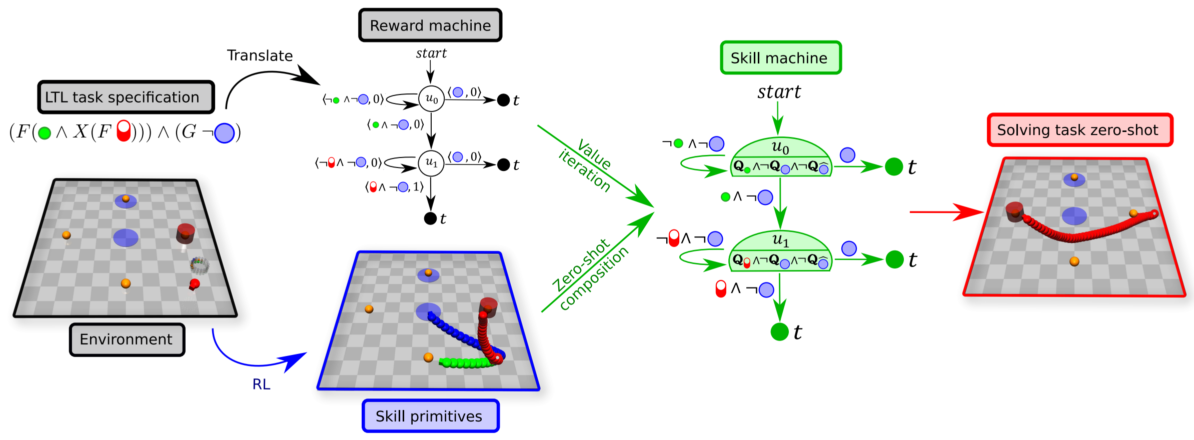


Figure 9.1: Illustration of our framework: Consider a continuous environment containing a robot (*red sphere*) with 3 LiDAR sensors that it uses to sense when it has reached a red cylinder (🔴), a green button (🟢), or a blue region (🔵). The agent first learns *skill primitives* to reach these 3 objects (the red, green, and blue sample trajectories obtained from them respectively). Then given any task specification over these 3 objects, such as: "Navigate to a button and then to a cylinder while never entering blue regions" with LTL specification (F(🟢 ∧ X(F 🔴))) ∧ (G ¬🔵), the agent first translates the *LTL task specification* into an RM, then plans which spatial skill to use at each temporal node using *value iteration* and composes its skill primitives to obtain said spatial skills (culminating in a *skill machine*), and finally uses them to solve the task without further learning. The RM is obtained by converting the LTL expression into an FSM using Spot [Duret-Lutz *et al.* 2016], then giving a reward of 1 for accepting transitions and 0 otherwise. The nodes labeled t in the RM and SM represent terminal states (sink/absorbing states where no transition leaves the state).

Given that any formal language can be used to specify a temporal logic task (Section 2.3), for clarity, we will focus our attention on tasks specified using regular fragments of LTL—such as co-safe LTL [Kupferman and Vardi 2001]. To ensure that the optimal policy for the resulting product MDP (Definition 2.10) is also the policy that maximises the probability of satisfying the LTL specification, we will henceforth assume that the environment dynamics are deterministic.

To describe our approach, we use the *Safety Gym Domain* [Ray *et al.* 2019] shown in Figure 9.1 as a running example. Here, the agent moves by choosing a direction and force ($\mathcal{A} = \mathbb{R}^2$) and observes a real vector containing various sensory information like joint velocities and distance to the objects in its surrounding ($\mathcal{S} = \mathbb{R}^{60}$). The LTL tasks in this environment can then be defined over 3 propositions: $\mathcal{P} = \{$🔴, 🟢, 🔵$\}$, where each proposition is true when the agent is $\epsilon = 1$ metre near its respective location.

Now consider an agent that has learned how to "Go to the cylinder" (F 🔴), "Go to a button" (F 🟢), and "Go to a blue region" (F 🔵). Say the agent is now required to solve the task with LTL specification $(F($🟢 $\wedge X(F$ 🔴$))) \wedge (G \neg$🔵$)$. Using prior LTL transfer works [Vaezipoor *et al.* 2021; Jothimurugan *et al.* 2021; Liu *et al.* 2022], the agent would have learned options for solving the first 3 tasks, but then would be unable to transfer those skills to immediately solve this new task. This is because the new task requires the agent to first reach a button that is not in a blue region (*eventually* satisfy 🟢 $\wedge \neg$🔵) while not entering a blue region along the way (*always* satisfy $\neg$🔵). Similarly, it then must eventually satisfy 🔴 $\wedge \neg$🔵 while never satisfying 🔵. However, all 3 options previously learned will enter a blue region if it is along the agent's path. Hence the agent will need to learn new options for achieving 🟢 $\wedge \neg$🔵 and 🔴 $\wedge \neg$🔵 where the option policies never enter 🔵 along the way.

In general, we can see that there are $2^{2^{\mathcal{P}}}$ possible Boolean expressions the agent may be required to *eventually* satisfy (spatial curse), and $2^{2^{\mathcal{P}}}$ possible Boolean expressions the agent may be required to *always* satisfy (temporal curse). This highlights the curses of dimensionality we aim to simultaneously address. In this section, we will introduce skill primitives as the proposed solution for addressing the aforementioned curses of dimensionality. We will then introduce skill machines as a state machine that can encode the solution to any temporal logic task by leveraging skill primitives.

9.1.1 From environment to primitives

We desire an agent capable of learning a sufficient set of skills that can be used to solve new tasks, specified through LTL, with little or no additional learning. We introduce the notion of *primitives* which aims to address the spatial and temporal curses of dimensionality we discussed above:

Spatial curse of dimensionality: To address this, we can learn WVFs (the composable value functions described in Chapter 5) for *eventually* achieving each proposition, then compose them to *eventually* achieve the Boolean expression over the propositions. For example, we can learn WVFs for tasks F 🔴, F 🟢, and F 🔵. However, the product MDP for LTL specified tasks have different states and dynamics (see Definition 2.10). Hence, they do not satisfy the assumptions for zero-shot logical composition (Chapter 6). To address this problem, we define task primitives below. These are product MDPs for achieving each proposition when the agent decides to terminate, and share the same state space and dynamics. We then define skill primitives as their corresponding WVFs.

Temporal curse of dimensionality: To address this, we introduce the concept of *constraints* $\mathcal{C} \subseteq \{\widehat{p} \mid p \in \mathcal{P}\}$ which we use to augment the state space of task primitives[1]. In a given environment, a constraint is a proposition that an agent may be required to *always* keep True or *always* keep False in some FSM state of a temporal logic task. Equivalently, it is a proposition which may never change across the trajectory of the agent in the FSM state. When contradicted it may transition the agent into a failure FSM state (an FSM sink state from which it can never solve the task). For example, some tasks like $(F($🟢$\wedge X(F$ 🔴$))) \wedge (G\, \neg$🔵$)$ require the agent to solve a task $F($🟢$\wedge X(F$ 🔴$))$ while never setting 🔵 to True $(G\, \neg$🔵$)$. By setting the 🔵 proposition as a constraint when learning a primitive (e.g achieving 🟢), the agent keeps track (in its cross-product state) of whether or not it has reached a blue region in a trajectory that did not start in a blue region. That is, in an episode where the agent does not start in a blue region but later goes through a blue region and terminates at a button, the agent will achieve the goal $g = \{$🟢$, \widehat{\text{🔵}}\} \in 2^{\mathcal{P} \cup \mathcal{C}}$. We henceforth assume the general case $\mathcal{C} = \{\widehat{p} \mid p \in \mathcal{P}\}$ for our theory, then later consider different choices for $\mathcal{C}$ in our experiments.

We now formally define the notions of task primitives and skill primitives such as "Go to a button":

Definition 9.1 (Primitives)**.** *Let $\langle \mathcal{S}, \mathcal{A}, P, \gamma, \mathcal{P}, L\rangle$ represent the environment the agent is in, and $\mathcal{C}$ be the set of constraints. We define a task primitive here as an MDP $M_p = \langle \mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, P_\mathcal{G}, R_p, \gamma\rangle$ with absorbing states $\mathcal{G} = 2^{\mathcal{P} \cup \mathcal{C}}$ that corresponds to achieving a proposition $p \in \mathcal{P} \cup \mathcal{C}$, where $\mathcal{S}_\mathcal{G} := (\mathcal{S} \times 2^{\mathcal{C}}) \cup \mathcal{G}$; $\mathcal{A}_\mathcal{G} := \mathcal{A} \times \mathcal{A}_\tau$, where $\mathcal{A}_\tau = \{0, 1\}$ is an action that terminates the task;*

$$P_\mathcal{G}(\langle s, c\rangle, \langle a, a_\tau\rangle) := \begin{cases} l' \cup c & \textit{if } a_\tau = 1 \\ \langle s', c'\rangle & \textit{otherwise} \end{cases};\; R_p(\langle s, c\rangle, \langle a, a_\tau\rangle) := \begin{cases} 1 & \textit{if } a_\tau = 1 \textit{ and } p \in l' \cup c \\ 0 & \textit{otherwise} \end{cases},$$

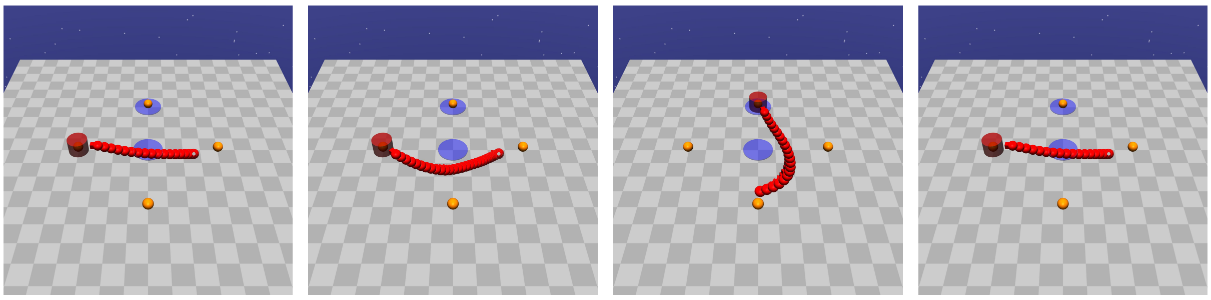
where $s' \sim P(\cdot|s, a)$, $l = L(s)$, $l' = L(s')$, and $c' = c \cup ((\widehat{l} \oplus \widehat{l'}) \cap \mathcal{C})$.

A skill primitive is defined as $\mathbf{Q}^*_p(\langle s, c\rangle, g, \langle a, a_\tau\rangle)$, the WVF for the task primitive M_p.

The above defines the state space of primitives to be the product of the environment states and the set of constraints, incorporating the set of propositions that are currently true. The action space is augmented with a terminating action following Barreto *et al.* [2019] and Nangue Tasse *et al.* [2020a], which indicates that the agent wishes to achieve the goal it is currently at, and is similar to an option's termination condition [Sutton *et al.* 1999]. The transition dynamics update the environment state s and the set of violated constraints c when any other action is taken. Here, the labeling function is used to return the set of propositions l and l' achieved in s and s' respectively. Any constraint present exclusively in l or l' is added to c, since it has not been kept always True or always False. Finally, the agent receives a reward of 1 when it terminates in a state where the proposition p is true, and 0 otherwise. These primitives can then be learned using any suitable RL algorithm, such as Q-learning in the tabular case (Algorithm 9 shows the full pseudo-code). Figure 9.2 shows examples of the resulting optimal policies when the set of constraints is empty and non-empty.

[1]The notation $\widehat{p}$ represents when a *literal* (a proposition $p \in \mathcal{P}$ or its negation $\neg p$) is being used as a constraint. Similarly, we will use $\widehat{\mathcal{P}}$ or $\widehat{\sigma}$ respectively when the literals in a set $\mathcal{P}$ or Boolean expression σ are constraints.

Since all task primitives $\mathcal{M}_{\mathcal{G}} := \{M_p \mid p \in \mathcal{P} \cup \mathcal{C}\}$ share the same state space, action space, dynamics, and rewards at non-terminal states, the corresponding skill primitives $\mathcal{Q}^*_{\mathcal{G}} := \{\mathbf{Q}^*_p \mid p \in \mathcal{P} \cup \mathcal{C}\}$ can be composed to achieve any Boolean expression over $\mathcal{P} \cup \mathcal{C}$ (Chapter 6). We next introduce *skill machines* which leverages skill primitives to encode the solution to temporal logic tasks.



(a) $M_{🔴} \wedge \neg M_{🔵}$, $g = \{🔴, 🟢, \widehat{🔵}\}$, $r = 1$ (b) $M_{🔴} \wedge \neg M_{🔵} \wedge \neg M_{\widehat{🔵}}$, $g = \{🔴, 🟢\}$, $r = 1$ (c) $M_{🔴} \wedge M_{🔵} \wedge \neg M_{\widehat{🔵}}$, $g = \{🔴, 🟢, 🔵\}$, $r = 1$ (d) $M_{🔴} \wedge \neg M_{🔵} \wedge \neg M_{\widehat{🔵}}$, $g = \{🔴, 🟢, \widehat{🔵}\}$, $r = 0$

Figure 9.2: Effect of constraints on primitives ($C = \{\widehat{🔵}\}$). We show compositions of task primitives (for example $M_{🔴} \wedge \neg M_{🔵}$ where the agent needs to achieve $🔴 \wedge \neg 🔵$), trajectories, goal reached (g), and reward obtained (r) when following: (a-c) Optimal policies; and (d) a non-optimal policy.

9.1.2 Skill machines

We now have agents capable of solving any logical composition of task primitives $\mathcal{M}_{\mathcal{G}}$ by learning only their corresponding skill primitives $\mathcal{Q}^*_{\mathcal{G}}$ and using the zero-shot composition operators (Chapter 6). Given this compositional ability over skills, and reward machines that expose the reward structure of tasks, agents can solve temporally extended tasks with little or no further learning. To achieve this, we define a skill machine (SM) as a representation of logical and temporal knowledge over skills.

Definition 9.2 (Skill Machine)**.** *Let $\langle \mathcal{S}, \mathcal{A}, P, \gamma, \mathcal{P}, L\rangle$ represent the environment the agent is in, and $\mathcal{Q}^*_{\mathcal{G}}$ be the corresponding skill primitives with constraints $\mathcal{C}$. Given a reward machine $R_{\mathcal{SA}} = \langle \mathcal{U}, u_0, \delta_u, \delta_r\rangle$, a skill machine is a tuple $\mathcal{Q}^*_{\mathcal{SA}} = \langle \mathcal{U}, u_0, \delta_u, \delta_Q\rangle$ where $\delta_Q : U \to [\mathcal{S}_{\mathcal{G}} \times \mathcal{A}_{\mathcal{G}} \to \mathbb{R}]$ is the state-skill function defined by:*

$$\delta_Q(u)(\langle s, c\rangle, \langle a, 0\rangle) := \max_{g \in \mathcal{G}} \mathbf{Q}^*_{\sigma_u}(\langle s, c\rangle, g, \langle a, 0\rangle),$$

and $\mathbf{Q}^*_{\sigma_u}$ is the composition of skill primitives $\mathcal{Q}^*_{\mathcal{G}}$ according to a Boolean expression $\sigma_u \in 2^{2^{\mathcal{P}\cup\mathcal{C}}}$

.

For a given state $s \in \mathcal{S}$ in the environment, the set of constraints violated $c \subseteq \mathcal{C}$, and state u in the skill machine, the skill machine computes a skill $\delta_Q(u)(\langle s, c\rangle, \langle a, 0\rangle)$ that an agent can use to take an action a. The environment then transitions to the next state s' with true propositions l'—where $\langle s', c'\rangle \leftarrow P_{\mathcal{G}}(\langle s, c\rangle, \langle a, 0\rangle)$ and $l' \leftarrow L(s')$—and the skill machine transitions to $u' \leftarrow \delta_u(u, l')$. This process is illustrated in Figure 9.3 for the skill machine shown in Figure 9.1.

Algorithm 9: Q-learning for skill primitives

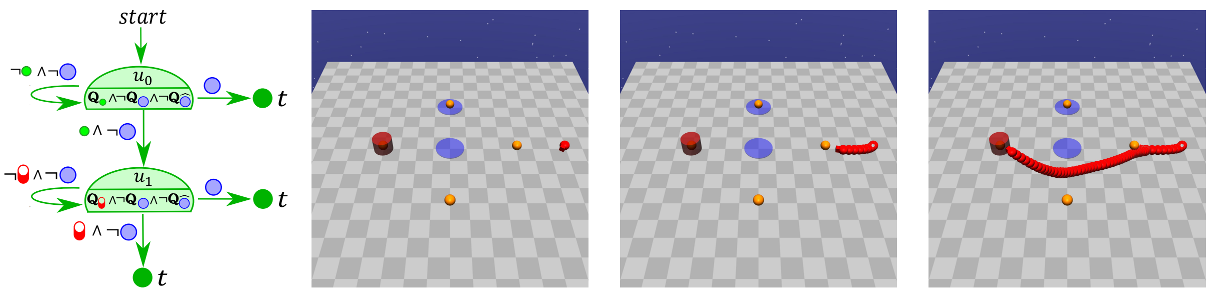
```
Input      : $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C}, \gamma, \alpha, R_{\text{MAX}} = 1, R_{\text{MIN}} = 0$
Initialise : $\mathbf{Q}_{MAX}(\langle s, c\rangle, g, \langle a, a_\tau\rangle)$ and $\mathbf{Q}_{MIN}(\langle s, c\rangle, g, \langle a, a_\tau\rangle)$, goal buffer $G = \{\emptyset\}$
1  foreach episode do
2      Observe initial state $s \in \mathcal{S}$ and true propositions $l \in 2^{\mathcal{P}}$, sample $c \in 2^{\mathcal{C}}$ and $g \in G$
3      while episode is not done do
4          $\langle a, a_\tau\rangle \leftarrow \begin{cases} \arg\max_{\langle a, a_\tau\rangle} \mathbf{Q}_{MAX}(\langle s, c\rangle, g, \langle a, a_\tau\rangle) & \text{if } Bernoulli(1-\epsilon) = 1 \\ \text{sample } \langle a, a_\tau\rangle \in \mathcal{A} \times \{0, 1\} & \text{otherwise} \end{cases}$
5          Execute $a$ and observe next state $s'$ and true propositions $l'$
6          Get true constraints $c' \leftarrow c \cup ((\widehat{l} \oplus \widehat{l'}) \cap \mathcal{C})$
7          if $(a_\tau = 1)$ then $G \leftarrow G \cup \{l' \cup c\}$
8          foreach $\mathbf{Q} \in \{\mathbf{Q}_{MAX}, \mathbf{Q}_{MIN}\}$ do
9              if $(a_\tau \neq 1)$ then $r \leftarrow 0$
10             if $(a_\tau = 1$ and $\mathbf{Q} = \mathbf{Q}_{MAX})$ then $r \leftarrow R_{\text{MAX}}$
11             if $(a_\tau = 1$ and $\mathbf{Q} = \mathbf{Q}_{MIN})$ then $r \leftarrow R_{\text{MIN}}$
12             foreach $g' \in G$ do
13                 $\bar{r} \leftarrow R_{\text{MIN}}$ if $(a_\tau = 1$ and $g' \neq l' \cup c)$ else $r$
14                 if ($s'$ is terminal or $a_\tau = 1$) then
15                     $\mathbf{Q}(\langle s, c\rangle, g', \langle a, a_\tau\rangle) \overset{\alpha}{\leftarrow} \bar{r}$
16                 else
17                     $\mathbf{Q}(\langle s, c\rangle, g', \langle a, a_\tau\rangle) \overset{\alpha}{\leftarrow} \left[\bar{r} + \gamma \max_{\langle a', a'_\tau\rangle} \mathbf{Q}(\langle s', c'\rangle, g', \langle a', a'_\tau\rangle)\right]$
18         $s \leftarrow s'$ and $c \leftarrow c'$
19         if $(a_\tau = 1)$ then terminate episode
20 $\mathcal{Q}_{\mathcal{G}} \leftarrow \emptyset$
21 foreach $p \in P \cup \mathcal{C}$ do
22     $\mathbf{Q}_p(\langle s, c\rangle, g, \langle a, a_\tau\rangle) := \mathbf{Q}_{MAX}(\langle s, c\rangle, g, \langle a, a_\tau\rangle)$ if $(p \in g)$ else
        $\mathbf{Q}_{MIN}(\langle s, c\rangle, g, \langle a, a_\tau\rangle)$
23     $\mathcal{Q}_{\mathcal{G}} \leftarrow \mathcal{Q}_{\mathcal{G}} \cup \{\mathbf{Q}_p\}$
24 return $\mathcal{Q}_{\mathcal{G}}$
```

Remarkably, because the Boolean compositions of skill primitives are optimal, there always exists a choice of skill machine that is optimal with respect to the corresponding reward machine, as shown in Theorem 9.1, which demonstrates that SMs can be used to solve tasks without having to relearn action level policies:

Theorem 9.1. *Let $\pi^*(s,u)$ be the optimal policy for a task $M_\mathcal{T}$ specified by an RM $R_{\mathcal{SA}}$. Then there exists a corresponding skill machine $\boldsymbol{\mathcal{Q}}^*_{\mathcal{SA}}$ such that*

$$\pi^*(s,u) \in \arg\max_{a\in\mathcal{A}} \delta_Q(u)(\langle s,c\rangle,\langle a,0\rangle).$$

Proof. Define the skill per SM state $\mathbf{Q}^*_u$ to be the Boolean composition of skill primitives that satisfy the set of propositions $g \in 2^{\mathcal{P}\cup\mathcal{C}}$, where g is the set of propositions satisfied and constraints violated when following $\pi^*(s,u)$. Then $\pi^*(s,u) \in \arg\max_{a\in\mathcal{A}} \delta_Q(u)(\langle s,c\rangle,\langle a,0\rangle)$ since $\mathbf{Q}^*_u$ is optimal using Nangue Tasse *et al.* [2022a] and optimal policies maximise the probability reaching goals (since the rewards are non-zero only at the desirable goal states, where they are 1). □

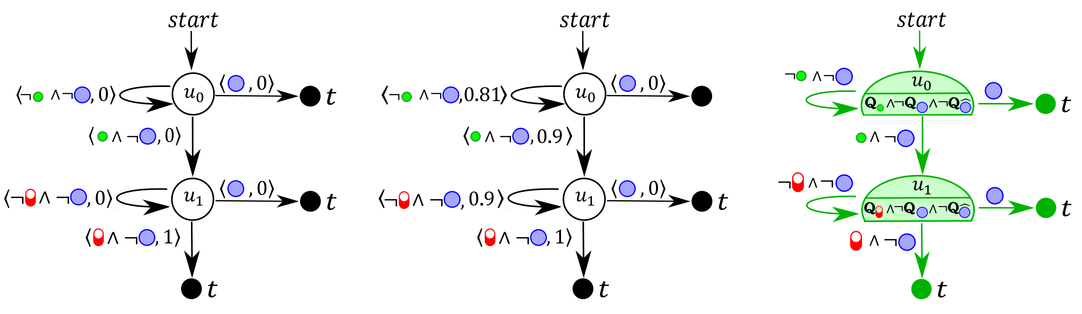


(a) Skill machine (b) $c=\emptyset, l=\emptyset$, $\mathbf{Q}_{\sigma_u} = \mathbf{Q}_{🟢} \wedge \neg\mathbf{Q}_{🔵} \wedge \neg\mathbf{Q}_{\widehat{🔵}}$ (c) $c{=}\emptyset$, $l{=}\{🟢\}$, $\mathbf{Q}_{\sigma_u} = \mathbf{Q}_{🟥} \wedge \neg\mathbf{Q}_{🔵} \wedge \neg\mathbf{Q}_{\widehat{🔵}}$ (d) $c=\emptyset$, $l=\{🟥, 🟢\}$, episode terminates

Figure 9.3: Execution of a skill machine in the Safety Gym domain. (a) An example skill machine; (b) A snapshot of the environment at the initial state. In this state, no constraint has been reached ($c=\emptyset$), no proposition is true ($l=\emptyset$), the SM is at state $u=u_0$, and the composed skill outputted by the SM is $\mathbf{Q}_{\sigma_u} = \mathbf{Q}_{🟢} \wedge \neg\mathbf{Q}_{🔵} \wedge \neg\mathbf{Q}_{\widehat{🔵}}$ (which the agent uses to act in the environment); (c) The trajectory of the agent until it achieves $🟢 \wedge \neg 🔵 \wedge \neg \widehat{🔵}$. In the current environment state, no constraint has been reached ($c=\emptyset$), the agent is at a green button ($l=\{🟢\}$), the SM transitions to state $u=u_1$, and the composed skill outputted by the SM is $\mathbf{Q}_{\sigma_u} = \mathbf{Q}_{🟥} \wedge \neg\mathbf{Q}_{🔵} \wedge \neg\mathbf{Q}_{\widehat{🔵}}$ (which the agent uses to act in the environment); (d) The trajectory of the agent until the agent achieves $🟥 \wedge \neg 🔵 \wedge \neg \widehat{🔵}$. In the current environment state, no constraint has been reached ($c=\emptyset$), the agent is at the red cylinder and a green button ($l=\{🟥, 🟢\}$), the SM transitions to the terminal state t, and the episode terminates.

9.1.3 From temporal logic tasks to skill machines

In the previous section, we introduced skill machines and showed that they can be used to represent the logical and temporal composition of skills needed to solve tasks specified by reward machines. However, we only proved their existence—for a given task, how can we acquire an SM that solves it?



(a) Reward machine (b) Value iterated RM (c) Skill machine

Figure 9.4: The reward machine, value iterated reward machine (using $\gamma = 0.9$) and skill machine for the task with LTL specification $(F($🟢$\wedge X(F$🔴$))) \wedge (G\,\neg$🔵$)$. The agent composes its skill primitives to achieve $\sigma_{\mathcal{P}} \wedge \neg\sigma_{\mathcal{C}} = ($🟢$\wedge\neg$🔵$)\wedge\neg(\widehat{🔵})$ at u_0 and $\sigma_{\mathcal{P}} \wedge \neg\sigma_{\mathcal{C}} = ($🔴$\wedge\neg$🔵$)\wedge\neg(\widehat{🔵})$ at u_1.

Algorithm 10: Skill machine from reward machine

```
Input     : $\mathcal{Q}_{\mathcal{G}}, \langle \mathcal{U}, u_0, \delta_u, \delta_r, \rangle, \gamma$
Initialise : RM value function $Q(u, \sigma)$, value iteration error $\Delta = 1$
Let $\mathcal{B}(u) :=$ the set of Boolean expressions defining the RM transitions $\delta_u(u, \cdot)$
/* Value iteration */
while $\Delta > 0$ do
    $\Delta \leftarrow 0$
    for $u \in \mathcal{U}$ do
        for $\sigma \in \mathcal{B}(u)$ do
            $v' \leftarrow \delta_r(u, \sigma) + \gamma \max_{\sigma'} Q(\delta_u(u, \sigma), \sigma')$
            $\Delta = \max\{\Delta, |Q(u, \sigma) - v'|\}$
            $Q(u, \sigma) \leftarrow v'$

/* Skill machine's skill function */
for $u \in \mathcal{U}$ do
    $\sigma_{\mathcal{P}}, \sigma_{\mathcal{C}} \leftarrow argmax_{\sigma'}\, Q(u, \sigma'),\ \bigvee\{\widehat{\sigma} \mid Q(u, \sigma) = 0\}$
    $\mathbf{Q}_{\sigma_{\mathcal{P}} \wedge \neg\sigma_{\mathcal{C}}} \leftarrow$ composition of $\mathcal{Q}_{\mathcal{G}}$ as per the Boolean expression $\sigma_{\mathcal{P}} \wedge \neg\sigma_{\mathcal{C}}$
    $\delta_Q(u)(\langle s, c\rangle, \langle a, 0\rangle) \leftarrow \max_{g \in \mathcal{G}} \mathbf{Q}_{\sigma_{\mathcal{P}} \wedge \neg\sigma_{\mathcal{C}}}(\langle s, c\rangle, g, \langle a, 0\rangle)$
return $\langle \mathcal{U}, u_0, \delta_u, \delta_Q \rangle$
```

Zero-shot via planning over the RM: To obtain the SM that solves a given RM, we first plan over the reward machine (using value iteration, for example) to produce action-values for each transition. We then select skills for each SM state greedily by applying Boolean composition to skill primitives according to the Boolean expressions defining: (i) the transition with the highest value (propositions to eventually satisfy); and (ii) the transitions with zero value (constrains to always satisfy). This process is illustrated by Figure 9.4, and the full pseudo-code is shown in Algorithm 10. Since the skills per SM state are selected greedily, the policy generated by this SM is *recursively optimal* [Hutsebaut-Buysse *et al.* 2022]—that is, it is locally optimal (optimal for each sub-task) but may not be globally optimal (optimal for the overall task). Interestingly, we show in Theorem 9.2 that this policy is also *satisficing* (reaches an accepting state) if we assume global reachability—all FSM transitions (that is all Boolean expressions $\sigma \in 2^{2^{\mathcal{P}}}$) are achievable from any environment state. This is a more relaxed version of the assumption "any state is reachable from any other state" that is required to prove optimality in most RL algorithms, since an agent cannot learn an optimal policy if there are states it can never reach.

Theorem 9.2. *Let $R_{\mathcal{SA}} = \langle \mathcal{U}, u_0, \delta_u, \delta_r \rangle$ be a satisfiable RM where all the Boolean expressions σ defining its transitions are in negation normal form (NNF) [Robinson and Voronkov 2001] and are achievable from any state $s \in \mathcal{S}$. Define the corresponding SM $\mathcal{Q}^*_{\mathcal{SA}} = \langle \mathcal{U}, u_0, \delta_u, \delta_Q \rangle$ with*

$$\delta_Q(u)(\langle s, c \rangle, \langle a, 0 \rangle) \mapsto \max_{g \in \mathcal{G}} \mathbf{Q}^*_{(\sigma_{\mathcal{P}} \wedge \neg \sigma_{\mathcal{C}})}(\langle s, c \rangle, g, \langle a, 0 \rangle)$$

where $\sigma_{\mathcal{P}} := argmax_{\sigma}\ Q^*(u, \sigma)$, $\sigma_{\mathcal{C}} := \bigvee \{ \widehat{\sigma} \mid Q^*(u, \sigma) = 0 \}$, and $Q^*(u, \sigma)$ is the optimal Q-function for $R_{\mathcal{SA}}$. Then, $\pi(s, u) \in \arg\max_{a \in \mathcal{A}} \delta_Q(u)(\langle s, c \rangle, \langle a, 0 \rangle)$ is satisficing.

Proof. This follows from the optimality of Boolean skill composition and the optimality of value iteration, since each transition of the RM is satisfiable from any environment state.

□

Theorem 9.2 is critical as it provides soundness guarantees, ensuring that the policy derived from the skill machine will always satisfy the task requirements.

Few-shot via RL in the environment: Finally, in cases where the composed skill $\delta_Q(u)(\langle s, c \rangle, \langle a, 0 \rangle)$ obtained from the approximate SM is not sufficiently optimal, we can use any off-policy RL algorithm to learn the task-specific skill $Q_{\mathcal{T}}(s, u, a)$ few-shot. This is achieved by using the maximising Q-values $\max\{\gamma Q_{\mathcal{T}}, (1-\gamma)\delta_Q\}$ in the exploration policy during learning. Here, the discount factor γ determines how much of the composed policy to use. Consider Q-learning, for example: during the ϵ-greedy exploration, we use $a \leftarrow \arg\max_{\mathcal{A}} \max\{\gamma Q_{\mathcal{T}}, (1-\gamma)\delta_Q\}$ to select greedy actions. This improves the initial performance of the agent where $\gamma Q_{\mathcal{T}} < (1-\gamma)\delta_Q$, and guarantees convergence in the limit of infinite exploration, as in vanilla Q-learning. Algorithm 11 shows the full pseudo-code for this process.

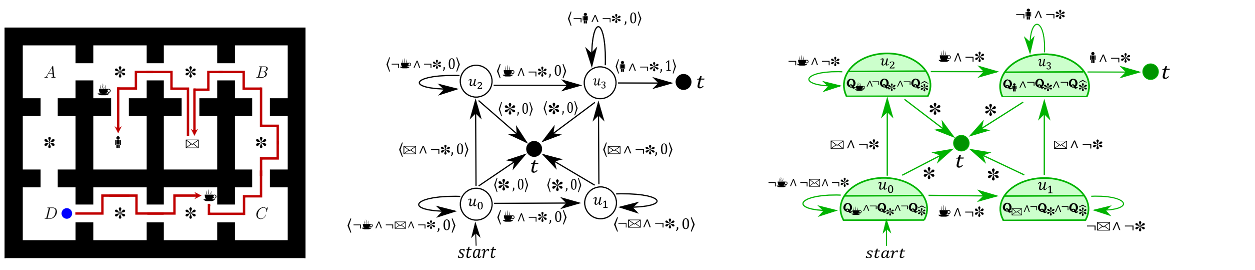
9.2 Experiments

We evaluate our approach in three domains, including a high-dimensional, continuous control task. In particular, we consider the *Office Gridworld* (the same one used in Chapter 1), the *Moving Targets* domain and the *Safety Gym* domain. We briefly describe the domains and training procedure here, and provide more hyperparameter settings in the appendix.

Office Gridworld [Icarte *et al.* 2022]: For clarity, we illustrate the environment again and an example temporal logic task in it in Figure 9.5. Tasks here are specified over 10 propositions $\mathcal{P} = \{A, B, C, D,$ ✻, ☕, ✉, 🚹, ✉$^{+}$, 🚹$^{+}\}$ and 1 constraint $\mathcal{C} = \{\widehat{✻}\}$. We learn the skill primitives $\mathcal{Q}^*_{\mathcal{G}}$ (visualised by Figure 9.6) using goal-oriented Q-learning [Nangue Tasse *et al.* 2020a], where the agent keeps track of reached goals and uses Q-learning [Watkins 1989] to update the WVF with respect to all previously seen goals at every time step.

Algorithm 11: Zero-shot and Few-shot Q-learning with skill machines

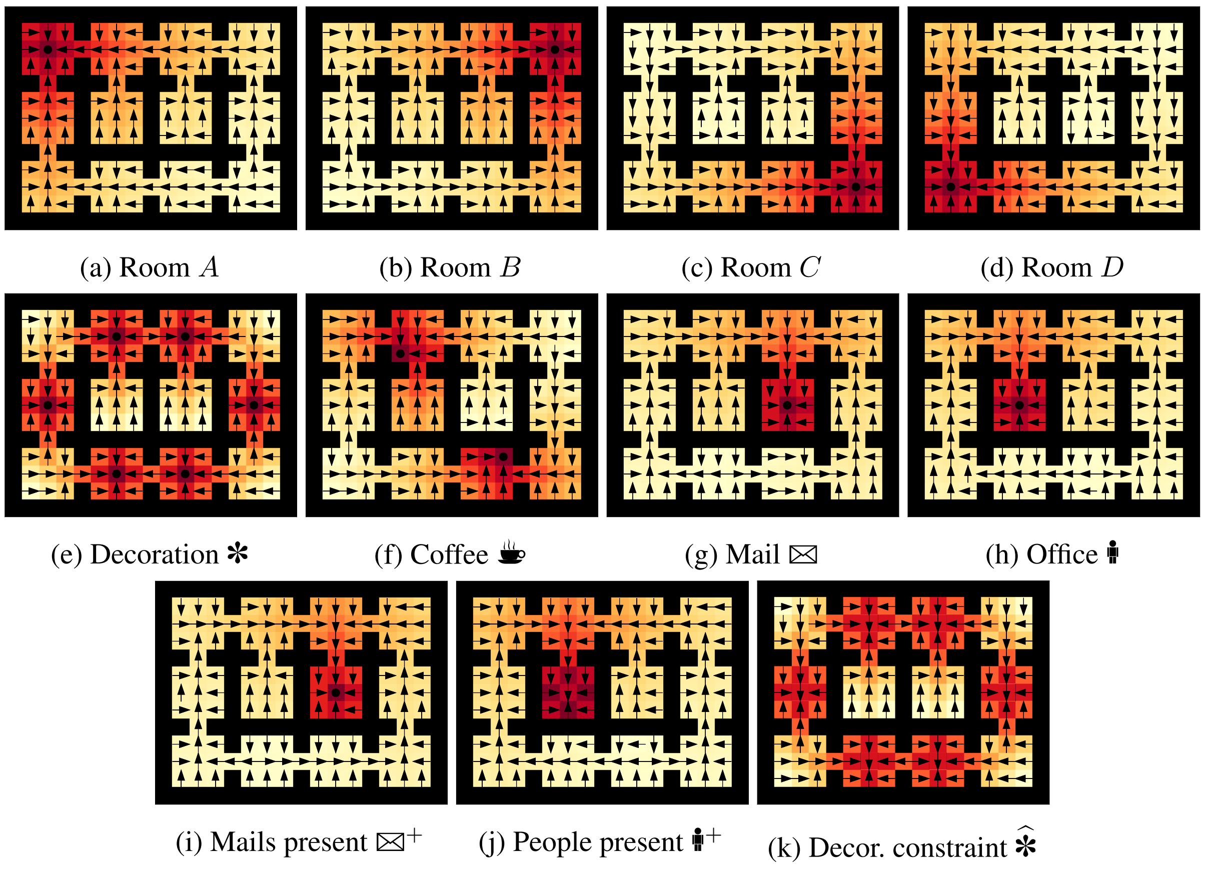
```
Input      : $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C}, \langle \mathcal{U}, u_0, \delta_u, \delta_Q \rangle, \gamma, \alpha$
Initialise : $Q(s, u, a)$
1  foreach episode do
2      Observe initial state $s \in \mathcal{S}$ and propositions $l \in 2^{\mathcal{P}}$, RM state $u \leftarrow u_0$ and constraints
        $c \leftarrow \emptyset$
3      while episode is not done do
4          if zero-shot then
5              $a \leftarrow \arg\max_a \delta_Q(u)(\langle s, c \rangle, \langle a, 0 \rangle)$
6          else
               /* Fewshot by using $\delta_Q$ in the behaviour policy */
7              $a \leftarrow$
               $\begin{cases} \arg\max_a (\max\{\gamma Q(s,u,a), (1-\gamma)\delta_Q(u)(\langle s, c \rangle, \langle a, 0 \rangle)\}) & \text{if } Bernoulli(1-\epsilon) = 1 \\ \text{sample } a \in \mathcal{A} & \text{otherwise} \end{cases}$
8          Take action $a$ and observe the next state $s'$ and true propositions $l'$
9          Get reward $r \leftarrow \delta_r(u)(s, a, s')$, true constraints $c \leftarrow c \cup ((\widehat{l} \oplus \widehat{l'}) \cap \mathcal{C})$,
10         and the next RM state $u' \leftarrow \delta_u(u, l')$
11         if $u \neq u'$ then $c \leftarrow \emptyset$
12         if $s'$ or $u'$ is terminal then
13             $Q(s, u, a) \overset{\alpha}{\leftarrow} r$
14         else
15             $Q(s, u, a) \overset{\alpha}{\leftarrow} \left[ r + \gamma \max_{a'} Q(s', u', a') \right]$
16         $s \leftarrow s'$ and $u \leftarrow u'$
```



(a) Office Gridworld (b) Reward Machine (c) Skill Machine

Figure 9.5: Illustration of (a) the office gridworld where the blue circle represents the agent; (b) the reward machine for the task "deliver coffee and mail to the office without breaking any decoration", given by the LTL specification $((F(☕ \wedge X(F(\boxtimes \wedge X(F🚹))))) \vee (F(\boxtimes \wedge X(F(☕ \wedge X(F🚹)))))) \wedge (G\neg✻)$; (c) the skill machine obtained from the reward machine which can then be used to achieve the task specification zero-shot—the red trajectory in (a). The nodes labeled t represent terminal states.

Moving Targets Domain [Nangue Tasse *et al.* 2020a]: This is a canonical object collection domain with high dimensional pixel observations ($84 \times 84 \times 3$ RGB images). We illustrate the environment in Figure 9.7. The agent here needs to pick up objects of various shapes and colours;


(a) Room A (b) Room B (c) Room C (d) Room D

(e) Decoration ✻ (f) Coffee ☕ (g) Mail ✉ (h) Office 🚹

(i) Mails present ✉$^{+}$ (j) People present 🚹$^{+}$ (k) Decor. constraint $\widehat{✻}$

Figure 9.6: The policies (arrows) and value functions (heat map) of the primitive tasks in the Office Gridworld. These are obtained by maximising over the goals of the learned WVFs.

collected objects respawn at random empty positions similarly to previous object collection domains [Barreto *et al.* 2020]. There are 3 object colours—*beige* (□), *blue* (■), *purple* (■)—and 2 object shapes—*squares* (▤), *circles* (○). The tasks here are defined over 5 propositions $\mathcal{P} = \{$□, ■, ■, ▤, ○$\}$ and 5 constraints $\mathcal{C} = \widehat{\mathcal{P}}$. We learn the corresponding skill primitives with goal-oriented Q-learning, but using deep Q-learning [Mnih *et al.* 2015] to update the WVFs.

Safety Gym Domain [Ray *et al.* 2019]: This domain (illustrated in Figure 9.1) has a continuous state space ($\mathcal{S} = \mathbb{R}^{60}$) and continuous action space ($\mathcal{A} = \mathbb{R}^2$). The agent here is a point mass that needs to navigate to various regions defined by 3 propositions ($\mathcal{P} = \{$▮, ●, ◯$\}$) corresponding to its 3 lidar sensors for the *red cylinder* (▮), the *green buttons* (●), and the *blue regions* (◯). The agent, 4 buttons and 2 blue regions are randomly placed on the plane. The cylinder is randomly placed on one of the buttons. We first learn the 4 base skill primitives corresponding to each predicate (with constraints $\mathcal{C} = \{\widehat{◯}\}$), with goal-oriented Q-learning Nangue Tasse *et al.* [2020a] but using Twin Delayed DDPG [Fujimoto *et al.* 2018] to update the WVFs.

9.2.1 Zero-shot and few-shot temporal logics

We use the Office Gridworld as a multitask domain, and evaluate how long it takes an agent to learn a policy that can solve the four tasks described in Table 9.1. The tasks are sampled uniformly at random for each episode. In all of our experiments, we compare the performance

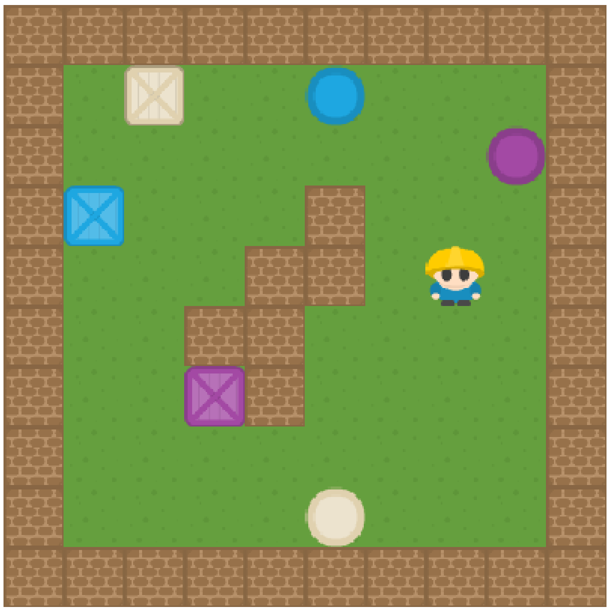
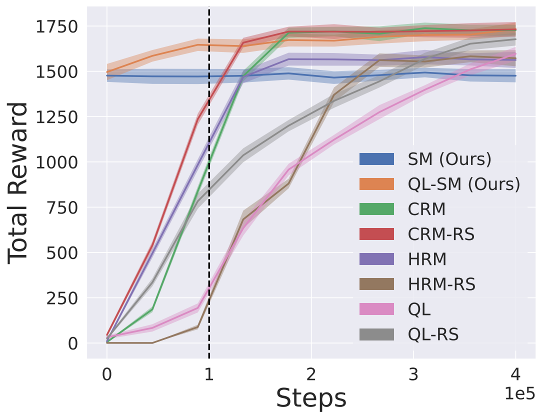


Figure 9.7: Moving Targets domain

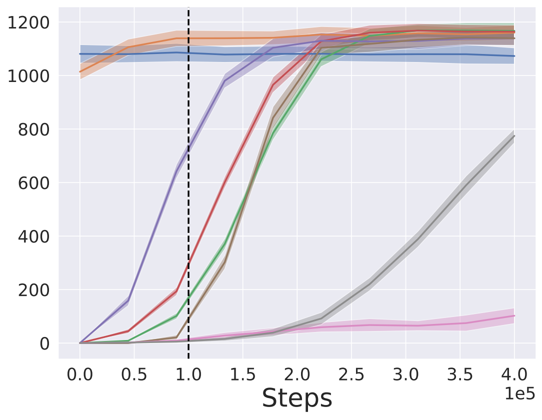
Task	Description — LTL
1	Deliver coffee to the office without breaking decorations — $(F(☕ \land X(F\,🚹))) \land (G\,\neg ✻)$
2	Patrol rooms A, B, C, and D without breaking any decoration — $(F(A \land X(F(B \land X(F(C \land X(FD))))))) \land (G\,\neg ✻)$
3	Deliver coffee and mail to the office without breaking any decoration — $((F(☕ \land X(F(\boxtimes \land X(F 🚹))))) \lor (F(\boxtimes \land X(F(☕ \land X(F 🚹)))))) \land (G \neg ✻)$
4	Deliver mail to the office until there is no mail left, then deliver coffee to office while there are people in the office, then patrol rooms A-B-C-D-A, and never break a decoration — $(F(\boxtimes \land X(F(🚹 \land X(\neg\boxtimes U(\neg\boxtimes^{+} \land \boxtimes \land X(F(☕ \land X(\neg 🚹 U(\neg 🚹^{+} \land 🚹 \land X(FA \land X(F(B \land X(F(C \land X(F(D \land X(FA))))))))))))))))))) \land (G\,\neg ✻)$

Table 9.1: Tasks in the Office Gridworld. The RMs are generated from the LTL expressions.

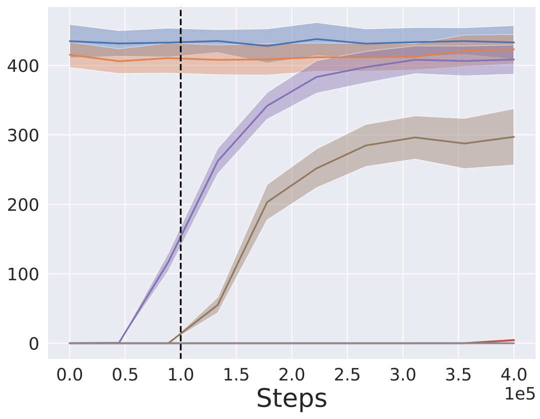
of SMs without further learning and SMs paired with Q-learning (QL-SM) with that of regular Q-learning (QL) and the following state-of-the-art RM-based baselines [Icarte *et al.* 2022]: (i) **Counterfactual RMs (CRM)**: This augments Q-learning by updating the action-value function at each state ($Q(s, u, a)$) not just with respect to the current RM transition, but also with respect to all possible RM transitions from the current environment state. This is representative of approaches that leverage the compositional structure of RMs to learn optimal policies efficiently. (ii) **Hierarchical RMs (HRM)**: The agent here uses Q-learning to learn options to achieve each RM state-transition, and an option policy to select which options to use at each RM state that are grounded in the environment states. This is representative of option-based approaches that learn hierarchically-optimal policies. (iii) **Reward-shaped variants (QL-RS, CRM-RS, HRM-RS)**: The agent here uses the values obtained from value iteration over the RMs for reward shaping, on top of the regular QL, CRM, HRM algorithms. This is representative of approaches that leverage planning over the RM to speed up learning.



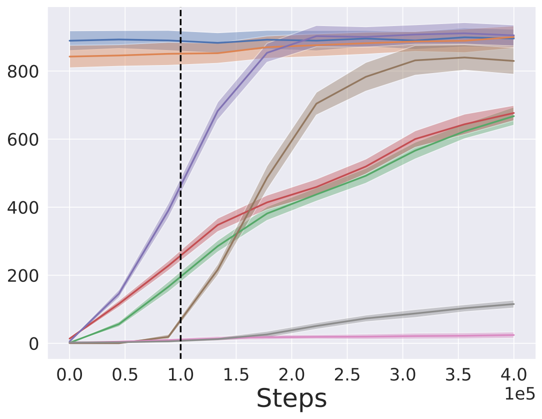
(a) Task 1



(b) Task 3



(c) Task 4

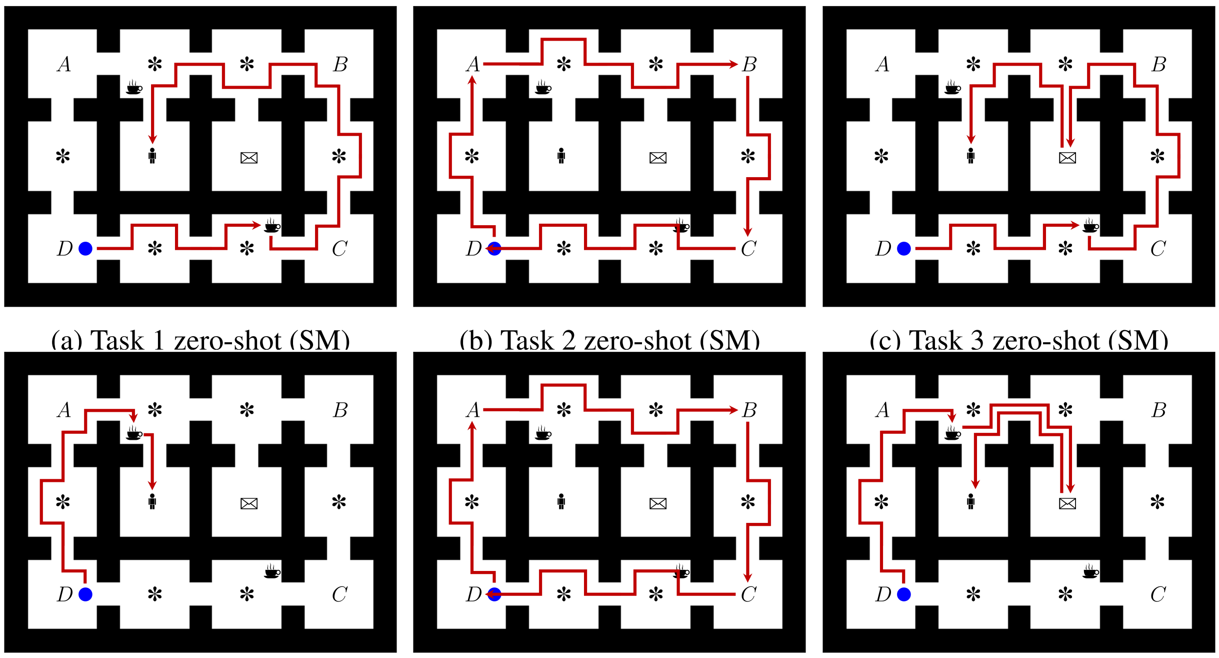


(d) Multiple tasks (1-4)

Figure 9.8: Average returns over 60 independent runs during training in the Office Gridworld. The shaded regions represent 1 standard deviation. For each training run, we evaluate the agent ϵ-greedily ($\epsilon = 0.1$) after every 1000 step and report the average total rewards obtained over each 40 consecutive evaluation. The black dotted line indicate the point at which the baselines have trained for the same number of time steps as the skill primitives pretraining.

In addition to learning all four tasks at once, we also experiment with Tasks 1, 3 and 4 in isolation. In these single-task domains, the difference between the baselines and our approach should be more pronounced, since QL, CRM and HRM now cannot leverage the shared experience across multiple tasks. Thus, the comparison between multi-task and single-task learning in this setting will evaluate the benefit of the compositionality afforded by SMs, given that the 11 skill primitives used by the SMs here are pretrained only once for 1×10^5 time steps and used for all four experiments. For fairness towards the baselines, we run each of the four experiments for 4×10^5 time steps.

The results of these four experiments are shown in Figure 9.8. Regular Q-learning struggles to learn Task 3 and completely fails to learn the hardest task (Task 4). Additionally, notice that while QL and CRM can theoretically learn the tasks optimally given infinite time, only HRM, SM, and QL-SM are able to learn hard long horizon tasks in practice (like task 4). This is



(a) Task 1 zero-shot (SM) (b) Task 2 zero-shot (SM) (c) Task 3 zero-shot (SM)

(d) Task 1 few-shot (QL-SM) (e) Task 2 few-shot (QL-SM) (f) Task 3 few-shot (QL-SM)

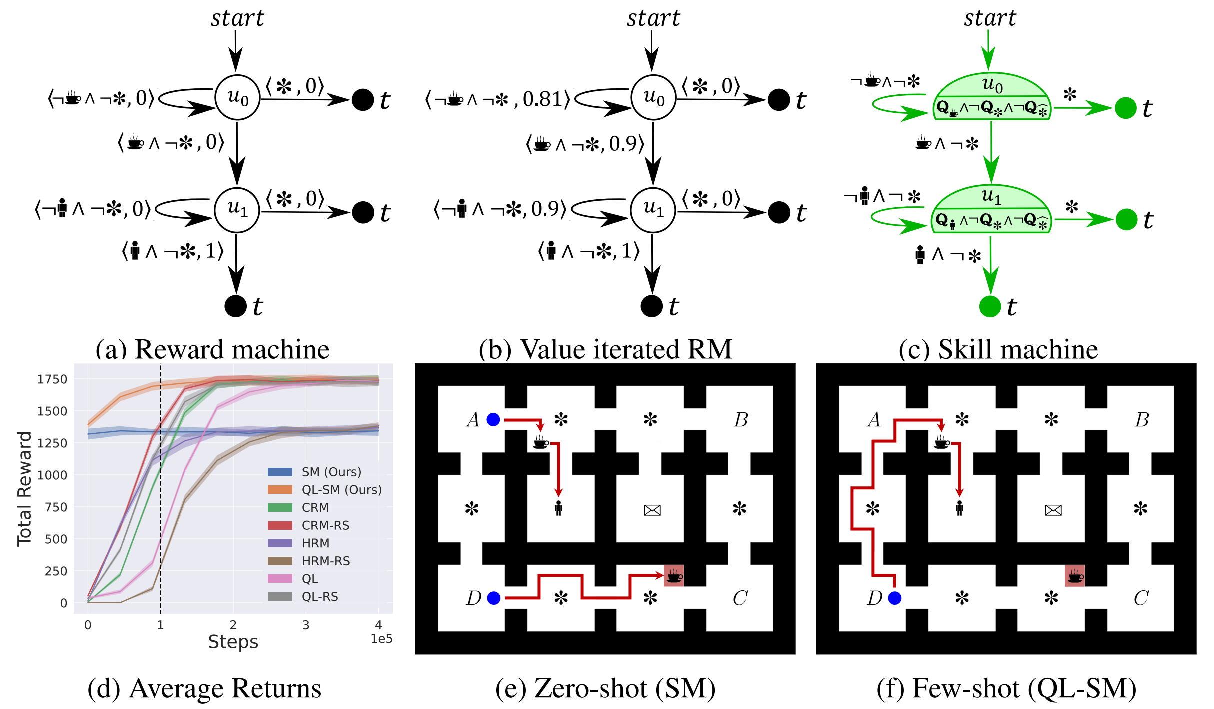
Figure 9.9: Agent trajectories for various tasks in the Office Gridworld (Table 9.1) using the skill machine without further learning (top) and with further learning (bottom).

because of the temporal composition of skills leveraged in HRM, SM, and QL-SM. In addition, the skill machines are being used to zero-shot generalise to the office tasks using skill primitives. Thus using the skill machines alone (SM in Figure 9.8) may provide sub-optimal performance compared to the task-specific agents, since the SMs have not been trained to optimality and are not specialised to the domain. Even under these conditions, we observe that SMs perform near-optimally in terms of final performance, and due to the amortised nature of learning the WVF will achieve its final rewards from the first epoch.

It is apparent from the results shown in Figure 9.8 that SMs paired with Q-learning (QL-SM) achieve the best performance when the zero-shot performance is not already optimal. This can also be observed from the trajectories of the agent with and without few-shot learning (Figure 9.9). Additionally, SMs with Q-learning always begin with a significantly higher reward and converge on their final performance faster than all baselines. The speed of learning is due to the compositionality of the skill primitives with SMs, and the high final performance is due to the generality of the learned primitives being paired with the domain-specific Q-learner. In sum, skill machines provide fast composition of skills and achieve optimal performance compared to all benchmarks when paired with a learning algorithm.

Finally, we run 2 experiments to demonstrate the performance of our zero-shot and few-shot approach when the global reachability assumption does not hold.

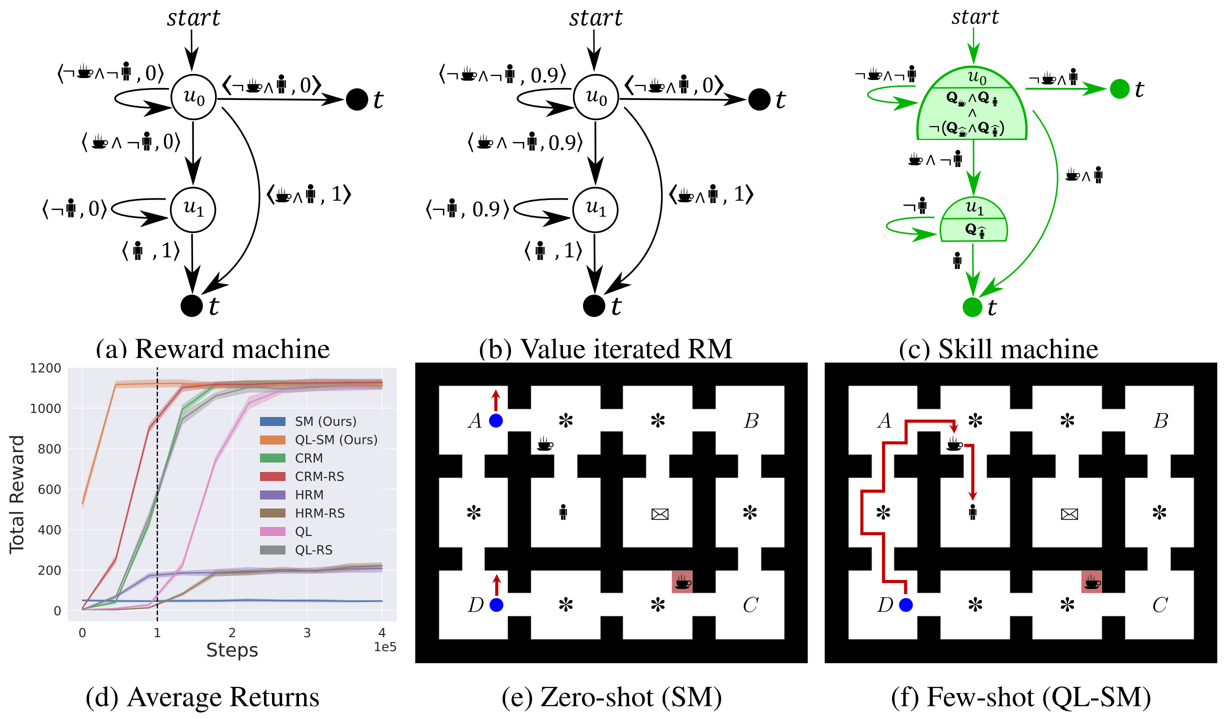
1. **When the reachability assumption is not satisfied in some initial states:** In the first experiment (Figure 9.10), the agent needs to solve task 1 of Table 9.1 ($((F(☕ \wedge X(F\ 🚹))) \wedge (G\ \neg ✻))$, but we modify the environment such that one of the coffee locations is absorbing (a sink environment state). This breaks the global reachability assumption since the agent



(a) Reward machine (b) Value iterated RM (c) Skill machine

(d) Average Returns (e) Zero-shot (SM) (f) Few-shot (QL-SM)

Figure 9.10: Results for the task with LTL specification (*F*(☕ ∧ *X*(*F* 🚹))) ∧ (*G* ¬✻) when the global reachability assumption does not hold.



(a) Reward machine (b) Value iterated RM (c) Skill machine

(d) Average Returns (e) Zero-shot (SM) (f) Few-shot (QL-SM)

Figure 9.11: Results for the task with LTL specification (*F* 🚹) ∧ (¬🚹 *U* ☕) where the global reachability assumption is not satisfied.

can no longer reach the office location after it reaches the absorbing coffee location. As a result, we observe that the zero-shot agent (SM) is even more sub-optimal than before because it cannot satisfy the task when it starts at locations that are closer to the absorbing coffee location. However, we can observe that the few-shot agent (QL-SM) is still able to learn the optimal policy, starting with the same performance as the zero-shot agent. Note that the hierarchical agent (HRM) also converges to the same performance as our zero-shot agent because it also tries to reach the nearest coffee location.

2. **When the reachability assumption is not satisfied in all initial states:** In the second experiment (Figure 9.11), the agent needs to solve the task with LTL specification $(F$ 🚹$) \wedge (\neg$🚹 U ☕$)$—the environment is still modified such that one of the coffee locations is absorbing. Here, the Boolean expression ☕ $\wedge$ 🚹 is not satisfiable since there is no state where both propositions (☕ and 🚹) are true. Hence, this can be seen as the worst-case scenario for our approach (without outright making the task unsatisfiable), since ☕ $\wedge$ 🚹 is the Boolean expression greedily selected in the starting RM state. As a result, our zero-shot agent completely fails to solve this task. Even in this case, we can observe that the few-shot agent is still able to learn the optimal policy.

9.2.2 Zero-shot transfer with function approximation

Task	Description — LTL
1	Pick up any object. Repeat this forever. — F (○ $\vee$ ▤)
2	Pick up blue then purple objects, then objects that are neither blue nor purple. Repeat this forever. — $F($🟦 $\wedge X(F($🟥 $\wedge X(F(($○ $\vee$ ▤$) \wedge \neg($🟦 $\vee$ 🟥$))))))$
3	Pick up blue objects or squares, but never blue squares. Repeat this forever. — $(F($🟦 $\vee$ ▤$)) \wedge (G\, \neg($🟦 $\wedge$ ▤$))$
4	Pick up non-square blue objects, then non-blue squares in that order. Repeat this forever. — $F((\neg$▤$\wedge$🟦$) \wedge X(F($▤$\wedge \neg$🟦$)))$

Table 9.2: Tasks in the Moving Targets domain. To repeat forever, the terminal states of the RMs generated from LTL are removed, and transitions to them are looped back to the start state.

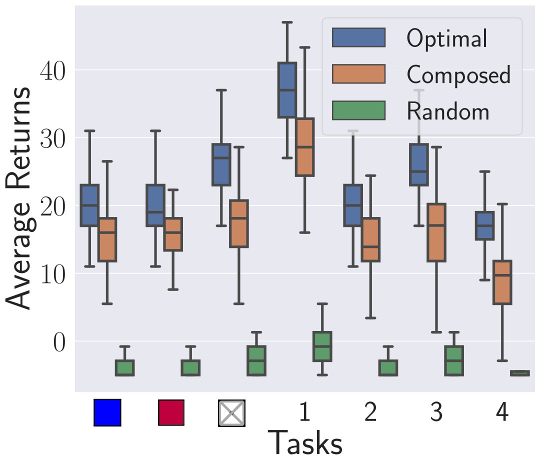
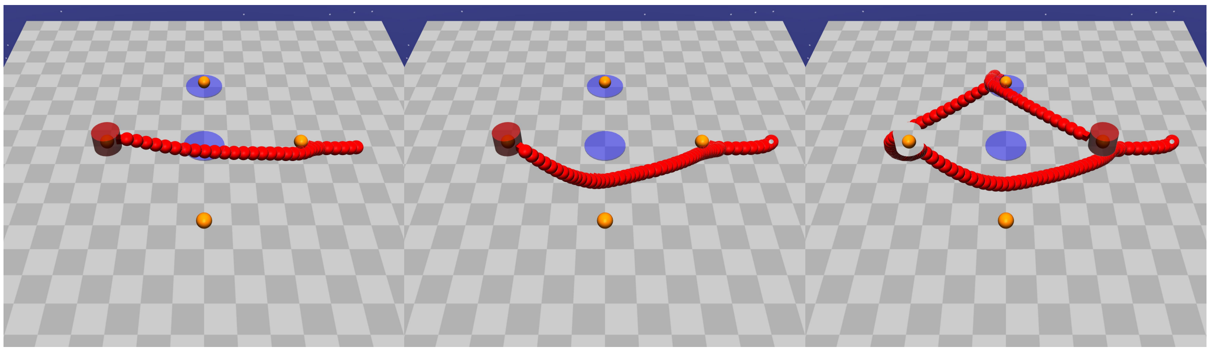


Figure 9.12: Average returns over 100 runs for tasks in Table 9.2. The agent and object positions are randomised and objects respawn randomly when collected.

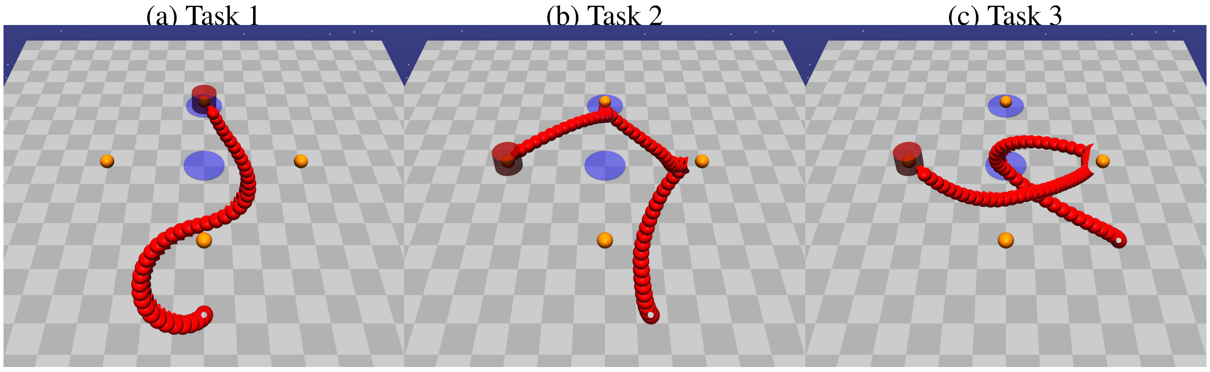
We now demonstrate our temporal logic composition approach in the Moving Targets domain where function approximation is required. Figure 9.12 shows the average returns of the optimal policies and SM policies for the four tasks described in Table 9.2 with a maximum of 50 steps per episode. Our results show that even when using function approximation with sub-optimal skill primitives, the zero-shot policies obtained from skill machines are very close to optimal on average. We also observe that for very challenging tasks like Tasks 3 and 4 (where the agent must satisfy difficult temporal constraints), the compounding effect of the sub-optimal policies sometimes leads to failures. Finally, we provide a qualitative demonstration of our method's applicability to continuous control tasks using Safety Gym, a benchmark domain used

Task	Description — LTL
1	Navigate to a button and then to a cylinder. — (F(🟢 ∧ X(F 🔴)))
2	Navigate to a button and then to a cylinder while never entering blue regions — (F(🟢 ∧ X(F 🔴))) ∧ (G ¬🔵)
3	Navigate to a button, then to a cylinder without entering blue regions, then to a button inside a blue region, and finally to a cylinder again. — F(🟢 ∧ X(F ((🔴 ∧ ¬🔵) ∧ X(F((🟢 ∧ 🔵) ∧ X(F🔵))))))
4	Navigate to a button and then to a cylinder in a blue region. — (F(🟢 ∧ X(F 🔴 ∧ 🔵)))
5	Navigate to a cylinder, then to a button in a blue region, and finally to a cylinder again. — (F(🔴 ∧ X(F ((🟢 ∧ 🔵) ∧ X(🔴)))))
6	Navigate to a blue region, then to a button with a cylinder, and finally to a cylinder while avoiding blue regions. — (F(🔵 ∧ X(F((🟢 ∧ 🔴) ∧ X((F 🔴) ∧ (G¬🔵))))))

Table 9.3: Tasks in the Safety Gym domains. The RMs are generated from the LTL expressions.



(a) Task 1 (b) Task 2 (c) Task 3



(d) Task 4 (e) Task 5 (f) Task 6

Figure 9.13: Visualisations of the trajectories obtained by following the zero-shot composed policies from the skill machine for tasks in Table 9.3.

for developing safe RL methods [Ray *et al.* 2019]. We define a set of increasingly complex tasks (Table 9.3) and visualise the resulting trajectories after composing the agent's learned primitive skills. Figure 9.1 illustrates the trajectory that satisfies the task requiring the agent to navigate to a blue region, then to a red cylinder, and finally to another red cylinder while avoiding blue regions.

9.3 Related works

Prior works built on successor features (SF) have shown some promise in solving temporal tasks by using linear preferences over features [Barreto *et al.* 2020], while Alver and Precup [2022b] show that an SF basis can be learned that is sufficient to span the space of such linear tasks. By contrast, our framework allows for both spatial composition (including operators such as negation that others do not support) and temporal composition such as LTL.

A popular way of achieving temporal composition is through the options framework [Sutton *et al.* 1999]. Here, high-level skills are first discovered and then executed sequentially to solve a task [Konidaris and Barto 2009]. Barreto *et al.* [2019] leverage the SF and options framework and learn how to linearly combine skills, chaining them sequentially to solve temporal tasks. However, these approaches offer a relatively simple form of temporal composition. By contrast, we are able to solve tasks expressed through regular languages zero-shot, while providing soundness guarantees.

Approaches to defining tasks using human-readable logic operators also exist. Li *et al.* [2017] and Littman *et al.* [2017] specify tasks using LTL, which is then used to generate a reward signal for an RL agent. Camacho *et al.* [2019] perform reward shaping given LTL specifications, while Jothimurugan *et al.* [2019] develop a formal language that encodes tasks as sequences, conjunctions and disjunctions of subtasks. This is then used to obtain a shaped reward function that can be used for learning. These approaches focus on how to improve learning given such specifications, but we show how an explicitly compositional agent can immediately solve such tasks using WVFs without further learning.

9.4 Conclusion

We proposed skill machines—finite state machines that can be learned from reward machines—that allow agents to solve extremely complex tasks involving temporal and spatial composition. We demonstrated how skills can be learned and encoded in a specific form of goal-oriented value function that, when combined with the learned skill machines, are sufficient for solving subsequent tasks without further learning. Our approach guarantees that the resulting policy adheres to the logical task specification, which provides assurances of safety and verifiability to the agent's decision making, important characteristics that are necessary if we are to ever deploy RL agents in the real world. While the resulting behaviour is provably satisficing, empirical results demonstrate that the agent's performance is near optimal; further fine-tuning can be performed should optimality be required, which greatly improves the sample efficiency. We see this approach as a step towards truly generally intelligent agents, capable of immediately solving human-specifiable tasks in the real world with no further learning.

Chapter 10

Safe Sim-to-Real

This chapter is based on the published work
"Facilitating Safe Sim-to-Real through Simulator Abstraction and Zero-shot Task Composition"
[Love et al. 2022], jointly lead with Tamlin Love and Devon Jarvis, in collaboration with
Branden Ingram, Steven James, and Benjamin Rosman.

While we have shown how to construct agents with all the FIRe desiderata (, ,), there is still one glaring issue to be addressed. As in much of RL [Arumugam *et al.* 2019; Zhao *et al.* 2020; Osiński *et al.* 2020], all of our experiments here occurred in non physical domains, such as games and physics simulators.

Since exploration is a key component of RL, it is often infeasible and unsafe to train an agent in anything but a non-physical system, where the cost and damage from exploration can be mitigated or avoided. Thus, deploying an RL agent on a physical system remains an open challenge, made worse by the "reality gap". This describes the fact that no simulator is perfectly able to replicate reality. The consequence is that an agent which performs well in simulation is not guaranteed to perform comparably in the real-world.

Due to the discrepancy between simulators and the real-world, the reality gap is an example of a *domain adaptation* [Wang and Deng 2018; Jiang *et al.* 2021] problem in which an agent must train in a source domain but generalise to performing well in another target domain [Matas *et al.* 2018]. The degree of discrepancy between the domains depends on the fidelity of the simulator itself, as well as the complexity of the task the agent is being trained to perform. Thus, the goal of domain adaptation is for the agent to learn robust and broadly useful features and policies which are common between the source and target domains. Importantly, this does not necessitate that the source domain be a perfect replica of the target domain. Consequentially, it does not require a simulator as the source domain to be any higher fidelity than is useful for learning a policy which transfers between domains. This is a consideration which we aim to leverage with the goal of providing a safe robot learning framework that also accommodates high sample complexity learning algorithms and compositional generalisation of skills.

We introduce a framework which can facilitate a combinatorial explosion of skills within a robotics domain while utilising physical system training sparingly, and only in a safe manner. Our framework is depicted in Figure 10.2 and our design philosophy can be summarised as follows: the more difficult the task, the less physical system training should be done. To avoid

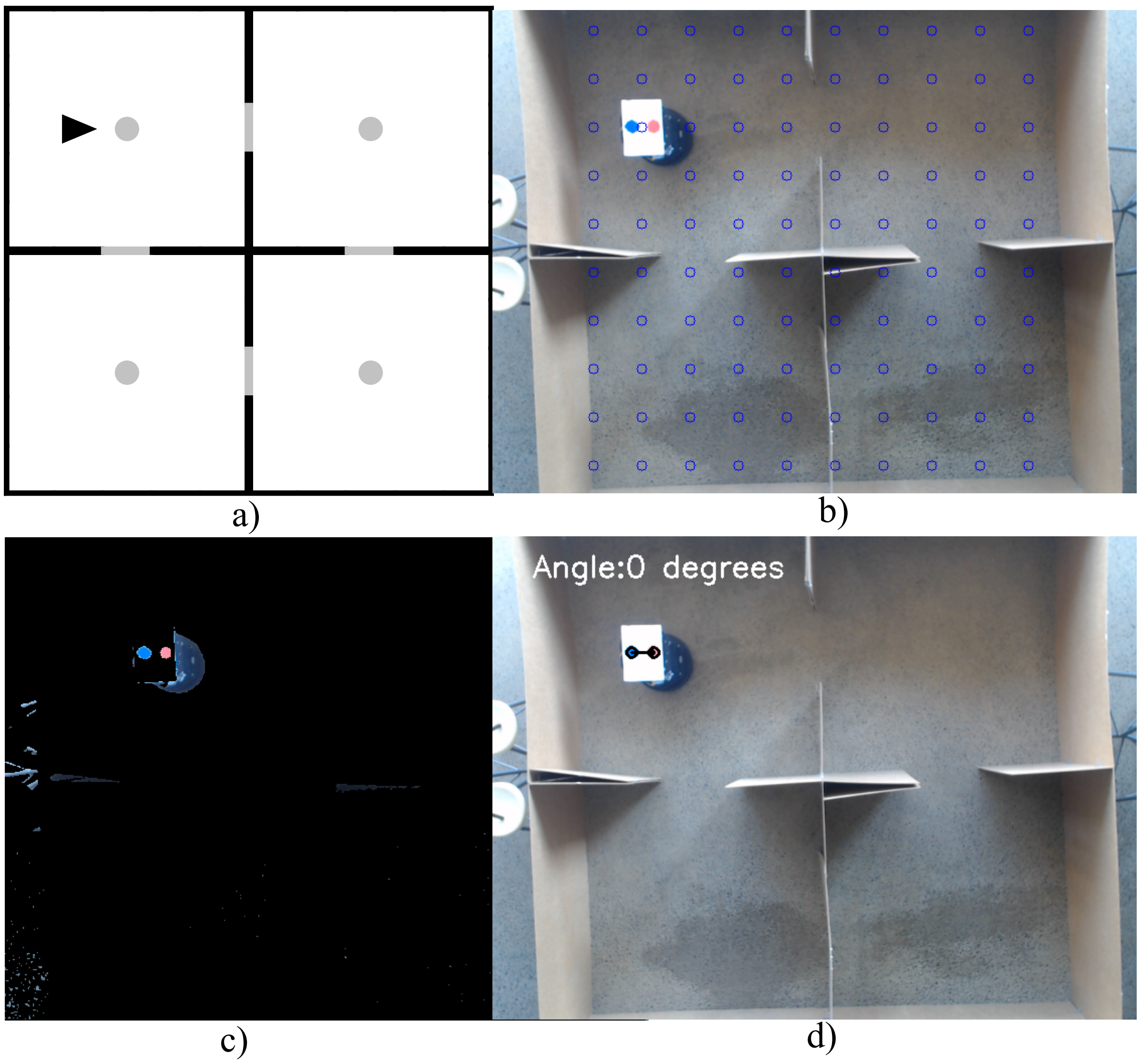
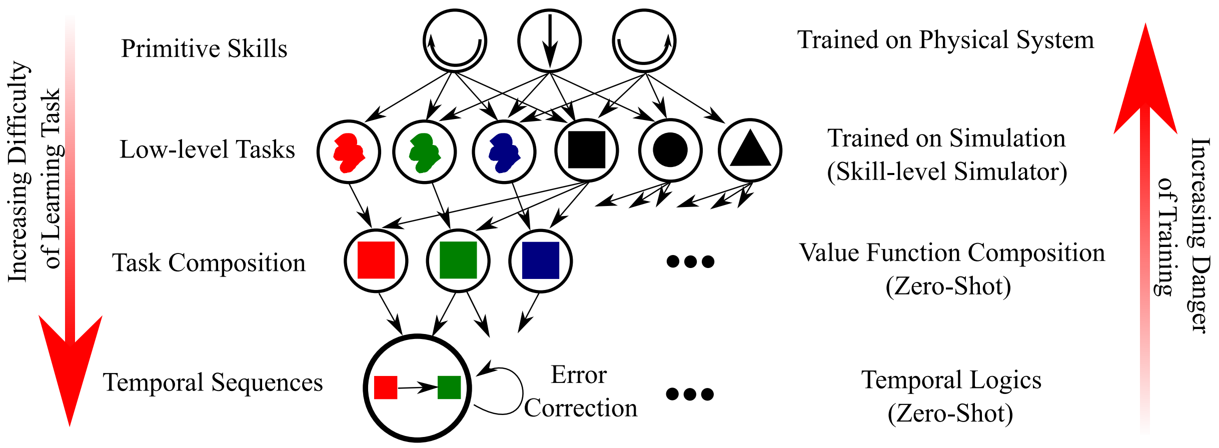


Figure 10.1: Visualisation and real world correspondence of the low fidelity simulation of the real Four Rooms domain. a) The agent in the abstract skill-level simulator. b) Centres of each grid-state displayed on top of the overhead camera feed. c) Segmentation mask for robot and colour tags identification in the overhead camera feed. d) Localisation of the robot with position and angle in the real world Four Rooms domain.

training in a complex continuous space simulator at the level of actuators, we instead use an abstract simulator at the level of primitive skills. Learning the primitive skills is the only point at which we train on a physical system. This provides us with an abstraction, allowing us to thereafter operate in a discrete, higher-level of simulation where we train an agent to perform a set of low-level tasks (one step higher in abstraction than primitive skills). This is easy to do in simulation, but importantly due to the abstraction of the simulator there will be less discrepancy when transferring these tasks to the real-world. This is because the abstract simulator would only need to match the real-world in dynamics relative to primitive skills, rather than the far more intricate level of actuator dynamics.



Increasing Difficulty of Learning Task
Primitive Skills
Low-level Tasks
Task Composition
Temporal Sequences
Error Correction
Trained on Physical System
Trained on Simulation (Skill-level Simulator)
Value Function Composition (Zero-Shot)
Temporal Logics (Zero-Shot)
Increasing Danger of Training

Figure 10.2: Hierarchical abstraction of simulation which aims to leverage the benefits of training an agent in the real world while containing the risk of damage. The more dangerous a task is to train the higher the level of abstraction in training, containing the risk of real world training to the most basic skills. Similarly, the more possible tasks there are at a level of abstraction the less time is needed for individual training, leading to zero-shot skill composition and temporal logic.

Having learned the low-level tasks, we are then able to easily learn complex behaviours in a simple simulator, leveraging recent advances in RL. This allows us to draw on techniques to generalise to combinatorially many higher-level tasks and tasks requiring temporal logic in a zero-shot manner. Thus, no further training is required for these more difficult tasks as long as they can be composed from a set of low-level tasks in the same domain. The effect of this is that for complex tasks, we can avoid training on a physical system, and instead train in simulation in a way that allows zero-shot generalisation onto the hardware. We describe our entire framework in more detail in Section 10.1.

10.1 Leveraging simulator abstraction and skill machines

Our approach to mitigating the reality gap can be summarised in three steps, and shown graphically in Figure 10.2. These steps involve firstly abstracting the simulator into what we refer to as a "skill-level simulator", from which low-level tasks are learned. Secondly, zero-shot value function composition is utilised to compose tasks in order to solve more complex problems. Thirdly, we utilise temporal logics in order to generate task sequences.

10.1.1 Skill-level simulator

To avoid the difficulties associated with using a high fidelity simulator for training we instead work with an abstract simulator which captures only the necessary structure in the source domain which is common with the target domain. This is achieved by training a set of primitive skills on a physical system. It is key that these primitive skills be small and easy enough to learn that training is safe and quick, avoiding the usual pitfalls of training on a physical system. Additionally, the use of early training on the physical system has been used to make downstream sim-to-real easier in prior work [Golemo *et al.* 2018; Jeong *et al.* 2019; Hanna and Stone 2017; Desai *et al.* 2020]. This abstracts the simulator away from the level of actuators and instead simulates the environment in terms of discrete primitive skills. By learning in the skill-level simulator we are then able to train policies for low-level tasks (sequence of skills). Figure 10.3 shows the returns obtained during training of the WVFs and VFs of said tasks. Similarly to Chapter 5, we observe that learning WVFs has the additional benefit of faster training than regular VFs—since the WVF learns how to achieve all goals leading to better goal-directed exploration during learning.

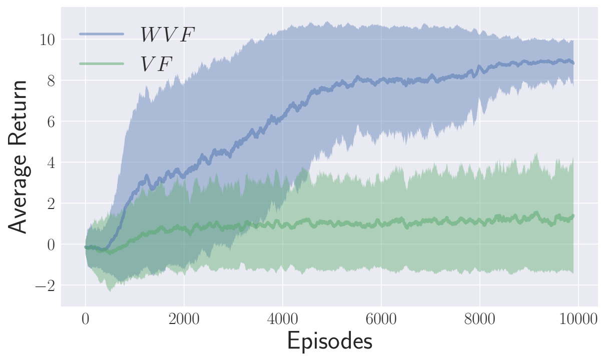


Figure 10.3: Average returns per episode obtained when learning WVFs and regular VFs in the Four Rooms simulation. The shaded regions represent 1 standard error over the returns obtained when training the following 10 tasks: Navigate to the ”bottom-left room”, ”bottom-right room”, ”top-right room”, ”top-left room”, ”front of left door”, ”front of top door”, ”front of right door” and ”front of bottom door”.

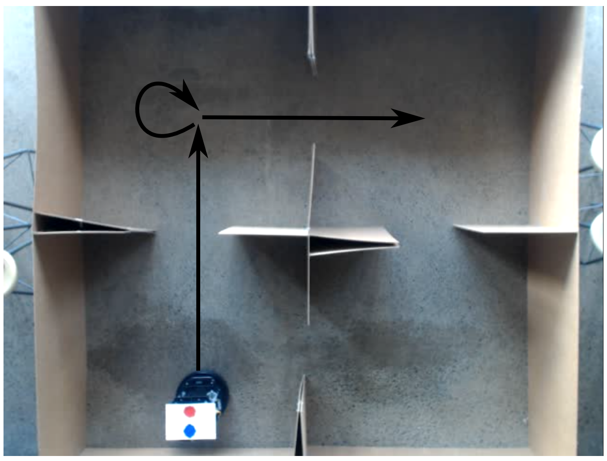
10.1.2 Value function composition using WVFs

Our second step is to leverage the results from Part II of zero-shot value function composition, described in Chapter 6, which provides super-exponential growth in the number of possible tasks an agent can perform just from learning low-level tasks in the skill-level simulator. This alone speeds up learning significantly in our framework. However, due to the discrete nature of the skill-level simulator and resulting value functions it is necessary to perform error correction when transferring to the continuous real-world. Error correction would then be performed after every primitive skill which is time-consuming and a potential drawback of our framework. Thus, we leverage another level of abstraction introduced in previous chapters: temporal logic composition (Chapter 9).

10.1.3 Temporal logics using skill machines

Temporal logic of task composition defines a problem as a sequence of steps to be completed by modelling the sequence with an RM. Thus the task of making a cup of coffee would be split into adding coffee granules to a cup, adding sugar, pouring hot water, etc. This is beneficial as we can use the RM to perform error correction only at the level of transitions between RM states as opposed to after transitions in the skill-level states (after each skill is performed). Thus, we have multiple levels of abstraction, each serving a purpose in minimising physical risks, speeding up learning and reducing the amount of error correction needed when switching from sim-to-real respectively. The aim of this work is to provide a helpful framework for robot sim-to-real generalisation by applying hierarchical and compositional techniques from RL to split a learning problem into multiple levels of abstraction. The benefit is that by training a very few easy and safe primitive skills on a physical system the framework can facilitate a combinatorial explosion of skills within a domain, culminating in zero-shot generalisation to temporal logic sequences of complex tasks.

10.2 Experiments


a) Task 1 Trajectory

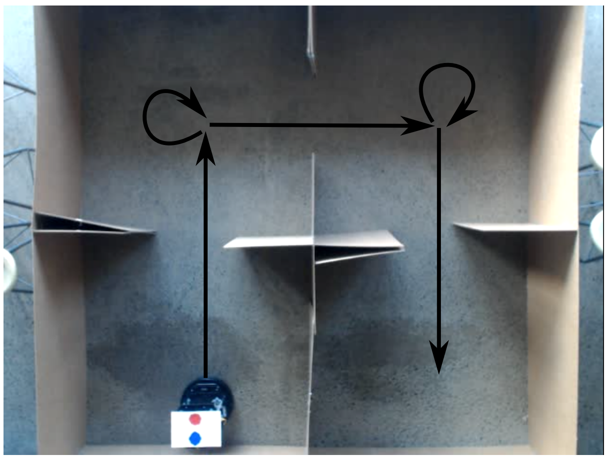

b) Task 2 Trajectory

Figure 10.4: Desired trajectories for the experiment 1 (shown in a) and experiment 2 (shown in b). Note that for experiment 2 the door between the bottom left (source) and bottom right (target) is closed forcing the robot to take the longer path through the rooms. The RM and temporal logics allows for zero-shot generalisation to such changes in the environment, unlike naive Q-learning.

In order to test the benefits of our approach, we conducted a number of experiments in a real-world "Four Rooms" domain on a Kobuki TurtleBot2. The "Four Rooms" domain consists of four square rooms, connected by doors which can be open or closed. In the skill-level simulation, the domain is represented by a discrete grid of cells connected by a move-forward skill (each room being 5$\times$5 cells), with goal states in the centres of rooms. Three primitive actions (move forward one cell, turn left $90°$, turn right $90°$) were hand-coded on the robot. The policy of the robot was trained using the following three levels of complexity: naive Q-learning, value function composition (WVFs) from naive Q-learning with low-level tasks and lastly with temporal logics (skill machines).

The error-correction procedure was designed to be a simple and naïve approach to match the robot's state to that of the simulated agent. We suspended a webcam above the domain and placed colour markers on the robot to allow for its location and orientation to be tracked (shown in Figure 10.1. A grid of coordinates was superimposed on the webcam feed to map the simulated gridworld to the real world. The error-correcting procedure attempts to move the robot to the corresponding discretised position of the simulated agent following the same policy. As such, an error-correcting step may consist of a rotation or forward movement towards the coordinates of the grid cell.

	Successful Runs	**Total Time (s)**		**C Time (%)**		**Distance from Goal (pixels)**		**Number of C**	
		Average	STD	Average	STD	Average	STD	Average	STD
Transferred Policy, No C	3/5	12.80	0.01	0	0	38.78	13.49	0	0
Transferred Policy + C	5/5	199.83	34.98	93.50	1.44	5.59	2.76	175	32.81
VFC Policy + C	5/5	205.72	29.20	93.53	0.82	4.35	2.98	179.4	28.32
Temporal Logics + C	3/3	49.55	2.27	73.98	1.13	10.21	3.73	34.33	2.05

Table 10.1: The results for the first set of experiments (goal state: top-right, all doors open). The abbreviation C here refers to Corrections.

In the first set of experiments, we consider the problem of moving from the bottom-left room to the top-right. For naive Q-learning this is learned directly as part of the broader training of navigation the entire domain, where the robot is trained to move from any source and to any target room where each combination is treated as its own task. For the value function composition the low-level tasks involve the agent learning to move to any two target rooms. To reach one specific target room, the intersection between two low-level task rooms is used. Finally, for the temporal logics instead of learning to move between a sequence of rooms to reach the target, the transition between rooms is modelled with an RM and so the robot is only required to learn how to move between adjacent rooms with the composed value functions. For example, the RM used to perform task 1 is shown in Figure 10.5, with desired trajectories shown in Figure 10.4. Thus, the full set of methods we compare are: a simulated naive Q-learning policy transferred directly to the robot (no error-correction), the same policy with error-correction after every discrete action, a composed value-function policy with error-correction (which we call the VFC Policy) and a policy obtained using temporal logics with error correction after each RM state (so that error correction only happens at doorways and room centres). Each method is evaluated using the metrics of "successful runs" (the number of successful runs, where a run is considered failed if the robot crashes into a wall), "total time" (the total time in seconds to perform a run), "correction time" (the percentage of time spent correcting the robot's position), "distance from goal" (the distance, in pixels on the webcam feed, from the final goal at the end of the run), and "number of corrections" (the number of error-correction steps performed in a run). Results are averaged over the number of successful runs and detailed in Table 10.1.

While the directly transferred policy executes quickly, it has a poor success rate, and even the runs that do succeed end up relatively far from the centre of the room. Applying error-correction after every step of the policy greatly improves the success rate and final position accuracy, at the cost of time. The benefits of the composed value functions is not seen on the real world

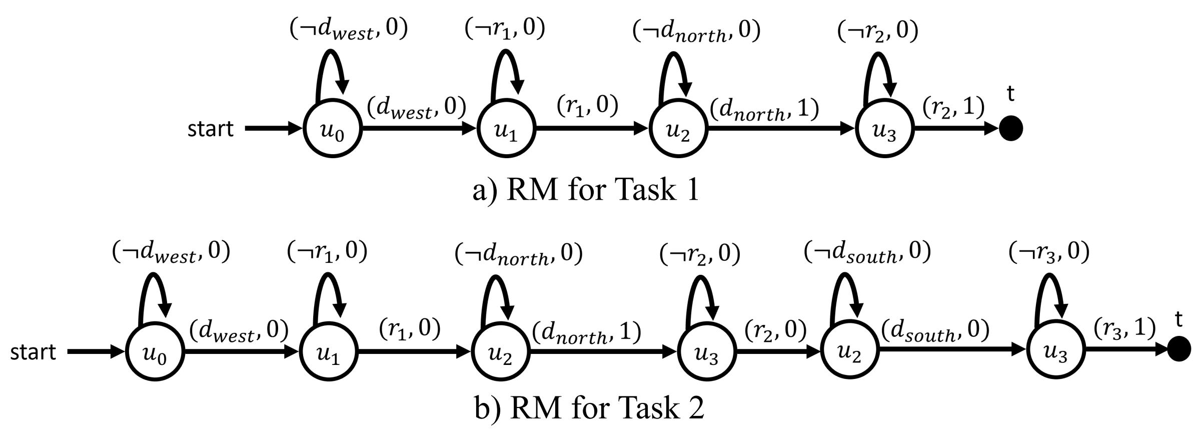


Figure 10.5: Example of an RM used to complete task 2 in the Four Rooms domains. The agent will navigate from the bottom left to top left and finally to the top right. d_{west}, d_{north}, d_{south}, r_1, r_2 and r_3 are respectively propositions that are true when the agent is in front of the west door, in front of the north door, in front of the south door, in the middle of the top-left room, in the middle of the top-right room in the middle of the bottom-right room. The U_i symbols represent state which track the sequence of propositions that are used and correspond to value functions which are relevant to the state.

generalisation but significantly speeds up the training time of robots using our framework. Finally, the temporal logics method strikes a balance between speed and safety, achieving a comparable success rate to the error-correction after ever step but using less error corrections and time to complete the task. In the second set of experiments, the task is to move from the bottom-left room to the bottom-right room. However, the door between the two rooms is closed, forcing the robot to take a less-direct path. With a greater distance to traverse and more doorways to pass through, this represents a harder task for the robot. The same methods are evaluated using the same metrics as before, tabulated in Table 10.2. From these results we may draw the same conclusions. Naive Q-learning without error correction is not a reliable procedure for completing the task, and does significantly worse on this more challenging second task which requires a longer sequence of accurate decisions. The value function composition method (we omit naive Q-learning with error correction since it is equivalent at test time to the value function composition but trains significantly slower) is safe and reliably completes the task but is very slow. Similarly to the first task, temporal logics with state-machine error corrections is both reliable and efficient at completing the task, demonstrating the benefits of the final level of abstraction.

	Successful Runs	**Total Time (s)**		**C Time (%)**		**Distance from Goal (pixels)**		**Number of C**	
		Average	STD	Average	STD	Average	STD	Average	STD
Transferred Policy, No C	0/3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
VFC Policy + C	3/3	304.03	6.13	93.57	0.14	3.46	1.42	265.67	6.18
Temporal Logics + C	4/4	154.04	29.65	86.26	2.66	6.17	1.71	125.25	27.84

Table 10.2: The results for the second set of experiments (goal state: bottom-right, bottom door closed). The abbreviation C here refers to Corrections.

10.3 Related works

Due to the potential benefits, addressing the reality gap has received a significant amount of attention in recent years from multiple fields, such as robotics and computer vision. The problem of transferring control policies from simulation to the real world can be viewed as an instance of domain adaptation, where a model trained in a source domain is transferred to a new target domain. One of the key assumptions behind these domain adaption methods is that the different domains share common characteristics such that representations and behaviours learned in one will prove useful for the other [Peng *et al.* 2018]. However, most approaches tend to either increase the sample complexity of training or rely on a fine-tuned adaptation to a simulator which may not generalise in its own right. For the first set of approaches which increase sample complexity, the most promising are domain randomisation [Tobin *et al.* 2017] and dynamics randomisation [Peng *et al.* 2018]. This entails adding noise to some aspect of the simulator such that a model cannot learn to exploit idiosyncrasies in the simulator to artificially improve performance. Additionally, if by adding noise the simulator is randomly pushed towards being more reflective of the real world then the model will generalise significantly better. Thus, these methods rely on the real-world dynamics being within the sample space of the simulator dynamics. In general the noise is often added to the low-level dynamics of the simulator, but can also be added to the hyper-parameters of the simulator. Our approach avoids utilising noise due to the nature of our abstraction, where we can be certain skill-level dynamics of the simulator overlaps with that of the real world.

Other methods have aimed to learn a network which adapts a simulator policy to a real-world domain [Ganin *et al.* 2016; Golemo *et al.* 2018] and generally aim to push the model to learn invariant features that are common between source and target domain [Taigman *et al.* 2016; Rusu *et al.* 2017; Tzeng *et al.* 2014]. This latter group of methods includes learning to adjust a simulator in a manner which mitigates the discrepancies between the simulator and real-world domains. For example the Neural-Augmented Simulation (NAS) trains a recurrent neural network to predict the discrepancies between the simulator and reality and then uses the network to augment and adapt the simulator to have more realistic dynamics [Golemo *et al.* 2018]. This approach improves upon the even more tailored approaches of learning a forward model of the real-world dynamics [Punjani and Abbeel 2015; Fu *et al.* 2016; Mordatch *et al.* 2016]. Unfortunately, due to the accumulation of errors over time, unless any of these learned-simulator based approaches are nearly perfect they will be limited to short time horizons, which is prohibitive for training RL agents. Thus, there is still a need to explore new ideas to improving sim-to-real generalisation. One recent idea of interest is the use of a small amount of real-world data at the beginning of a training pipeline which serves to improve the simulator's accuracy [Golemo *et al.* 2018; Jeong *et al.* 2019]. Specifically, this small amount of real-world data is used to train the simulator error prediction or an inverse-RL model. In this work we begin similarly with a small amount of real-world training. However, to the best of our knowledge, addressing the reality gap by simulating the environment at a more abstract level and employing hierarchical RL has not been explored. Unlike these approaches we leverage the fact that it is advantageous to utilise an abstracted simulator using skill-level dynamics rather than actuator-level dynamics.

10.4 Conclusion

To date the application of RL in robotics has been limited due to the danger of training real world systems. This has meant training has traditionally required the use of a simulator to avoid the risk of damaging expensive equipment. However, transferring a policy learned in simulation to the real world is not trivial due to the reality gap. We demonstrate a novel framework for solving this problem by training a robot in an abstracted simulator we dubbed the “skill-level simulator” and leveraging value function composition with temporal logics from prior work in RL. This framework allowed a robot to learn a set of primitive actions which can then be used to learn low-level tasks in the skill-level simulator. These low-level task value functions can then be composed in a combinatorial and sequential fashion without the need for additional training. This approach allowed us to solve complex problems while only needing to train a small set of primitive skills in the real world and yet still outperformed comparative baselines. The abstracted simulator and error correcting mechanisms also alleviated the risks associated with traditional sim-to-real systems. We believe this work provides a powerful first framework for training complex robot policies which balance safety, training efficiency and reliability considerations. Importantly, many aspects of the framework are self-contained, such as the error correction, which is left purposefully basic in this work, or method of determining the value function used to navigate the domain. Thus, there is room for future work to improve upon pieces of the framework while still utilising the demonstrated benefits of the levels of abstraction.

Chapter 11

Conclusion

This thesis has explored the fundamental challenges in building artificial agents that can effectively assist humans in a variety of tasks, embodying the FIRe desiderata of *flexibility*, *instructability*, and *reliability*. Through an examination of reinforcement learning methods and their limitations, particularly in addressing these desiderata, we have identified the need for a principled framework that leverages the principle of *compositionality*.

By extending the existing logical composition framework of Nangue Tasse *et al.* [2020b] to stochastic tasks with arbitrary rewards, introducing the concept of *world value functions* (WVFs) to encode knowledge about environment dynamics with all possible goal-reaching tasks in it, and developing methods for agents to provably understand and follow language instructions, this work makes significant strides towards addressing these challenges. Our contributions span across multiple facets of AI research, from theoretical advancements in task composition and safety guarantees to algorithmic implementations and evaluations for empirical results. We have demonstrated how our framework enables agents to efficiently learn and generalise across tasks, ensuring robust performance irrespective of the specific RL algorithms used for learning.

11.1 Discussion and future work

While this thesis makes significant progress towards the goal of general purpose agents, there is of course much room for improvement. In fact, each chapter of this thesis opens up a number of interesting and relevant questions for future works.

Safe reinforcement learning

This chapter investigates a new approach towards safe-RL by asking the question: *Is a scalar reward enough to solve tasks safely?* While we show that it is indeed enough, both theoretically and experimentally, the current approach is only applicable to unsafe terminal states—which only covers tasks that can be naturally represented by stochastic-shortest path MDPs. Hence this may not be applicable to all conceivable safe RL settings. However, since our approach relies on the concept of MDP diameter and controllability, which are properties of the entire MDP and not just the terminal states, our approach may also be extendable in future work to other settings. Finally, given that other popular RL settings like discounted MDPs can be converted to stochastic shortest path MDPs [Bertsekas 1987; Sutton and Barto 1998], a promising future direction could be to find the dual of our results for other theoretically equivalent settings.

We also only consider terminal states that are maximally unsafe. This leads to very risk-averse policies, as shown by the trajectories produced by our TRPO-Minmax agent. Additionally, it may be desirable for an agent to accommodate different degrees of safety—for example "breaking a vase" is less unsafe than "hitting a baby". Our focus on scalar rewards at unsafe states leads to a natural future extension to the case of different degrees of safety. Since the Minmax reward is the least negative reward that guarantees safety ($\bar{R}_{MIN} - \epsilon$), we may assign weights to it corresponding to different levels of unsafety. Here, smaller weights would lead to policies that pass near unsafe states, and large weights would lead to the safest policy that chooses the longer path to the goal.

World value functions

We introduced a new form of goal-oriented value function that encodes knowledge about how to solve all possible goal-reaching tasks in the world. This value function can be learned in a sample efficient manner, and can subsequently be used to infer the dynamics of the environment for model-based planning, or solve new tasks zero-shot given just their terminal rewards. An obvious path for future work is to extend these results to the stochastic high-dimensional setting. While we have demonstrated that WVFs can be learned with neural networks, planning in high-dimensional environments is still an open challenge; WVFs may provide a promising avenue for unifying both learning and planning in this space. Overall, our work is a step towards more general agents capable of solving any new task they may encounter.

Logical composition of skills

In this chapter, we only considered the logical composition of WVFs for goal-reaching tasks. Given how we defined these tasks as MDPs in the same deterministic environment sharing the same non-terminal rewards, a clear direction for future works is extending these to wider families of tasks. For example, tasks in stochastic environments, or partially observable environments, or even more general non-goal-reaching tasks like the ones considered in Chapter 3. Finally, a particularly interesting extension could be to temporal logic tasks like LTL and Turing machines, where the agent learns a more complex formal language representation of long-horizon tasks.

Generalisation in lifelong RL

In this chapter, we developed a framework with theoretical guarantees for an agent to quickly generalise over a task space by autonomously determining whether a new task can be solved zero-shot using existing skills, or whether a task-specific skill should be learned few-shot. However, one limitation of this work and previous works is the finite goal space. This is not a strong practical limitation since one can think of $\mathcal{G}$ as the set of latent goal states in a partially observable setting. For example, when the goal states are defined as continuous regions of the state space, the goal observations may not be goal states; however, the latent goal states are usually still finite. In this case one may need only learn a map from the goal observations to the latent goal states (for example with a supervised learning or clustering approach). This is a promising experimental extension for future work.

Just like previous works, we also inherit some of the problems in regular RL. SOPGOL relies on an RL method that is able to learn goal-reaching tasks (e.g DQN). If the RL method is unable to reach goals in the environment of interest, then it is unable to learn the tasks and SOPGOL will be unable to learn the task vectors. An interesting direction for future work will be to combine SOPGOL with methods that attempt to address the delayed reward problem (e.g with reward shaping or subgoal temporal compositions depending on the setting).

Natural language instruction following

In this chapter, we investigated a sampling based approach and an end-to-end approach for learning to translate from natural language instructions to the correct Boolean expressions that solve given tasks. While these showed promising results, one drawback is that they are both limited and functions in smaller domains compared to the previously stated works [Ahn *et al.* 2023; Driess *et al.* 2023]. We also do not use the full repertoire of Boolean compositions available in Chapter 6. Future work could investigate utilising more complex Boolean expressions and tasks. By creating more complex expressions, the number of tasks can be scaled to evaluate the ability of the agent to learn and generalise to many more tasks. Our model is also limited to expressions of two variables, and future work can investigate composing an unlimited number variables with more complex expressions. Language also has properties besides attribute composition that are not investigated in the RL literature or our work, such as recursion and temporal operators.

Temporal logic composition

In this chapter, we showed how agents equipped with the right type of knowledge representation (skill primitives) are able to leverage the temporal logic structure in formal languages (via reward machines) to achieve great sample efficiency and generalisation to arbitrary new tasks. However, to achieve zero-shot composition, we assumed that every proposition is reachable from every state. Additionally, to ensure that the optimal policy of tasks is also the policy that maximises the probability of satisfying the LTL specification, we also assumed that the environment dynamics are deterministic. A clear avenue for future works is to relax these assumptions.

Finally, we also focused on tasks specified using regular languages, which is the weakest formal language in Chomsky's hierarchy (but stronger than propositional logics). While these are popular in prior works, a natural avenue for future works is to extend our results to more powerful formal languages. Eventually, we want to build our way up to Turing machines, such that any task that is specifiable in any language is also immediately solvable.

11.2 Concluding remarks

Moving forward, our work paves the way for the development of artificial agents that can seamlessly integrate into our daily lives, assisting us in a wide range of tasks with reliability and efficiency. By embracing the principles of compositionality and continually refining our methods, we can achieve ever greater levels of intelligence and autonomy in artificial systems, bringing us closer to realising the vision of general purpose artificial intelligence.

Appendix A

Natural language instruction following

A.1 Appendix A

Task primitive	Success rate
pickup_ball	0.997 ± 0.004
pickup_box	0.996 ± 0.006
pickup_key	1.000 ± 0.000
pickup_red	0.996 ± 0.005
pickup_blue	0.999 ± 0.003
pickup_green	1.000 ± 0.000
pickup_grey	0.996 ± 0.005
pickup_purple	0.996 ± 0.005
pickup_yellow	0.995 ± 0.008

Table A.1: The mean success rate of the individual pretrained world value functions for each task primitive over 100 episodes. The standard deviations are over 10 runs.

A.2 Learning the world value functions with DQN

The DQNs used to learn the world value functions have the following architecture, with the CNN part being identical to that used by Mnih *et al.* [2015]:

1. Three convolutional layers:
    (a) Layer 1 has 3 input channels, 32 output channels, a kernel size of 8 and a stride of 4.
    (b) Layer 2 has 32 input channels, 64 output channels, a kernel size of 4 and a stride of 2.
    (c) Layer 3 has 64 input channels, 64 output channels, a kernel size of 3 and a stride of 1.
2. Two fully-connected linear layers:

(a) Layer 1 has input size 3136 and output size 512 and uses a ReLU activation function.

(b) Layer 2 has input size 512 and output size 7 with no activation function.

We used the ADAM optimiser with batch size 256 and a learning rate of 10^{-3}. The target Q-network was updated every 1000 steps, and we used ϵ-greedy exploration, annealing ϵ from 1.0 to 0.1 over 1000000 timesteps.

Appendix B

Skill machines: Temporal logic composition

B.1 Details of Experimental Settings

In this section we elaborate further on the hyper-parameters for the various experiments in Section 9.2. We also describe the pretraining of WVFs for all of the experimental settings which corresponds to learning the task primitives for each domain. The same hyper-parameters are used for all algorithms in a particular experiment. This is to ensure that we evaluate the relative performance fairly and consistently. The full list of hyper-parameters for the Office World, Moving Targets and SafeAI Gym domain experiments are shown in Tables B.1-B.3 respectively.

Hyper-parameter	Value
Timesteps	$1e^5$
Training exploration (ϵ)	0.5
Per-episode evaluation exploration (ϵ)	0.1
Discount Factor (γ)	0.9

Table B.1: Table of hyper-parameters used for Q-learning in the Office World experiments.

Hyper-parameter	Value
Timesteps	$1e^6$
Neural Network architecture	$CNN + MLP$
CNN architecture	Defaults of Mnih *et al.* [2015]
MLP hidden layers	$1024 \times 1024 \times 1024$
Start exploration (ϵ)	1
End exploration (ϵ)	0.1
Exploration decay duration (ϵ)	$5e^5$
Discount Factor (γ)	0.99
Others	Defaults of Mnih *et al.* [2015]

Table B.2: Table of hyper-parameters used for Deep Q-learning in the Moving Targets experiments.

Hyper-parameter	Value
Timesteps	$1e^6$
Neural Network architecture	MLP
MLP hidden layers	$2024 \times 2024 \times 2024$
Max episodes length	100
Target noise	0.2
Action noise	0.2
Discount Factor (γ)	0.99
Others	Defaults of Achiam [2018]

Table B.3: Table of hyper-parameters used for the TD3 in the SafeAI Gym experiments.

To use skill machines we require pre-trained WVFs. As mentioned above, all WVFs are trained using the same hyper-parameters as any other training. Additionally, for all experiments the WVFs are pre-trained on the base task primitives for the domain. For example, in the Office World domain, the WVFs are trained on the $|\mathcal{P} \cup \mathcal{C}|$ base task primitives corresponding to achieving each predicate, $\mathcal{P} = \{A, B, C, D,$ ✻, ☕, ✉, 🚹, ✉$^+$, 🚹$^+\}$ (reaching states the predicate is set to True), with constraints $\mathcal{C} = \{\widehat{✻}\}$. All other primitives in this domain can be obtained zero-shot through value function composition. Similarly, for the moving targets domain (Figure 9.7), the WVFs are pre-trained on the primitives corresponding to obtaining objects by shape or colour in the environment separately, $\mathcal{P} = \{$□, 🟦, 🟥, ≣, ○$\}$, with constraints $\mathcal{C} = \widehat{\mathcal{P}}$. From here the value functions for finding objects of particular colours or any more complex primitives can be composed zero-shot. Finally, for the SafeAI Gym environment the base skill primitives correspond to going to a *cylinder* (⬮), a *button* (●), and a *blue region* (◯): $\mathcal{P} = \{$⬮, ●, ◯$\}$, trained with constraints $\mathcal{C} = \{\widehat{◯}\}$.

References

[Abel *et al.* 2018] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman. Policy and value transfer in lifelong reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 20–29, 2018.

[Achiam *et al.* 2017] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proceedings of the International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[Achiam 2018] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[Ackley *et al.* 1985] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[Adamczyk *et al.* 2023a] J. Adamczyk, A. Arriojas, S. Tiomkin, and R. V. Kulkarni. Utilizing prior solutions for reward shaping and composition in entropy-regularized reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6658–6665, 2023.

[Adamczyk *et al.* 2023b] J. Adamczyk, S. Tiomkin, and R. Kulkarni. Leveraging prior knowledge in reinforcement learning via double-sided bounds on the value function. *arXiv preprint arXiv:2302.09676*, 2023.

[Ahn *et al.* 2023] M. Ahn, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR, 2023.

[Alshiekh *et al.* 2018] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Altman 1999] E. Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

[Alver and Precup 2022a] S. Alver and D. Precup. Constructing a good behavior basis for transfer using generalized policy updates. In *International Conference on Learning Representations*, 2022.

[Alver and Precup 2022b] S. Alver and D. Precup. Constructing a good behavior basis for transfer using generalized policy updates. In *International Conference on Learning Representations*, 2022.

[Amodei *et al.* 2016] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[Anderson *et al.* 2018] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

[Andreas *et al.* 2016] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[Andrychowicz *et al.* 2017] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[Araki *et al.* 2021] B. Araki, X. Li, K. Vodrahalli, J. DeCastro, M. Fry, and D. Rus. The logical options framework. In *Proceedings of the International Conference on Machine Learning*, pages 307–317. PMLR, 2021.

[Arjona-Medina *et al.* 2019] J. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

[Arumugam *et al.* 2019] D. Arumugam, J. K. Lee, S. Saskin, and M. L. Littman. Deep reinforcement learning from policy-dependent human feedback. *arXiv preprint arXiv:1902.04257*, 2019.

[Bagaria and Konidaris 2019] A. Bagaria and G. Konidaris. Option discovery using deep skill chaining. In *International Conference on Learning Representations*, 2019.

[Barreto *et al.* 2017] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4055–4065, 2017.

[Barreto *et al.* 2018] A. Barreto, D. Borsa, J. Quan, T. Schaul, D. Silver, M. Hessel, D. Mankowitz, A. Zidek, and R. Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *Proceedings of the International Conference on Machine Learning*, pages 501–510. PMLR, 2018.

[Barreto *et al.* 2019] A. Barreto, D. Borsa, S. Hou, G. Comanici, E. Aygün, P. Hamel, D. Toyama, S. Mourad, D. Silver, D. Precup, et al. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[Barreto *et al.* 2020] A. Barreto, S. Hou, D. Borsa, D. Silver, and D. Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020.

[Barreto *et al.* 2021] A. Barreto, D. Borsa, S. Hou, G. Comanici, E. Aygün, P. Hamel, D. Toyama, J. Hunt, S. Mourad, D. Silver, et al. The option keyboard: Combining skills in reinforcement learning. *arXiv preprint arXiv:2106.13105*, 2021.

[Barto and Mahadevan 2003] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

[Bertsekas and Tsitsiklis 1991] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[Bertsekas 1987] D. P. Bertsekas. *Dynamic Programming: Determinist. and Stochast. Models*. Prentice-Hall, 1987.

[Blukis *et al.* 2020] V. Blukis, Y. Terme, E. Niklasson, R. A. Knepper, and Y. Artzi. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *Conference on Robot Learning*, pages 1415–1438. PMLR, 2020.

[Brohan *et al.* 2022] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[Camacho *et al.* 2019] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, volume 19, pages 6065–6073, 2019.

[Chaplot *et al.* 2018] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Chevalier-Boisvert *et al.* 2018] M. Chevalier-Boisvert, L. Willems, and S. Pal. *Minimalistic Gridworld Environment for OpenAI Gym*. https://github.com/maximecb/gym-minigrid, 2018.

[Chevalier-Boisvert *et al.* 2019] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019.

[Cho *et al.* 2014] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[Chomsky 1956] N. Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

[Chow *et al.* 2018] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[Cohen *et al.* 2021] V. Cohen, G. Nangue Tasse, N. Gopalan, S. James, M. Gombolay, and B. Rosman. Learning to follow language instructions with compositional policies. In *AAAI Fall Symposium Series*, 2021.

[Cohen *et al.* 2022] V. Cohen, G. Nangue Tasse, N. Gopalan, S. James, R. Mooney, and B. Rosman. End-to-end learning to follow language instructions with compositional policies. In *Workshop on Language and Robotics at CoRL*, 2022.

[Colas *et al.* 2019] C. Colas, P. Fournier, M. Chetouani, O. Sigaud, and P. Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 1331–1340, 2019.

[Dalal *et al.* 2018] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[Degrave *et al.* 2022] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[Desai *et al.* 2020] S. Desai, H. Karnan, J. P. Hanna, G. Warnell, and P. Stone. Stochastic grounded action transformation for robot learning in simulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6106–6111. IEEE, 2020.

[Devidze *et al.* 2021] R. Devidze, G. Radanovic, P. Kamalaruban, and A. Singla. Explicable reward design for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34:20118–20131, 2021.

[Devlin *et al.* 2019] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186, 2019.

[Driess *et al.* 2023] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[Duan *et al.* 2016] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the International Conference on Machine Learning*, pages 1329–1338, 2016.

[Duret-Lutz *et al.* 2016] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0—a framework for LTL and ω-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis*, pages 122–129. Springer, 2016.

[Dzifcak *et al.* 2009] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *2009 IEEE International Conference on Robotics and Automation*, pages 4163–4168. IEEE, 2009.

[Fawzi *et al.* 2022] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[Finn *et al.* 2017] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[Foster and Dayan 2002] D. Foster and P. Dayan. Structure in the space of value functions. *Machine Learning*, 49(2-3):325–346, 2002.

[Fox *et al.* 2016] R. Fox, A. Pakman, and N. Tishby. Taming the noise in reinforcement learning via soft updates. In *32nd Conference on Uncertainty in Artificial Intelligence*, 2016.

[Fu *et al.* 2016] J. Fu, S. Levine, and P. Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4019–4026. IEEE, 2016.

[Fu 2016] M. C. Fu. Alphago and monte carlo tree search: the simulation optimization perspective. In *Proceedings of the 2016 Winter Simulation Conference*, pages 659–670. IEEE Press, 2016.

[Fujimoto *et al.* 2018] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[Ganin *et al.* 2016] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030, 2016.

[Golemo *et al.* 2018] F. Golemo, A. A. Taiga, A. Courville, and P.-Y. Oudeyer. Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pages 817–828. PMLR, 2018.

[Gopalan *et al.* 2018] N. Gopalan, D. Arumugam, L. Wong, and S. Tellex. Sequence-to-sequence language grounding of non-markovian task specifications. *Robotics: Science and Systems XIV*, 2018.

[Grätzer 2002] G. Grätzer. *General lattice theory*. Springer Science & Business Media, 2002.

[Grätzer 2011] G. Grätzer. *Lattice theory: foundation*. Springer Science & Business Media, 2011.

[Haarnoja *et al.* 2018a] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE International Conference on Robotics and Automation*, pages 6244–6251, 2018.

[Haarnoja *et al.* 2018b] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[Haarnoja *et al.* 2018c] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[Hanna and Stone 2017] J. P. Hanna and P. Stone. Grounded action transformation for robot learning in simulation. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[HasanzadeZonuzy *et al.* 2021] A. HasanzadeZonuzy, A. Bura, D. Kalathil, and S. Shakkottai. Learning with safety constraints: Sample complexity of reinforcement learning for constrained MDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7667–7674, 2021.

[Hopcroft *et al.* 2001] J. E. Hopcroft, R. Motwani, and J. D. Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[Hu *et al.* 2018] R. Hu, J. Andreas, T. Darrell, and K. Saenko. Explainable neural computation via stack neural module networks. In *Proceedings of the European Conference on Computer Vision*, pages 53–69, 2018.

[Hunt *et al.* 2019] J. Hunt, A. Barreto, T. Lillicrap, and N. Heess. Composing entropic policies using divergence correction. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2911–2920. PMLR, 09–15 Jun 2019.

[Hutsebaut-Buysse *et al.* 2022] M. Hutsebaut-Buysse, K. Mets, and S. Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1):172–221, 2022.

[Icarte *et al.* 2018] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 2107–2116, 2018.

[Icarte *et al.* 2022] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73:173–208, 2022.

[Jaksch *et al.* 2010] T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[Jang *et al.* 2017] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with Gumbel-softmax. In *International Conference on Learning Representations*, 2017.

[Jeong *et al.* 2019] R. Jeong, J. Kay, F. Romano, T. Lampe, T. Rothorl, A. Abdolmaleki, T. Erez, Y. Tassa, and F. Nori. Modelling generalized forces with reinforcement learning for sim-to-real transfer. *arXiv preprint arXiv:1910.09471*, 2019.

[Jiang *et al.* 2021] Y. Jiang, T. Zhang, D. Ho, Y. Bai, C. K. Liu, S. Levine, and J. Tan. Simgan: Hybrid simulator identification for domain adaptation via adversarial reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2884–2890. IEEE, 2021.

[Jothimurugan *et al.* 2019] K. Jothimurugan, R. Alur, and O. Bastani. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

[Jothimurugan *et al.* 2021] K. Jothimurugan, S. Bansal, O. Bastani, and R. Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34:10026–10039, 2021.

[Kaelbling 1993] L. P. Kaelbling. Learning to achieve goals. In *International Joint Conferences on Artificial Intelligence*, pages 1094–1099, 1993.

[Kirkpatrick *et al.* 2017] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[Konidaris and Barto 2009] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in Neural Information Processing Systems*, 22, 2009.

[Kuo *et al.* 2021] Y.-L. Kuo, B. Katz, and A. Barbu. Compositional RL agents that follow language commands in temporal logic. *Frontiers in Robotics and AI*, 8:689550, 2021.

[Kupferman and Vardi 2001] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal methods in system design*, 19:291–314, 2001.

[Lake and Baroni 2018] B. Lake and M. Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*, pages 2873–2882. PMLR, 2018.

[Levy *et al.* 2017] A. Levy, R. Platt, and K. Saenko. Hierarchical actor-critic. *arXiv preprint arXiv:1712.00948*, 12, 2017.

[Li *et al.* 2017] X. Li, C.-I. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3834–3839, 2017.

[Lipton *et al.* 2016] Z. C. Lipton, K. Azizzadenesheli, A. Kumar, L. Li, J. Gao, and L. Deng. Combating reinforcement learning's Sisyphean curse with intrinsic fear. *arXiv preprint arXiv:1611.01211*, 2016.

[Littman *et al.* 2017] M. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan. Environment-independent task specifications via GLTL. *arXiv preprint arXiv:1704.04341*, 2017.

[Liu *et al.* 2022] J. X. Liu, A. Shah, E. Rosen, G. Konidaris, and S. Tellex. Skill transfer for temporally-extended task specifications. *arXiv preprint arXiv:2206.05096*, 2022.

[Loshchilov and Hutter 2019] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[Love *et al.* 2022] T. Love, D. Jarvis, G. Nangue Tasse, B. Ingram, S. James, and B. Rosman. Facilitating safe sim-to-real through simulator abstraction and zero-shot task composition. In *Workshop on Lifelong Learning of High-level Cognitive and Reasoning Skills at IROS*, 2022.

[Maei *et al.* 2009] H. Maei, C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver, and R. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. *Advances in Neural Information Processing Systems*, 22, 2009.

[Matas *et al.* 2018] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743. PMLR, 2018.

[Mendez and Eaton 2023] J. A. Mendez and E. Eaton. *How to Reuse and Compose Knowledge for a Lifetime of Tasks: A Survey on Continual Learning and Functional Composition*, 2023.

[Mirowski *et al.* 2017] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In *International Conference on Learning Representations*, 2017.

[Mnih *et al.* 2015] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Moore *et al.* 1999] A. Moore, L. Baird, and L. Kaelbling. Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In *Proceedings of the international joint conference on artificial intelligence*, pages 1316–1323, 1999.

[Mordatch *et al.* 2016] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 242–248. IEEE, 2016.

[Nangue Tasse *et al.* 2020a] G. Nangue Tasse, S. James, and B. Rosman. A Boolean task algebra for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:9497–9507, 2020.

[Nangue Tasse *et al.* 2020b] G. Nangue Tasse, S. James, and B. Rosman. *A task algebra for agents in reinforcement learning*. Master's thesis, School of Computer Science and Applied Mathematics at the University of the Witwatersrand, 2020.

[Nangue Tasse *et al.* 2022a] G. Nangue Tasse, S. James, and B. Rosman. World value functions: Knowledge representation for multitask reinforcement learning. In *The 5th Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2022.

[Nangue Tasse *et al.* 2022b] G. Nangue Tasse, B. Rosman, and S. James. World value functions: Knowledge representation for learning and planning. *Bridging the Gap Between AI Planning and Reinforcement Learning Workshop at ICAPS*, 2022.

[Nangue Tasse *et al.* 2023a] G. Nangue Tasse, S. James, and B. Rosman. Generalisation in lifelong reinforcement learning through logical composition. In *International Conference on Learning Representations*, 2023.

[Nangue Tasse *et al.* 2023b] G. Nangue Tasse, T. Love, M. Nemecek, S. James, and B. Rosman. ROSARL: Reward-only safe reinforcement learning. *arXiv preprint arXiv:2306.00035*, 2023.

[Nangue Tasse *et al.* 2024] G. Nangue Tasse, D. Jarvis, S. James, and B. Rosman. Skill Machines: Temporal logic skill composition in reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[Ng *et al.* 1999] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

[Niu *et al.* 2024] Y. Niu, Y. Pu, Z. Yang, X. Li, T. Zhou, J. Ren, S. Hu, H. Li, and Y. Liu. Lightzero: A unified benchmark for Monte Carlo tree search in general sequential decision scenarios. *Advances in Neural Information Processing Systems*, 36, 2024.

[Osiński *et al.* 2020] B. Osiński, A. Jakubowski, P. Ziecina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6411–6418. IEEE, 2020.

[Peng *et al.* 2018] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[Peng *et al.* 2019] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine. MCP: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems*, pages 3686–3697, 2019.

[Perez *et al.* 2018] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[Peters *et al.* 2018] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[Pnueli 1977] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[Punjani and Abbeel 2015] A. Punjani and P. Abbeel. Deep learning helicopter dynamics models. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3223–3230. IEEE, 2015.

[Puterman 2014] M. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Radford *et al.* 2018] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. *Improving language understanding by generative pre-training*. Technical report, 2018.

[Radford *et al.* 2019] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Raffel *et al.* 2020] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research 21*, 2020.

[Ray *et al.* 2019] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning.(2019). *OpenAI*, 2019.

[Robinson and Voronkov 2001] A. J. Robinson and A. Voronkov. *Handbook of automated reasoning*, volume 1. Elsevier, 2001.

[Russell and Norvig 2016] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[Rusu *et al.* 2017] A. A. Rusu, M. VeVcerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on robot learning*, pages 262–270. PMLR, 2017.

[Sahni *et al.* 2017] H. Sahni, S. Kumar, F. Tejani, and C. Isbell. Learning to compose skills. *arXiv preprint arXiv:1711.11289*, 2017.

[Saxe *et al.* 2017] A. Saxe, A. Earle, and B. Rosman. Hierarchy through composition with multitask LMDPs. *Proceedings of the International Conference on Machine Learning*, pages 3017–3026, 2017.

[Schaul *et al.* 2015] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR.

[Schrittwieser *et al.* 2020] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[Schulman *et al.* 2015] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning*, pages 1889–1897. PMLR, 2015.

[Schultz *et al.* 1997] W. Schultz, P. Dayan, and P. R. Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[Sennrich *et al.* 2016] R. Sennrich, B. Haddow, and A. Birch. *Neural Machine Translation of Rare Words with Subword Units*, 2016.

[Shridhar *et al.* 2021] M. Shridhar, L. Manuelli, and D. Fox. CLIPort: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, 2021.

[Singh *et al.* 2009] S. Singh, R. L. Lewis, and A. G. Barto. Where do rewards come from? In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 2601–2606. Cognitive Science Society, 2009.

[Singh *et al.* 2021] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2021.

[Sootla *et al.* 2022] A. Sootla, A. I. Cowen-Rivers, T. Jafferjee, Z. Wang, D. H. Mguni, J. Wang, and H. Ammar. Sauté RL: Almost surely safe reinforcement learning using state augmentation. In *Proceedings of the International Conference on Machine Learning*, pages 20423–20443. PMLR, 2022.

[Stooke *et al.* 2020] A. Stooke, J. Achiam, and P. Abbeel. Responsive safety in reinforcement learning by PID Lagrangian methods. In *Proceedings of the International Conference on Machine Learning*, pages 9133–9143. PMLR, 2020.

[Subrahmanyam 1964] N. Subrahmanyam. Boolean vector spaces. *Mathematische Zeitschrift*, 83(5):422–433, 1964.

[Sutton and Barto 1998] R. Sutton and A. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[Sutton and Barto 2018] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sutton *et al.* 1999] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[Sutton *et al.* 2011] R. Sutton, J. Modayil, M. Delp, T. Degris, P. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[Sutton 1988] R. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[Sutton 1990] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[Szabó 2020] Z. G. Szabó. Compositionality. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2020 edition, 2020.

[Taigman *et al.* 2016] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[Tambwekar *et al.* 2021] P. Tambwekar, A. Silva, N. Gopalan, and M. Gombolay. Interpretable policy specification and synthesis through natural language and rl. *arXiv preprint arXiv:2101.07140*, 2021.

[Taylor and Stone 2009] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: a survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[Tennenholtz *et al.* 2022] G. Tennenholtz, N. Merlis, L. Shani, S. Mannor, U. Shalit, G. Chechik, A. Hallak, and G. Dalal. Reinforcement learning with a terminator. *Advances in Neural Information Processing Systems*, 35:35696–35709, 2022.

[Thrun 1996] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646, 1996.

[Tobin *et al.* 2017] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[Todorov *et al.* 2012] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[Todorov 2007] E. Todorov. Linearly-solvable Markov decision problems. In *Advances in Neural Information Processing Systems*, pages 1369–1376, 2007.

[Todorov 2009] E. Todorov. Compositionality of optimal control laws. In *Advances in Neural Information Processing Systems*, pages 1856–1864, 2009.

[Tzeng *et al.* 2014] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[Urain *et al.* 2023] J. Urain, A. Li, P. Liu, C. D'Eramo, and J. Peters. Composable energy policies for reactive motion generation and reinforcement learning. *The International Journal of Robotics Research*, 42(10):827–858, 2023.

[Vaezipoor *et al.* 2021] P. Vaezipoor, A. C. Li, R. A. T. Icarte, and S. A. Mcilraith. LTL2action: Generalizing LTL instructions for multi-task RL. In *Proceedings of the International Conference on Machine Learning*, pages 10497–10508. PMLR, 2021.

[Van Hasselt *et al.* 2016] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[van Niekerk *et al.* 2019] B. van Niekerk, S. James, A. Earle, and B. Rosman. Composing value functions in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, volume 97, pages 6401–6409. PMLR, 09–15 Jun 2019.

[Vaswani *et al.* 2017] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[Veeriah *et al.* 2018] V. Veeriah, J. Oh, and S. Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

[Wagener *et al.* 2021] N. C. Wagener, B. Boots, and C.-A. Cheng. Safe reinforcement learning using advantage-based intervention. In *Proceedings of the International Conference on Machine Learning*, pages 10630–10640. PMLR, 2021.

[Wang and Deng 2018] M. Wang and W. Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[Watkins 1989] C. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[Williams *et al.* 2018] E. Williams, N. Gopalan, M. Rhee, and S. Tellex. Learning to parse natural language to grounded reward functions with weak supervision. In *IEEE International Conference on Robotics and Automation*, pages 4430–4436, 2018.

[Wolf *et al.* 2020] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[Yang *et al.* 2020] T. Yang, J. Rosca, K. Narasimhan, and P. Ramadge. Projection-based constrained policy optimization. *arXiv preprint arXiv:2010.03152*, 2020.

[Yang *et al.* 2023] Y. Yang, Y. Jiang, Y. Liu, J. Chen, and S. E. Li. Model-free safe reinforcement learning through neural barrier certificate. *IEEE Robotics and Automation Letters*, 8(3):1295–1302, 2023.

[Zhang *et al.* 2020] Y. Zhang, Q. Vuong, and K. Ross. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 33:15338–15349, 2020.

[Zhao *et al.* 2020] W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[Zhou *et al.* 2023] Z. Zhou, J. Song, X. Xie, Z. Shu, L. Ma, D. Liu, J. Yin, and S. See. Towards building AI-CPS with NVIDIA isaac sim: An industrial benchmark and case study for robotics manipulation. *arXiv preprint arXiv:2308.00055*, 2023.