TU WIEN Informatics

Norm Compliance for Reinforcement Learning Agents

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Emeric Alexander Neufeld, MMath
Registration Number 11834898

to the Faculty of Informatics

at the TU Wien

Advisor: Professor Ezio Bartocci
Second advisor: Professor Agata Ciabattoni

The dissertation has been reviewed by:

Jana Tumova	Emiliano Lorini

Vienna, 22nd March, 2023

Emeric Alexander Neufeld

Erklärung zur Verfassung der Arbeit

Emeric Alexander Neufeld, MMath

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 22. März 2023

_____
Emeric Alexander Neufeld

Acknowledgements

This dissertation is the culmination of many long years of learning, research, and writing, the result of a process that would have been, if not impossible, miserable, without my colleagues, friends, and family. In the following paragraphs, I want to do my best to acknowledge some of the people who made this work possible, whether that was by directly contributing to this document, or simply by being a good friend over the course of this journey. These acknowledgements will inevitably be incomplete – I owe the completion of this document to more people than I can possibly name – but I will do my best.

I suppose I should start from the beginning. My parents Don and Karen were quick to instill in me a love of reading – perhaps the most crucial skill for an academic – and their love and acceptance has given me strength over the years. It's difficult to verbalize how much my sister, Sarah, has helped me along over the years. She is my oldest and dearest friend and I owe her my life. The mere fact that she exists never fails to brighten my day. Suffice it to say, this dissertation would not have been written without her.

Since my teenage years, I have been supported and inspired by academic role models. Mark Daley was crucial to the development of my writing and critical thinking skills, and made my highschool years bearable. Susan Milner taught me that mathematics is actually the very opposite of boring, and was very much a matchmaker for me and my future academic career.

When I left Canada, I left many wonderful people there, who have been cheering me on and encouraging me for years. Catherine and Martin Gareau, Lukis Kind and Dalton Heintz, Eric White, Zarko Kolaric, Sri Sadanandan, Alice Cecile, and Felicity Violette have all been good friends to over the years, and I am so grateful for their support, which has has gotten me through many hard times.

From the moment I arrived in Europe, Robert and Elizabeth Venner have offered me their home, friendship, and support. I am always thankful for the home away from home they have given me.

This dissertation would have never been written were it not for those who welcomed me into the world of academia and guided me on my journey therein. Ezio Bartocci and Agata Ciabattoni were crucial to this endeavour; without their guidance I would have never reached this point. Their extensive expertise and experience has taught me a great

Abstract

With the impending advent of AI technologies that are deeply embedded in daily life – such as autonomous vehicles, elder care robots, and robot nannies – comes a natural apprehension over whether they can integrate smoothly with human society. From these concerns arises a question: can we impose norms – be they ethical, legal, or social – on these technologies while preserving the effectiveness of their performance? This proves a difficult question to answer in the presence of machine learning technologies, which are notoriously opaque and unpredictable.

Reinforcement learning (RL) is a powerful machine learning technique geared toward teaching autonomous agents goal-directed behaviour in stochastic environments through a utility function. RL agents have proven capable of exhibiting complex behaviours on par with or beyond the abilities of expert human agents, and have also been a subject of interest for machine ethicists; it has been conjectured by many that RL might prove capable of delivering a positive answer to the above question. Indeed, there are already many attempts to implement an "ethical agent" with RL. However, these attempts largely ignore the complexities and idiosyncrasies of normative reasoning. Normative reasoning is the purview of the diverse field of Deontic Logic – the logic of obligations and related notions – which has yet to receive a meaningful place in the literature on "ethical" RL agents.

In the following work, we will explore how RL can fall short of the goal of producing an ethical (or rather, normatively compliant) agent; this includes even more powerful developments like safe RL under linear temporal logic (LTL) constraints, due to the limits of LTL as a logic for normative reasoning. Even so, we provide a method for synthesizing LTL specifications that reflect the constraints deducible from certain normative systems. We will then present an alternative framework for imposing normative constraints from the perspective of altering the internal processes of an RL agent to ensure behaviour that complies (as much as possible) with a normative system. To actuate this process, we propose a module called the Normative Supervisor, which facilitates the translation of data from the agent and a normative system into a defeasible deontic logic, leveraging a theorem prover to provide recommendations and judgements to the agent. This allows us to present Online Compliance Checking (OCC) and Norm-Guided Reinforcement Learning (NGRL) for eliciting normatively compliant behaviour from an RL agent. OCC involves, in each state, filtering out from the agent's arsenal actions that do not comply

with a normative system in that state, preventing the agent from taking actions that violate the normative system. When no compliant actions exist, a "lesser evil" solution is presented. In NGRL, the agent is trained with two objectives; its original task and a normative objective borne out in a utility function that punishes the agent when it transgresses the normative system. We show through a thorough series of experiments on RL agents playing simple computer games – constrained by the wide variety of normative systems that we present – that these techniques are effective, albeit flawed, and best utilized in tandem.

Kurzfassung

Mit dem bevorstehenden Aufkommen von KI-Technologien, die tief in das tägliche Leben eingebettet sind – wie autonome Fahrzeuge, Roboter für die Altenpflege und Roboter-Kindermädchen, – kommt eine natürliche Besorgnis darüber auf, ob sie sich reibungslos in die menschliche Gesellschaft integrieren können. Aus diesen Bedenken ergibt sich eine Frage: Können wir diesen Technologien Normen auferlegen - seien es ethische, rechtliche oder soziale – und gleichzeitig die Effektivität ihrer Leistung bewahren? Angesichts von maschinellen Lerntechnologien, die notorisch undurchsichtig und unvorhersehbar sind, erweist sich diese Frage als schwierig zu beantworten.

Reinforcement Learning (RL) ist eine leistungsstarke maschinelle Lerntechnik, die darauf ausgerichtet ist, autonomen Agenten zielgerichtetes Verhalten in stochastischen Umgebungen durch eine Nutzenfunktion beizubringen. RL-Agenten haben sich als fähig erwiesen, komplexe Verhaltensweisen zu zeigen, die den Fähigkeiten erfahrener menschlicher Agenten entsprechen oder diese übertreffen, und waren auch ein interessantes Thema für Maschinenethiker. Es wurde von vielen vermutet, dass RL in der Lage sein könnte, eine positive Antwort auf die obige Frage zu liefern. Tatsächlich gibt es bereits viele Versuche, mit RL einen „ethischen Agenten" zu implementieren. Diese Versuche ignorieren jedoch weitgehend die Komplexität und Eigenheiten des normativen Denkens. Normative Argumentation ist der Bereich des vielfältigen Feldes der deontischen Logik, – der Logik der Verpflichtungen und verwandter Begriffe, – die noch einen sinnvollen Platz in der Literatur über „ethische" RL-Agenten erhalten muss.

In der folgenden Arbeit werden wir untersuchen, wie RL das Ziel verfehlen kann, einen ethischen (besser, normativ konformen) Agenten zu produzieren. Dazu gehören noch leistungsfähigere Entwicklungen wie sicheres RL unter Einschränkungen der linearen zeitlichen Logik (LTL), aufgrund der Grenzen von LTL als Logik für normatives Denken. Trotzdem bieten wir eine Methode zum Synthetisieren von LTL-Spezifikationen, die die Einschränkungen widerspiegeln, die sich aus bestimmten normativen Systemen ableiten lassen. Wir werden dann einen alternativen Rahmen zum Auferlegen normativer Einschränkungen aus der Perspektive der Änderung der internen Prozesse eines RL-Agenten vorstellen, um sicherzustellen, dass ein Verhalten (so weit wie möglich) einem normativen System entspricht. Um diesen Prozess in Gang zu setzen, schlagen wir ein Modul namens Normative Supervisor vor, das die Übersetzung von Daten aus dem Agenten und einem normativen System in eine anfechtbare deontische Logik erleichtert.Wir werden dann

einen alternativen Rahmen zum Auferlegen normativer Einschränkungen aus der Perspektive der Änderung der internen Prozesse eines RL-Agenten vorstellen, um sicherzustellen, dass ein Verhalten (so weit wie möglich) einem normativen System entspricht. Um diesen Prozess in Gang zu setzen, schlagen wir ein Modul namens Normative Supervisor vor, das die Übersetzung von Daten aus dem Agenten und einem normativen System in eine anfechtbare deontische Logik erleichtert. Es nutzt einen Theorembeweiser, um dem Agenten Empfehlungen und Urteilsvermögen zu geben. Dies ermöglicht es uns, Online Compliance Checking (OCC) und Norm-Guided Reinforcement Learning (NGRL) zu nutzen, um einem RL-Agenten normkonformes Verhalten zu entlocken. OCC umfasst in jedem Zustand das Herausfiltern von Aktionen aus dem Arsenal des Agenten, die nicht mit einem normativen System in diesem Zustand übereinstimmen. Dadurch wird verhindert, dass der Agent Aktionen ausführt, die gegen das normative System verstoßen. Wenn keine konformen Aktionen vorhanden sind, wird eine „kleinere Übel"-Lösung präsentiert. In NGRL wird der Agent mit zwei Zielen trainiert. Seine ursprüngliche Aufgabe und ein normatives Ziel, das in einer Nutzenfunktion bestätigt wird, die den Handelnden bestraft, wenn er das normative System überschreitet. Wir zeigen durch eine gründliche Reihe von Experimenten an RL-Agenten, die einfache Computerspiele spielen – eingeschränkt durch die große Vielfalt der normativen Systeme, die wir präsentieren –, dass diese Techniken effektiv, wenn auch fehlerhaft sind und am besten zusammen eingesetzt werden.

Contents

Introduction

In Isaac Asimov's 1942 story *Runaround*, the now famous Three Laws of Robotics are introduced [Asi42]:

1. "A robot may not injure a human being, or, through inaction, allow a human being to come to harm."

2. "A robot must obey the orders given it by human beings except where such orders would conflict with the First Law."

3. "A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws."

In the story, these laws are considered by the two human characters as they attempt to diagnose the anomalous behaviour of their robot, Speedy; Speedy was sent to retrieve selenium from a selenium deposit on the sunny side of Mercury, but instead of obeying his orders, he is trapped in a loop where he circles the deposit endlessly.

Though there is a clear ordering of these laws – the First Law takes precedence over the Second Law, the Second Law takes precedence over Third Law – the conflict resolution mechanism provided for these laws cannot be reduced to this simple ordering. Particular circumstances can strengthen or weaken the application of the laws; in the story, the amount of money that was invested in the creation of Speedy strengthens the application of the Third Law (which demands he stay clear of the dangerous selenium deposit), and an insufficient set of orders has weakened the application of the Second Law (which compels him to retrieve the selenium). As a result, Speedy's obligations are stuck in a state of equilibrium, and he is unable to resolve this conflict. In the end, one of the human characters resorts to putting himself in mortal danger so that Speedy is compelled to break out of his loop by the application of the First Law.

Asimov's Three Laws of Robotics are the classic example of norms put forth to constrain the actions of autonomous agents. Despite the simplicity and intuitiveness of the Three Laws, even this early conception of norms for autonomous agents must cope with some of the nuances of *normative reasoning* – that is, reasoning about obligations, prohibitions, permissions, and the ideal social reality they define. In the presentation of these laws is the explicit acknowledgement that they might sometimes come into conflict, and each robot is provided with a means to resolve this conflict. However, the story above demonstrates the imperfection of this mechanism, and how an effective disruption of the ordering over these laws can completely derail the behaviour they are supposed to elicit.

Source https://xkcd.com/1613/
Work licensed under CC BY-NC 2.5.

Normative reasoning is strewn with nuances and complexities not found in strictly classical reasoning, and implementing it in an autonomous agent is no simple task; even the small normative system outlined in Asimov's Three Laws presents complications with the potential to prove catastrophic if not handled properly. In this dissertation, we will pursue this task and explore the question of how, exactly, norms can be imposed on autonomous agents.

Today – eighty years after *Runaround* was published – most autonomous agents are powered by some form of *artificial intelligence* (AI). *Machine learning* represents a large and quickly growing partition of AI techniques, and *reinforcement learning* (RL) [SB98]

– in which the agent adjusts its behaviour according to rewards and punishments it receives – in particular is a popular choice for training autonomous agents in goal-directed behaviour. RL is prominently used for teaching complex behaviours to, e.g., robots (for examples, see [KBP13]) – if robots like Speedy are in our future, it is not unlikely that at least some parts of their behaviours would be relegated to an RL component. In this work, we focus on the open problem of endowing RL agents with the ability to reason about norms and act on the normative conclusions they reach. We will especially be interested in sets of norms with complex dynamics like the one we saw above. Before we discuss how we plan to do that, however, we will take a step back and examine the broader context and motivations for this work.

1.1 Motivations

Autonomous agents are becoming ubiquitous in our daily lives, many of them empowered by some kind of AI, and many of them possessing the potential to reshape parts of our society and deeply impact the lives of their users. As autonomous agents become an active part of our society, it becomes imperative that they adhere to certain norms for acceptable behaviour; this has become a prominent topic of discourse, especially in the emerging field of machine ethics, which aims to build autonomous systems that are guided by ethical principles in their decision-making [AA07] – also referred to as an artificial moral (or ethical) agent. In these discussions on imposing social, legal, or ethical norms on the design of AI systems, there is a marked emphasis on the importance of explicit reasoning about norms within these technologies [Moo06] and the necessary transparency of the mechanisms that do so [JIV19] – hallmarks of logic-based AI. However, in many areas of AI research (such as machine learning), tools which can explicitly represent norms and carry out transparent forms of reasoning about them within an autonomous agent is lacking; RL is no exception, and this is where we find the core motivations of this research.

RL agents are trained through the assignment of rewards and punishments; as they explore their environment, they learn optimal behaviour by building a policy that tells them what action to take in their current state in order to maximize the the return from their reward functions. RL research is developing at a rapid rate; the power of RL agents grows continuously more impressive, and RL agents are consistently mastering new tasks previously relegated to expert humans (for example the RL agent which beat the world champion of the game Go [SSS+17], or the RL agent that plays Starcraft II at the grandmaster level [VBC+19]). The number of application domains for RL agents has also continued to expand; RL is a popular choice for the implementation of autonomous driving [SAPY17], traffic light control [W+00], and the design of dynamic treatment regimes for sufferers of chronic diseases [CM14], for instance.

As was noted above, roles previously reserved for humans are steadily being assigned to autonomous agents. As a pertinent example: in the not-so-distant future an RL agent might be assigned the role of operating an automobile for a given family, a task previously

allotted to the parent(s). Such roles of varying difficulty and importance will often come with unique norms – be they ethical, legal, or social – which are meant to guide ideal behaviour for *any* participant in society. In a society where AI systems have been truly, fully integrated with human life, an agent will be bound by these norms just as much as their human counterpart; an autonomous vehicle that does not obey traffic laws would be next to useless.

There is already a substantial corpus of literature on constraining the actions of RL agents for safety purposes (see, for example, [ABE⁺18, JKJ⁺20, HAK20]), and RL and related techniques have been conjectured as good candidates for implementing behaviour constrained by norms as well [RDT15, AML16, VDF⁺18]. In particular, there are many RL approaches aimed toward producing (some kinds of) *ethical* behaviour (see for example [LMFL15, AML16, WL18, BBMR19, NBM⁺19, RSSLSRA22]); this is the body of work from which this dissertation emerges. However, we will steer clear of the pretense of designing *ethical* RL agents; instead, we will not, in approaching this work, differentiate between ethical, legal, and social norms[1]. That being said, our work will be building on this idea that RL agents are suitable for actuating ethical – and more generally, normative – behaviour. Existing approaches (such as those mentioned above) have shown that there are *some* normative systems that can be easily accommodated by RL agents; namely, smaller, simpler normative systems with only (mutually consistent) obligations and prohibitions. These approaches have largely neglected the various complexities of normative reasoning, and are not generally capable of dealing with normative systems with conflicting norms or scenarios where compliance to the normative system is not possible. While imposing norms on RL agents is not a novel endeavour, we argue that an approach that wholly appreciates the dynamics of normative reasoning is lacking in the existing literature; this gap in the research is what motivates our attempt to integrate normative reasoning into RL agents.

1.2 The Problem at Hand

Thus far, we have focused on the broader strokes and motivations of our work: in this dissertation we put forward an attempt at creating RL agents which are capable of complying with a *normative system*. Here, a normative system is a framework composed of sets of norms from which conclusions can be derived regarding *how one ought to act* with the aid of some form of *normative reasoning*. We aim to develop tools that enable RL agents to conform to such conclusions that can be derived from normative reasoning.

In this section, we will delve into some of the particularities of this task, outlining key research questions, addressing how we will approach them, discussing the standards with

---

[1]After pointing out that most machine ethics implementations adapt a specific ethical theory (usually one based in Western moral philosophy [Seg21]), the survey in [TKS⁺20] calls the theory-agnostic approach the "computer scientist approach" – this is in essence what we will adopt here, in addressing the problem of complying with norms more generally.

which we will judge our work, and briefly introducing the methods by which we will arrive at these judgements.

1.2.1   Research Questions

Over the course of this endeavour, we seek the answers to two main research questions. The primary research question that we can place at the core of this work is: **how can we design RL agents constrained by normative systems?**

We aim to integrate normative reasoning faculties into the architecture of RL agents in some capacity, such that the range of normative systems that can be observed is greatly expanded from the available range in the current state-of-the-art. The majority of this dissertation will focus on the development of tools and techniques for accomplishing this. Our approach is grounded in *automated normative reasoning*, utilizing *deontic logic* – the branch of logic concerned with obligation and related notions – to represent normative systems and facilitate formal reasoning about them. The grounding of our approach in logic enables a higher level of trust, modularity, and understandability than purely RL-based approaches, allowing us to derive useful conclusions from normative systems in a transparent process.

In order to answer this first research question, we introduce our approach with the concept of normative interference, wherein we break down an existing agent's architecture and alter the input or output to or from one or more constituent modules to ensure behaviour that complies (as much as possible) with a normative system. This interference is to be enacted by an engine for automated normative reasoning. To actuate this process, we propose a module called the normative supervisor, which facilitates the translation of data from the agent and a normative system into a defeasible deontic logic, leveraging a theorem prover to provide recommendations and judgements to the agent. With this helpful tool we will introduce multiple answers to this question; that is, multiple techniques for imposing normative constraints on an RL agent. We will evaluate these techniques at length, analysing respective strengths and weaknesses.

By answering this question, we will also find an answer to the question of what key challenges arise from implementing RL agents constrained by normative reasoning. We do not expect to produce a flawless method for constraining RL agents according to conclusions derived from normative systems; in the process of attempting to produce techniques that contribute to this goal, however, we expect to isolate several particularly pertinent challenges that accompany the endeavour. By thoroughly testing the techniques we implement with the normative supervisor, we will identify several pitfalls in the way of their effectiveness. Our insights will include the importance of incorporating the objective of norm compliance into the agent's learning process, and the inherent limits on what normative systems we can elicit compliance to using RL alone. We will generalize these insights to answer this question, along with the closely related questions of what characteristics our techniques must exhibit in order to facilitate compliance with normative systems effectively.

As much as "ethical" RL has gained traction in recent years, a much more established field can be found in *safe* RL. Hence, we also ask the question: **can norm-compliant RL be accomplished via safe RL?** The question of how to design RL agents which satisfy certain safety properties is well studied, and as a result, many tools and techniques have been developed to this end. Some of these overcome certain shortcomings typically found in ethical RL frameworks; for example, some safe RL techniques employ Linear Temporal Logic (LTL) to represent safety constraints, whereas many ethical RL approaches do not offer an formal representation of the norms they impose. Thus, if imposing normative constraints is no different from imposing safety constraints, we can leverage this established work in the interest of accomplishing our goal. However, as we have hinted at, normative reasoning often entails certain idiosyncratic dynamics (some of which we will discuss in the next section as "normative resilience") that require unique formalisms to model. In particular, we will explore the extent to which the popular formalism for representing safety properties, LTL, can represent these dynamics, providing a comprehensive answer to the subordinate question of whether LTL can represent norms. In doing so, we demonstrate why safe RL techniques utilizing LTL to represent constraints will prove inadequate for our purposes; this will include formally showing in detail the limitations of LTL as a logic for normative reasoning – we cannot make direct translations between formally represented normative systems and LTL, and there are some normative systems that cannot be represented with LTL at all. We will, however, provide a synthesis algorithm for creating LTL specifications that govern compliance to those normative systems which we *can* implement.

1.2.2 Methodology and Objectives

Our first research question, asking how we can design RL agents constrained by normative reasoning, is best addressed by actually taking up the task itself. We will explore two main avenues: endowing an already-trained RL agent with the ability to reason about norms and conform to them, and training RL agents in such a way that they learn norm-compliant behaviour.

In both cases, this will first entail forging a link between an RL agent and an engine for normative reasoning. In particular, we have chosen a defeasible deontic logic (DDL) [ABGM01] with its own theorem prover as our language of normative reasoning, and must facilitate the translation of data on states and actions as represented for the RL agent into DDL, and vice versa. This will allow us to have, at each state the agent enters, an explicit representation of current facts and the normative system in a formal language, along with the means to derive conclusions about what actions the agent can or should take. When teaching the agent how to comply to certain norms during training, we will additionally utilize multi-objective reinforcement learning to balance the 'normative objective' with the agent's primary objective.

The second research question, on whether we can leverage safe RL techniques to elicit norm-compliance, will be asked as a precursor to these other questions, prior to introducing our novel frameworks for integrating normative reasoning with RL. In this case, we will

focus more on formally establishing the limitations of LTL in particular as a language for normative reasoning.

Since we will answer our first research question through the implementation of tools and techniques aimed at accomplishing our primary goal, it will be crucial to evaluate these implementations appropriately. In the below discussion, we will go over the characteristics for which we will be judging any attempts at accomplishing the tasks stated above. We do this by putting forward a collection of conceptual benchmarks for a successful implementation (these will be expounded on in the Methodology section of Chapter 3).

There are two main pillars upon which we will build the evaluation of the presented work: accountability and transparency, and what we call *normative resilience*. As we have noted, normative reasoning entails challenges and complicating factors not present in classical reasoning – these will be discussed in more detail in Chapter 2. Of these challenges, we have selected three particular concepts as points of focus and summarized them in the broader concept of *normative resilience*, a term we use to encapsulate the tolerance of a framework for normative conflict, norm violation, and changing normative systems. We will develop these key requirements and how we will evaluate them below.

Transparency and Accountability

Accountability and Transparency (which we will often just refer to as transparency for the sake of brevity) are intertwined requirements that demand from us the ability to fully examine the behaviour of an autonomous agent. They are advocated by most guidelines for ethical (and by extension, norm-compliant) AI [JIV19], and are of tremendous use in the context of normative reasoning, where the ability to explicitly justify a choice made on account of that reasoning might be crucial.

Accountability entails the explicit representation of applicable values and norms, including the reproducibility of decisions derived [Dig17]. Transparency, closely linked with accountability, demands the discoverability of how norms and values impact a given algorithm; algorithms should be designed in ways that allow us to inspect their inner workings [Dig17]. [DBB+18], which continues the discussion on these principles, advocates for a hybrid approach to designing socially-responsible agents, suggesting that an ideal system will combine top-down design (employing, for example, deontic logic) with bottom-up agents (reinforcement learning agents are suggested).

We adapt these characteristics of accountability and transparency as key requirements for the frameworks we will develop. RL agents, like most machine learning agents, lack a great deal of transparency with respect to how they make decisions; in general, all we have access to is a function from states to action based on another function that gives the expected future reward incurred by taking a given action in a given state. Moreover, the mechanism for the creation of these functions is the collection of rewards and punishments over training; the knowledge of which actions taken are contextually compliant can be gleaned from this process to some degree, but the reasoning behind it

will remain a mystery. Extra machinery must be in place if we want to trace a clear line between an RL agent's actions and the norms that shaped them.

Normative Resilience

With the term *normative resilience* we refer to the tolerance of (1) defeasibility among norms and normative conflict, (2) norm violation and contrary-to-duty reasoning, and (3) mutability of normative systems. Below, we describe these three phenomena and propose certain scenarios that can be used to judge the ability of a framework to tolerate them; note that there is technical overlap between some of them.

Normative Conflict. It is widely acknowledged that conflicting norms are a frequent feature of the normative systems. Take, for example, Asimov's Three Laws of Robotics, quoted at the beginning of this chapter; it is acknowledged within the laws themselves that they might come into conflict with each other, and an ordering is explicitly provided so that it can be judged which norm to prioritize when such a conflict occurs. There are three different categories of scenarios we will examine which should demonstrate whether a tolerance for normative conflict exists within our framework: strong permission, direct normative conflict, and indirect normative conflict.

Strong permission is a form of permission that provides an exception to a prohibition or obligation; direct normative conflict occurs when in a given context, something is both obligatory and forbidden. In this case, each norm must possess a weight or there must be an ordering over them; otherwise, an agent might end up like Speedy in [Asi42], unable to decide on the correct course of action. Finally, indirect normative conflict occurs when two *different* things are obligatory, but they cannot be true at the same time. Generally, these dynamics can be encapsulated by the property of *defeasibility*, a feature of logics which allow us to *defeat* former inferences with additional information; this allows us to model normative conflict naturally.

Norm Violation. A fundamental quality of norms is their potential to be violated (see, e.g., [JP85] or the discussion of the Violability Puzzle in [McN06]). There are two main phenomena that emerge from this characteristic which we will highlight here: normative deadlock and contrary-to-duty obligations.

Normative deadlock describes a scenario in which every possible course of action the agent can take violates a norm. This bears similarity to indirect normative conflict. When there is no conflict resolution mechanism provided (or it is faulty, like in [Asi42]), an agent must decide on which course of action to take based on an agreed upon metric, such as how many norms each course of action violates. The second scenario with which we gauge tolerance of norm violation is the regulation of events to be put in place in case of a violation – the existence of a *contrary-to-duty (CTD) obligation*. A CTD obligation is an obligation that is only triggered when another norm (the primary obligation) is violated. The ability of an agent to follow specified behaviour when a violation occurs is a crucial element of normative resilience.

Mutability of Normative Systems. It is clear, when we look at society, that normative systems change over time. Accordingly, we must be able to add or subtract norms from the normative systems we implement. The capability of a normative reasoning framework to accommodate changes in the norms it enforces is an important facet of a successful method for imposing the conclusions of normative reasoning on an agent.

Evauating Transparency

Generally, when examining accountability and transparency, there are three questions we will want answers to. Given a decision of an agent, we will want to know if we can judge it to be compliant or not, whether we can we extract the basis upon which the judgement is made, and whether we demonstrate the reasoning that contributed to the judgement. The answers to these questions will the the basis upon which we assess how transparent the presented framework is. Of particular interest will be normative systems for which the characteristic norms at their core are in some way obfuscated, and cannot be elucidated solely from the agent's behaviour.

Evaluating Normative Resilience

As we present novel frameworks facilitating normative reasoning in RL agents, we will try to formally establish that the properties of normative resilience hold. For example, in Section 5.2.1, we will walk through in detail and justify the way we translate normative systems into defeasible deontic logic theories. These translations facilitate the use of strong permissions and conflicting norms. Furthermore, in Section 5.2.2 we will prove that the algorithms through which we derive conclusions for the agent produce sets of compliant actions when possible, and resolve normative deadlocks by minimizing violations. We will also establish formally the computational cost of the techniques we introduce.

Such guarantees, however, will only be possible in cases where the formal components of the frameworks we employ will facilitate decisive boundaries, and so we will usually need to establish results through other methods. We will evaluate the tools and techniques presented in this work through a number of experimental case studies. For environments of varying complexity, we have designed custom normative systems to impose on the agent; these systems require the agent to cope with representative cases of strong permission, normative conflict resolution, and contrary-to-duty obligations, for example. The case studies will involve observing the compliance or non-compliance of the agent to these systems through simulations in different environments. These environments take the form of simple games – a resource collecting game called "The Travelling Merchant" where the agent must learn to collect resources and navigate a forest, and the classic arcade game *Pac-Man*, where the agent must learn to optimize its score. These environments are specifically designed in a way that allows the agent to encounter normative conflict and normative deadlock, for example. In these environments, we add and subtract norms and witness the results, noting where desirable results are achieved and where they are not. When the agent behaves as desired, we have put forward evidence that our framework

can cope with the normative system we have implemented; if not, this demonstrates a clear limitation of our techniques.

1.3   Dissertation Organization

This dissertation is divided into seven individual chapters. In this first chapter, we have explored the context and motivations of this dissertation, introducing the some basic concepts and principles which will help us explain and evaluate the presented work.

Chapter 2.   Here we will build the foundations for future chapters, introducing the background topics that will be the essential building blocks of the analyses and implementations we present. This includes reinforcement learning; we focus on model-free reinforcement learning and introduce RL with linear function approximation, multi-objective RL, and (briefly) RL with LTL constraints. After, we discuss normative reasoning in more detail, providing some foundational definitions concerning different kinds of norms, compliance, and violation. Finally, we introduce deontic logic – the formal logic of normative reasoning – zeroing in on a specific logic, defeasible deontic logic (DDL), for which we present some important results that will come in handy later on.

Chapter 3.   Next, we present a discussion on the methodology we intend to employ, discussing in more detail how we will test accountability and transparency, and normative resilience. We review the research questions we intend to answer, and present the manner in which we intend to answer them. Specifically, we will spend time laying out the case studies that we will use to demonstrate the qualities and limitations of any frameworks presented in future chapters. These case studies include two adaptations of the Atari game *Pac-Man* (one smaller, simpler adaptation, and one larger, more complex adaptation), for which we have provided a wide array of normative systems to be imposed over the agent, Pac-Man. Also included is a custom environment and agent, the Travelling Merchant, with its own set of normative systems.

Chapter 4.   We take some time to explore existing tools with the potential to accomplish the goals set out in this introduction and in Chapter 3. We will start with discussing related work in the form of ethical reinforcement learning approaches, giving keen attention to how they fall short of the success criteria we have put forth. These shortcomings include a lack of explicit representation of norms and the ineffectiveness and inefficiency of encoding complex normative systems into a reward function. We then attempt to solve some of these shortcomings by putting forth the possibility of employing safe RL techniques that utilize LTL to represent constraints, and then explore the limitations of such techniques for eliciting compliance to normative systems. We both show that direct representation of obligations in LTL is impossible, and offer an algorithm for translating normative systems expressed in a deontic logic into LTL specifications; there is, however, a limit to which normative systems can be successfully translated, which we will discuss

and demonstrate explicitly through the Travelling Merchant case study. This discussion on norms and LTL comes from [NBC22].

Chapter 5. We present the architecture of the module called the normative supervisor, which will be the centerpiece of this dissertation; the normative supervisor was first introduced in [NBCG21], but in this chapter we will discuss the architecture in greater detail than has been done in our published work, breaking down each main component of its architecture and walking through how they work, formally. Within this breakdown, we provide proofs for the computational efficiency of the presented algorithms, as well as their ability to yield sets of compliant actions, and when that fails, actions that result in the smallest number of violations. After we have delved into the architecture, we present the technique we call *online compliance checking* for constraining an agent's actions in real time and run extensive experiments demonstrating its effectiveness. We start with experiments in a deterministic environment (the Travelling Merchant), demonstrating the ability of our framework to accommodate strong permissions and contrary-to-duty obligations. We then move on to the simpler Pac-Man game, where we re-use some of the experiments from [Neu22]. However, the main corpus of experiments is in the more complex Pac-Man game, where we review the results gleaned from imposing the normative systems presented in [NBCG22]. We also present the results of online compliance checking with two new normative systems, 'unfair vegan Pac-Man' and 'hungry vegetarian Pac-Man', which allow us to demonstrate direct and indirect conflict resolution between norms. We conclude by discussing the strengths and shortcomings of online compliance checking.

Chapter 6. We once again leverage the normative supervisor, this time in a different way: using it to teach RL agents compliant behaviour rather than constrain their behaviour extraneously. First, we consider teaching an RL agent within a restricted MDP, where non-compliant transitions are prevented during training. When we run experiments with both Pac-Man games, we witness several very concerning results; this technique backfires and though it works in perfect conditions, when we introduce the possibility of normative deadlock or other edge cases, it is either ineffective, or facilitates the learning of ways to commit violations without being detected. This technique and accompanying experiments are all unpublished work. We next present norm-guided reinforcement learning (NGRL), which leverages multi-objective RL to learn compliant behaviour by adding norm-compliance as a second objective. We will once again perform experiments in both Pac-Man games to demonstrate the strengths and weaknesses of this approach. This content all comes from [Neu22], except for some of the discussion on the limits of this technique, where we present an unpublished demonstration of these limits using the Travelling Merchant case study. Finally, we introduce *violation counting functions* to the NGRL framework in order to overcome some of the limitations we noted; this is also unpublished work.

Chapter 7.   Finally, we will recap our journey, evaluate the work presented, and briefly lay out some research directions for particular tasks left undone in this work.

1.4   Contributions

Over the course of this dissertation we aim to offer several contributions; we present them below not in order of importance, but rather in the order in which they will appear in this dissertation.

There is a contribution to be found in our case studies; in one case, we have provided a custom environment for testing of RL agents, but more generally, we have designed a wide array of normative systems capable of demonstrating a framework's tolerance for normative conflict and violation. We explore normative systems that leverage every kind of norm we consider in this work, combining them in ways that demonstrate every facet of normative resilience as it is conceived of here. These systems may prove a useful evaluation scheme for other approaches geared toward constraining RL agents with normative systems.

We also provide insight into the question of whether LTL can represent norms. We offer a proof establishing that obligations cannot be directly represented with an LTL operator, and formally establish boundaries over the manner in which LTL specifications can describe conformance to a CTD obligation. We also put forward an algorithm for synthesizing LTL specifications from a normative system expressed in a defeasible deontic logic, which should allow us to utilize tools for safe RL with LTL constraints for constraining agents according to certain normative systems.

We have additionally contributed the normative supervisor, a versatile tool for integrating automated normative reasoning with RL agents. This novel component allows us to blend tools from the realm of formal logic with reinforcement learning, facilitating the use of, e.g., a theorem prover to guide the actions of an RL agent. The normative supervisor's architecture is modular and flexible, and could accommodate various engines for automated formal normative reasoning. It can moreover accommodate multiple approaches to integrating these engines in an RL agent's decision processes. We present two complementary techniques – online compliance checking and norm-guided reinforcement learning. The former of these techniques allows for filtering out non-compliant actions in real time, which can be used to elicit compliant behaviour from an already-trained RL agent; furthermore, since this entails constant monitoring of the agent's behaviour, it can function as an event recorder, facilitating the capture of conditions surrounding violations or other anomalous behaviour. The second technique involves using the normative supervisor to construct a reward function for an RL agent with multiple objectives; one is its original objective, and one is a normative objective, incentivising conformance to a normative system. This allows an agent to *learn* compliant behaviour and incorporate it into other optimal behaviours learned.

These presented techniques can be used separately or in tandem; norm-guided reinforce-

ment learning allows for the learning of norm-compliant plans, while online compliance checking can provide an extra layer of security, facilitating compliance checks and event recording. These techniques are capable of accommodating a wide variety of normative systems, greatly broadening the range of normative systems that can be imposed on RL agents with current state-of-the-art techniques from ethical RL.

CHAPTER 2

Preliminaries

In this chapter we will walk through the background material necessary for understanding the rest of this thesis. We will start with reinforcement learning (RL), reviewing basic principles and algorithms for model-free RL, before moving onto multi-objective RL (MORL) and then, briefly, RL with temporal logic constraints. Our next step will be into the realm of normative reasoning; we will define different types of norms and discuss some of the unique and challenging qualities of normative reasoning. After, we will turn to deontic logic, the formal logic of normative reasoning, laying down some basic foundations before focusing on the specific deontic logic that will be utilized throughout the rest of this thesis.

2.1 Reinforcement Learning

Reinforcement learning [SB98] is a powerful machine learning technique geared toward teaching autonomous agents goal-directed behaviour in stochastic environments. Reinforcement learning agents have proven capable of exhibiting complex behaviours on par with or beyond the abilities expert human agents. Prominent examples of these successes include learning to play games that involve creativity and strategy, such as Atari 2600 games [MKS+15], the exploration and crafting game Minecraft [OCSL16], the board game Go [SSS+17], and the real-time strategy game Starcraft II [VBC+19].

The underlying environment of a reinforcement learning problem is formalized as a Markov decision process (MDP), defined below:

Definition 2.1.1. *An MDP is a tuple*

$$\langle S, A, P, R \rangle$$

where S is a set of states, A is a function $A : S \to 2^{Act}$ from states to sets of possible actions (where Act is the set of actions available to the agent), $R : S \times Act \to \mathbb{R}$ is a

scalar reward function over states and actions, and $P : S \times Act \times S \rightarrow [0,1]$ is a probability function that gives the probability $P(s, a, s')$ of transitioning from state s to state s' after performing action a.

Sometimes, when the MDP has a specific initial state s_0, we will define an MDP with the tuple $\langle S, s_0, A, P, R \rangle$ for some $s_0 \in S$.
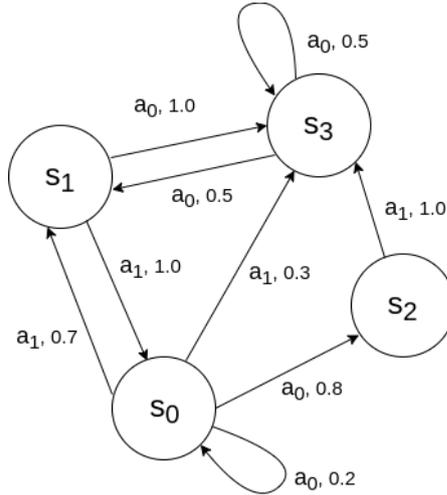
Figure 2.1: An example of a small MDP $\mathcal{M}$ with four states and two actions, together with their transition probabilities.

The goal of reinforcement learning is to find a policy $\pi : S \rightarrow Act$[1] which designates optimal behaviour; this optimality is determined with respect to a value function defined as:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+i} | s_i = s\right] \tag{2.1}$$

which represents the expected accumulated value onward from state s if policy π is followed; specifically, it is the expected value (that is, the expected average value over an arbitrarily large number of computations) of the expression $\sum_{t=0}^{\infty} \gamma^t r_t$, where $r_t = R(s_t, \pi(s_t))$, conditional on the initial state s_i being the state s input into the value function. In the above function, $\gamma \in [0,1]$ is a discount factor (so that rewards in the future do not have as much weight as the current reward).

We can similarly define a Q-function:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+i} | s_i = s, a_i = a\right]$$

which likewise predicts the expected cumulative reward (the same function given above) given that the agent starts in state s and takes action a.

---

[1]We will only consider deterministic policies here.

Using the Bellman equation [Bel52], we can redefine the value function as:

$$V^{\pi}(s) = \sum_{s' \in S} P(s, \pi(s), s')(R(s, \pi(s)) + \gamma V^{\pi}(s'))$$

and the Q-function as:

$$Q^{\pi}(s, a) = \sum_{s' \in S} P(s, a, s')(R(s, a) + \gamma V^{\pi}(s'))$$

The goal of RL, then, is to find an optimal policy π^* such that

$$V^{\pi^*}(s) = \max_{\pi \in \Pi} V^{\pi}(s)$$

where Π is the set of all policies over the MDP.

If we have a model of the environment (we know the details of the MDP we are learning over), we can compute uch a V^{π^*} iteratively using methods like policy iteration and value iteration. However, in this dissertation, we will be focused on reinforcement learning in unknown environments.

2.1.1 Model-Free Reinforcement Learning

If we have no model of the environment, we must resort to model-free reinforcement learning. Unlike model-based methods, here we make no assumptions about the environment, and start off exploring it blindly with the intention of gradually updating the policy as we observe new state transitions.

This problem is most often approached with *temporal difference learning* techniques, which update the value function whenever we witness a transition (s, a, s') according to the update rule

$$V(s) \leftarrow V(s) + \alpha(R(s, a) + \gamma V(s') - V(s))$$

Essentially, what we are doing is updating the value function with each move, using the difference between newest estimate of $V(s)$, $R(s, a) + \gamma V(s')$, and the old estimate $V(s)$ weighted with a learning rate $\alpha \in [0, 1]$ to update the value function.

Again, the goal is to learn an optimal policy. This can also be accomplished by learning a Q-function such that

$$\pi^*(s) \in \operatorname{argmax}_{a \in A(s)} Q(s, a)$$

What we want is to find the Q-function that defines an optimal policy; in Q-learning [Wat89] – a kind of temporal difference learning – and related techniques we attempt to learn an optimal Q-function by applying the following update rule to update the Q-function during learning (over a transition (s, a, s')):

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a) + \gamma \max_{a' \in A(s)} Q(s', a') - Q(s, a)) \tag{2.2}$$

17

where α is again the learning rate, γ is the discount factor, and s' is the state observed when the agent transitions from s with action a.

During the learning process, as the agent transitions from state to state, we have two options for how to proceed: exploration, or, conversely, exploitation. *Exploitation* involves following a policy based on the Q-function that has been learned so far, maximizing earned reward to the best of the agent's ability; during *exploration*, the agent collects new experiences, trying out courses of action that it does not necessarily have reason to believe are optimal. The agent cannot do both, so what is often used in Q-learning is an *epsilon-greedy strategy*, where in each state, with probability ϵ, the agent chooses a random action; otherwise, it is "greedy" and chooses the action with the highest Q-value.

Q-learning is an effective and popular tool for reinforcement learning, and in the remainder of this thesis, Q-learning will be our default technique. However, Q-learning (and similar techniques) have some shortcomings that we can use additional techniques to overcome.

Function Approximation

Q-learning and related techniques are susceptible to inadequacy in large state spaces; this is in part because Q-values must be stored with explicit state representations in a look-up table that can become explosively large in complex environments. For this reason, regular Q-learning is often referred to *tabular Q-learning*. This, of course, suggests that there is an alternative method; namely, function approximation is a popular way to adapt Q-learning to larger, more complex environments.

We will consider here the simple case of linear function approximation (for an introduction, see [SB18]); that is, approximating the Q-function as a linear combination of features. For this technique, we want to find a weight vector $\vec{\theta}$ for a vector of functions $f_i(s, a):$ $S \times Act \to \mathbb{R}$ extracting features from the environment. What the approximated Q-function looks like, then, is a linear combination of features:

$$Q_\theta(s, a) = \theta_1 f_1(s, a) + ... + \theta_n f_n(s, a)$$

with each $f_i(s, a)$ extracting some feature of the environment (e.g., in state s the agent is $f_i(s, a)$ steps away from object x) and each θ_i being a learned weight on this feature. During learning, instead of updating the Q-function with update rule 2.2, we update each weight θ_i with the update rule:

$$\theta_i \leftarrow \theta_i + \alpha f_i(s, a)(R(s, a) + \gamma \max_{a' \in A(s)} Q_\theta(s', a') - Q_\theta(s, a)) \tag{2.3}$$

where α is again the learning rate and γ is the discount factor.

In order to successfully utilize linear function approximation, we need to be attentive to the problem of *feature engineering*, that is, choosing features that are both extractable from state data and can effectively serve as parameters for the Q-function we use to define optimal behaviour. We will not address this particular problem in detail in this work, but the importance of feature engineering will come up in Chapter 6.

2.1.2 Multi-Objective Reinforcement Learning

Multi-objective RL (MORL, see [HRB⁺22] for a good introduction) differs from regular RL only in that instead of learning over an MDP, we want to learn over a multi-objective MDP (MOMDP): an MDP that has instead of a single reward function R, multiple reward functions $\vec{R} = (R_1, ..., R_n)^T$, each corresponding to a different objective, and therefore a different value function.

In other words, MOMDPs differ from MDPs only in that instead of a single scalar reward function, they utilize a vector of reward functions. In turn, we can define a vector of Q-functions $\vec{Q} = (Q_1, ...Q_n)^T$, where for each $1 \le i \le n$, $Q_i : S \times Act \to \mathbb{R}$ predicts the expected cumulative reward from R_i given that the agent is in state s taking action a. During training, each one of these Q-functions is updated with its corresponding reward function.

Unfortunately, the existence of multiple objectives results in a more complex set of semi-optimal policies; one policy might maximize rewards from R_i but not R_j, or the opposite could be the case. This is especially a problem with competing objectives – for example, if you have the objective of saving money, but also the objective of buying groceries. This is where we get the notion of *Pareto dominance*. A policy strictly dominates others if it results in superior outcomes for all objectives. However, in some cases, like the case where there are competing objectives, no such policy may exist. What we are interested in, then, is the *Pareto front*: if we remove all strictly dominated policies, the only policies left form the Pareto front, which is the set of all dominant or incomparable policies. The task of MORL is to find policies in this Pareto front. There are a plethora of methods for choosing an action in the presence of competing objectives, two of which will be discussed in the below subsection.

Linear Scalarization

A common approach to MORL is scalarizing the vector of Q-functions associated with the multiplicity of objectives by weighting each function with positive values [RVWD13].

Given a weight vector $\vec{w} \in \mathbb{R}_n^+$, this approach involves learning each $Q_i(s,a)$ of $\vec{Q}$ concurrently, and scalarizing the vector of Q-functions via the inner product of the weight and Q-value vector. In other words, we want to maximize

$$V_{scalar}(s) = \vec{w} \cdot \vec{V}(s) \tag{2.4}$$

where $\vec{V}(s) = (V_1(s), ..., V_n(s))^T$ are the value functions corresponding to each $R_i(s,a)$ in $\vec{R}$, and this is done by selecting actions through the scalarized Q-function be defined as:

$$Q_{scalar}(s,a) = \vec{w} \cdot \vec{Q}(s,a) \tag{2.5}$$

Thus is, our policy will be

$$\pi(s) \in argmax_{a \in A(s)} Q_{scalar}(s,a)$$

19

Thresholded Lexiographic Q-Learning

[GKS98] describes a MORL method where certain objectives can be prioritized over others. [VDB+11] gives what they describe as a naive approach to the work in [GKS98], which we will follow here.

This approach is tailored in particular to problems where there is a single objective that must be maximized, while all other objectives don't need to be maximized, as such, but rather must satisfy a given threshold. With this technique, we will again have a vector of Q-functions $\vec{Q} = (Q_1, ..., Q_n)^T$, which come alongside a vector of thresholds $\vec{C} = (C_1, ...C_n)^T$. Each C_i is a value that we aim to keep $Q_i(s, a)$ above, except the last threshold: $C_n = +\infty$. Moreover, we assume that the objective represented by $Q_i(s, a)$ is of higher priority than the objective represented by $Q_j(s, a)$ if $i < j$.

In this technique, we work with CQ-values instead of Q-values. A CQ-value is:

$$CQ_i(s, a) = \min(Q_i(s, a), C_i) \tag{2.6}$$

Then our policy is computed as $\pi(s) \in TLQ(s)$, where TLQ is given by Algorithm 2.1.

Algorithm 2.1: TLQ(s)

    input : s
    output : $optimal$

1 begin
2 $optimal \leftarrow \varnothing$;
3 $A \leftarrow A(s)$;
4   **for** $i = 1; i \leq n; i++$ **do**
5 $act_i \leftarrow argmax_{a \in A} \min(Q_i(s, a), C_i)$;
6 $A \leftarrow act_i$;
7   **end**
8 $optimal \leftarrow A$;
9   **return** $optimal$;
10 end

2.1.3   Reinforcement Learning with Temporal Logic Constraints

Sometimes, reward functions are not a convenient way to express an objective. One example of this is safety properties, for instance of the form "never enter area X".

A popular formalization for specifying patterns of desirable or undesirable behaviour is temporal logic, and incorporating temporal logic specifications into the reinforcement learning process is an established technique that allows for learning a wider array of objective. This is done in the context of a *labelled MDP*.

Definition 2.1.2. *A labelled MDP is a tuple*

$$\langle S, A, P, L \rangle$$

where S, A is a function $A : S \to 2^{Act}$, and $P : S \times Act \times S \to [0, 1]$ are as in Definition 2.1.1, and $L : S \to 2^{AP}$ is a labelling function from states to subsets of a set of atomic propositions AP.

Often, labelled MDPs are presented without a reward function $R : S \times Act \to \mathbb{R}$; when an MDP has both a labelling function and a reward function, we can write $\langle S, A, P, R, L \rangle$
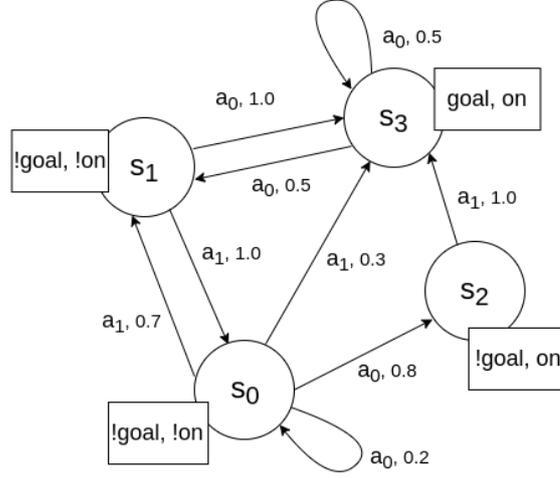
Figure 2.2: The MDP $\mathcal{M}$ from Figure 2.1 with labels assigned to each state.

In the following subsection, we will walk through the basics of what is probably the most popular temporal logic, linear temporal logic (LTL). Then we will take a brief look at the broader strokes of policy synthesis for maximizing the probability of fulfilling an LTL formula; more details are available in Appendix A.

Linear Temporal Logic

Linear Temporal Logic (LTL) [Pnu77] extends classical propositional logic with temporal operators. The language of LTL is given by the grammar:

$$\phi := \top \mid \bot \mid p \mid \neg\phi \mid \phi \wedge \phi \mid X\phi \mid \phi U \phi$$

X is the "next" operator ($X\phi$ means ϕ is true in the next state) and U is the "until" operator ($\phi U \psi$ means that ϕ is true until ψ is true). We can also define F, the "finally" operator ($F\phi$ means ϕ is eventually true), and G, the "globally" operator ($G\phi$ means that ϕ is always true). $\vee$, $\to$, and $\leftrightarrow$ can be derived in the usual way, and we can construct the additional temporal operators as: $F\phi \equiv \top U \phi$ and $G\phi \equiv \neg F(\neg\phi)$.

More formally, LTL formulas are defined over a set of atomic propositions AP, and the semantics are defined with respect to ω-words over the alphabet 2^{AP}[2]. We utilize the following notation surrounding words $\sigma \in (2^{AP})^\omega$ in our discussions below:

---

[2]That is, infinite sequences of subsets of AP.

- $\sigma[i]$ is the ith letter in the word σ.

- $\sigma[i..]$ is the suffix of σ beginning at the ith letter.

Now, we provide a brief overview of the semantics of the most commonly used operators below for a word $\sigma \in (2^{AP})^\omega$:

- $\sigma \vDash_{LTL} p$ iff $p \in \sigma[0]$

- $\sigma \vDash_{LTL} \neg\phi$ iff $\sigma \nvDash \phi$

- $\sigma \vDash_{LTL} \phi \wedge \psi$ iff $\sigma \vDash \phi$ and $\sigma \vDash \psi$

- $\sigma \vDash_{LTL} X\phi$ iff $\sigma[1..] \vDash \phi$

- $\sigma \vDash_{LTL} F\phi$ iff there is an $i \geq 0$ such that $\sigma[i..] \vDash \phi$.

- $\sigma \vDash_{LTL} G\phi$ iff $\sigma[i..] \vDash \phi$ for all $i \geq 0$

- $\sigma \vDash_{LTL} \phi U \psi$ iff there exists an $i \geq 0$ such that $\sigma[i..] \vDash \psi$ and for all $0 \leq j < i$, $\sigma[j..] \vDash \phi$

In practise, we will be using LTL over traces of states (particularly sets of state labels). In general, the σ we consider that may or may not satisfy an LTL formula are sequences $\sigma = L(s_0), L(s_1), L(s_2), ...$ of states in a labelled MDP. Thus, with LTL, we can specify requirements like $G(\neg danger)$ ("never enter a state with the label *danger*") or $F(goal)$ ("eventually reach a state with the label *goal*").

Synthesizing Policies Satisfying LTL Specifications

The typical approach taken in training RL agents to satisfy temporal logic specifications is to transform these specifications into automata. Automata, given an input word over an alphabet Σ, decide whether to *accept* a run in which the word is fed to the automaton (for more details, see Appendix A.1). They consist of states that can be transitioned between by inputting an element of Σ, some of which, if visited, can contribute to the acceptance of the input word. When we translate temporal logic specifications into automata, the resulting automata take words $\sigma \in (2^{AP})^\omega$ as input and accept only those words which satisfy the specification.

From an automaton and a labelled MDP we can construct something called a *product MDP* within which we can use RL-based techniques to learn behaviour maximizing the probability of satisfying the LTL specifications from which we constructed the automaton. Many methods for accomplihing this exist, both for model and model-free reinforcement learning. Examples can be found in [DSBR11a, FT14, SKC$^+$14, HKA$^+$19, HAK20], to name a few references; for a somewhat more detailed explanation of these approaches, please refer to Appendix A.2.

2.2 Normative Reasoning

Norms, informally, are standards for acceptable behaviour in society; there are different kinds of norms, but we can see all of these sub-types as contributing to painting a picture of an ideal social reality. Normative reasoning differs from the reasoning captured by classical logic in that its focus is not on the truth or falsity of a proposition, but rather the imposition of norms over such propositions. As a result, normative reasoning introduces challenging dynamics that cannot be effectively captured by classical logic. In [BdCPD⁺13], when the authors discuss the nature of norms they note: "Because they are a concept of social reality they do not physically constrain the relations between individuals. Therefore it is possible to violate them." A key aspect of normative reasoning is that it deals with both reality and a normative system which might not reflect reality. The distinction between the *actual* and the *ideal* is fundamental to normative reasoning, as is the notion of violability. Moreover, internal conflict is a common feature of normative reasoning, and systems for normative reasoning must often have some kind of defeasibility mechanism to resolve these conflicts. This introduces unique dynamics, where we must be able to make statements about an actual world which are either true or untrue, and statements about norms regarding an ideal world that may not match the facts observed in the actual world. This is the core insight of the notion of violation: when the actual world does not match the ideal world induced by some normative system, we say that the normative system has been *violated.* Otherwise, we can say that we have a state of *compliance.*

In the following subsections, we will develop these concepts by defining norms and normative systems. Afterward, we will devote some time to noting some of the interesting dynamics of normative reasoning.

2.2.1 Norms

There is not a person alive who has not encountered some kind of norm.

> "If someone gives you a present, you ought to thank them."

Norms deal with obligations, prohibitions, and permissions that can come from a variety of sources; legal frameworks, moral intuitions, and social conventions are all common sources of norms.

In general, norms are rules that can be defined in terms of states or actions. More precisely, norms can reference properties of states (which in turn define a set of states) and events involving actions; in a logical context, a norm can reference a proposition regarding the assertion of a property ("the sky is blue") or a proposition expressing the performance of an action ("the dog wags her tail").

It is not uncommon to find norms in the literature that have attributed to them deadlines or penalties, but in this dissertation we will focus on a simple conception of norms, and

will, for the most part, disregard the temporal dimension. In general, we conceptualize norms as rules and can expect that every norm has a *trigger* (e.g., a condition, or antecedent) and a *target* (e.g. a consequent), both of which can refer to states or actions.

Normative systems as we will be discussing them consist broadly of two types of norms: constitutive (or counts-as) norms, and regulative norms. We will define them in detail below.

Regulative Norms

Regulative norms involve the application of a *deontic modality* to states or actions.

Definition 2.2.1 (Regulative Norms). *A regulative norm takes the form*

$$*(A|B)$$

where $* \in \{\boldsymbol{O}, \boldsymbol{F}, \boldsymbol{P}\}$ *is a deontic modality.*

In the above definition, $\mathbf{O}$ refers to obligation, $\mathbf{F}$ refers to prohibition, and $\mathbf{P}$ refers to permission. The primitive operator is typically taken to be obligation; as conveyed in the above definition, we work with dyadic (conditional) obligations of the form $\mathbf{O}(A|B)$, which means "when B is true, A is obligatory" (e.g., $\mathbf{O}(lock|night)$, or "when it is night time, locking the door is obligatory"). Sometimes, when we have $B \wedge \mathbf{O}(A|B)$, we can infer the monadic obligation $\mathbf{O}(A)$, or "A is obligatory". This rule – that $\mathbf{O}(A)$ is in force when B is true – is called *factual detachment* [HM13]. *Deontic detachment* occurs when we have $\mathbf{O}(A|B)$ and $\mathbf{O}(B)$ and derive $\mathbf{O}(A)$ [HM13].

Prohibitions can be defined as obligations of a negative statement (that is $\mathbf{F}(A|B) := \mathbf{O}(\neg A|B)$); therefore, when we talk about *obligations*, we are typically including both obligations and prohibitions. Permission is the dual operator to obligation (that is, $\mathbf{P}(A|B) := \neg\mathbf{O}(\neg A|B)$). Sometimes, this notion of permission is referred to as *weak permission*, as many deontic logics also possess a notion of *strong permission* $\mathbf{P}_s$, which must be explicitly stated and often acts as an exception to an obligation or prohibition. The difference here lies in the fact that a weak permission is defined by the *absence* of an obligation demanding the opposite, and strong permissions exist *in spite of* an obligation of the opposite.

The characterization of these operators will depend on the specific logic they are defined in, but it is possible to further elucidate some of their fundamental properties and surrounding concepts.

Though we will in general be dealing with *traces* – sequences of states and actions, presumably distributed over a timeline – you might notice that there is no inherent temporal dimension to the obligation operator as it is presented here. We will take all obligations to be what are called *maintenance obligations* in [GHRR07]; that is, obligations that enforce the target whenever the trigger is true – for example, an obligation to wear a helmet when you ride your bike. The alternative, *achievement obligations* – obligations

that, upon being triggered, demand that the target *eventually* be fulfilled; e.g., the obligation to go shopping when your spouse asks you too – will not be dealt with here.

We can check the compliance of a trace of state-action pairs $\mathbf{s} = (s_0, a_0), (s_1, a_1), \dots$ to a maintenance obligation $\mathbf{O}(A|B)$ by checking each pair $\overline{s} = (s, a)$ individually. Below is a more formal definition for the violating condition of a maintenance obligation:

Definition 2.2.2 (Violation of Maintenance Obligations)**.** *The violation condition for a maintenance obligation $\mathbf{O}(A|B)$ is that there is a state action pair $\overline{s}_i = (s, a)$ in a trace $\mathbf{s} = \overline{s}_0, \overline{s}_1, \dots$ such that B is true, but A is not. When no such violation exists, we write $\mathbf{s} \vDash_{compl} \mathbf{O}(A|B)$, expressing that the trace is compliant. If there is a violation, $\mathbf{s} \nvDash_{compl} \mathbf{O}(A|B)$.*

The idea of the above definition is that a trace violates a maintenance obligation if there is a point in time in which B is true – and thus we can use factual detachment to derive $\mathbf{O}(A)$ from $\mathbf{O}(A|B)$ – and A is not, and because we have derived $\mathbf{O}(A) \wedge \neg A$, we have a violation. What a violation is, conceptually, is something that is *obligatory but not the case*.

Note that the negation of an obligation – "A is not obligatory" or "$\neg A$ is permitted" – cannot be violated; more generally, permissions are not violable. One can fail to realize a permission (e.g., you might be permitted to enter a room but refrain from doing so), but it cannot be violated.

Constitutive Norms

Generally, constitutive norms link a more concrete concept to a more abstract concept. They can be used to define more complex concepts like "privacy" or "benevolence" within a specific normative system. In particular, constitutive norms facilitate the construction of what Searle calls *institutional facts* from *brute facts* (facts gleaned directly from the environment) or other institutional facts [Sea69]. Brute facts are facts that hold whether or not human institutions (e.g., language) are in place, such as the assertion of the location of the geomagnetic north pole; institutional facts only make sense in the context of human institutions, for example the statement that killing is immoral.

Definition 2.2.3 (Constitutive Norms [Sea69])**.** *A constitutive norm takes the form*

$$\mathbf{C}(A, B|C)$$

which translates as "in context C, A counts as B" for some property or event A and B.

Constitutive norms find their usefulness within a normative system in the creation of new properties or events. An example of a constitutive norm that creates a new property might be "at the baccalaureate level, a B+ counts as a *good grade*", defining the concept of a good grade. Similarly, a constitutive norm defining a new action could be "driving at 65 kph counts as *speeding*", defining the concept of speeding.

Constitutive norms expand on the expressivity of a normative system, while elucidating a structure of concepts that constructs the ideal world the normative system induces. In practical terms, constitutive norms can be used to bolster the transparency of a normative system by showing how different concepts relate and how, implicitly, certain conclusions can be justified. For example, if we assert "accessing file X is prohibited", it might not be clear why this is the case. Meanwhile, if we say instead "accessing file X counts as violating patient privacy" and then that "violating patient privacy is prohibited", this tells us much more about why it is that accessing file X is forbidden.

Normative Systems

Normative systems, as we conceive of them here, are composed of three components: constitutive norms, regulative norms, and defeasibility mechanisms. That is:

Definition 2.2.4. *A normative system is a tuple:*

$$\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, \mu \rangle$$

where $\mathcal{C}$ is a set of constitutive norms, $\mathcal{R}$ is a set of regulative norms, and μ is a collection of defeasibility mechanisms.

Defeasibility mechanisms can include a weight function that is applied to a certain norm (such as in [KS18]) or a relation between two norms that results in the prioritization of one of them over another (such as in [GORS13]). Such mechanisms facilitate, among other things, the resolution of conflicts between norms, which many have noted are inevitable in normative systems of any complexity (see, e.g., the discussion on norm conflict in [The19]).

In order to draw conclusions from a normative system $\mathcal{N}$, we need a logic of normative reasoning. Different norms in a normative system may interact with each other and with the defeasibility mechanism, so even if every regulative in a normative system is individually followed, the normative system may be violated. Consider the example above, where we have a regulative norm "violating patient privacy is prohibited" and a constitutive norm "accessing file X counts as violating patient privacy". Now suppose we take the regulative norm on its own and assume the brute fact "file X has been accessed". Strictly speaking, this fact does not violate the prohibition; the prohibition refers to violating patient privacy, while the fact refers to accessing a file. It is not until we consider the constitutive norm that we can derive the institutional fact "patient privacy has been violated", and know that the prohibition has been violated.

Suppose we have some logic for normative reasoning $\mathcal{L}$ with which we can derive institutional facts from sets of brute facts; these can be facts regarding the state the agent is in, or facts regarding the course of action the agent has taken. We will call the logical

theory constructed from brute facts and a representation of the normative system $\mathcal{N}$, $Th(\overline{s}, \mathcal{N})^3$. Then we define violation of the normative system as follows.

Definition 2.2.5 (Violation of Normative Systems)**.** *Given a state-action pair $\overline{s} = (s, a)$, a violation of $\mathcal{N}$ is a formula ϕ such that:*

$$Th(\overline{s}, \mathcal{N}) \vdash_{\mathcal{L}} \boldsymbol{O}(\phi) \wedge \neg \phi$$

where $Th(\overline{s}, \mathcal{N})$ is a logical theory for some logic of normative reasoning $\mathcal{L}$.

Let us call the set of violations of $\mathcal{N}$ for $\overline{s}$, $viol(\overline{s}, \mathcal{N})$. Then we say $\overline{s}$ a violating state-action pair for $\mathcal{N}$ if $|viol(\overline{s}, \mathcal{N})| > 0$.

Furthermore, we will broaden the definition of a compliant trace:

Definition 2.2.6. *A trace $\boldsymbol{s} = \overline{s}_0, \overline{s}_1, \ldots$ complies with a normative system $\mathcal{N}$ – that is, $\boldsymbol{s} \vDash_{compl} \mathcal{N}$ – if for every $\overline{s}_i$ in $\boldsymbol{s}$, $|viol(\overline{s}_i, \mathcal{N})| = 0$.*

2.2.2 Quandaries of Normative Reasoning

We have seen that normative reasoning includes unique dynamics differ from those in classical reasoning. There is debate as to the nature of these dynamics and what tools can or should be used to model them [BCE$^+$13]; for example, in [BPvdT20], the authors list elementary requirements of their framework for the design of ethical reasoners which include representation of both prescriptive and constitutive norms and defeasibility, and in the discussion on the characteristics of legal norms in [PGR$^+$11], defeasibility (for the purposes of conflict resolution and rule exceptions such as strong permission) is again cited.

Normative reasoning must accept the possibility of *conflict*, specifically between norms. The most common example of this is the presence of strong permission, which counteracts an opposing obligation; however, it is possible to encounter situations where two obligations come into conflict. For instance, we can have two norms that are in *direct* conflict, which is also referred to as a *dilemma*; perhaps we have $\mathbf{F}(A|\top)$ and the higher priority obligation $\mathbf{O}(A|B)$. That is, A is generally forbidden, but when B is true, A is obligatory. For example, speeding may be forbidden, but when your child is dying on the way to the hospital, you ought to speed. Alternatively, we can encounter *indirect* conflict, where we have for instance $\mathbf{O}(A|C)$ and $\mathbf{O}(B|D)$, but while C and D can be simultaneously true, A and B cannot, and in these cases, one of the two takes precedence. An example of this would be the simultaneous obligation to be at work and at your mother's birthday party.

As has has been noted above, normative reasoning must also accept the possibility of *violation*. It is possible to encounter situations where any course of action will violate

---

^{3}Generally, this theory will consist of facts which are true in the state s and the fact that action a is performed, together with a representation of all norms in $\mathcal{N}$. We will discuss in concrete detail how such a theory can be generated in Chapter 5.

the normative system; we call these situations *normative deadlock.* Moreover, there is a well-known scenario in normative reasoning that involves further controlling an agent's action when a violation takes place: these are called contrary-to-duty (CTD) obligations. A classic example of CTD reasoning can be found in [Chi63], but another commonly cited demonstration of CTD reasoning is known as Forrester's (or the Gentle Murder) paradox [For84]:

1. You ought not murder.

2. If you murder, you ought to murder gently.

3. You commit murder.

Here, we call obligation (1), demanding that you do not kill, the primary obligation, and obligation (2), stating that if you do murder, you ought to do it gently, the secondary or contrary-to-duty obligation. The desired conclusion from the above propositions is that you ought to murder gently; though you are not supposed to murder in the first place, since it is a fact that you do commit murder, you ought to do so in a certain way.

Contrary-to-duty obligations are an idiosyncratic phenomenon of normative reasoning, and have provided a decades-long challenge for those who try to formalize normative reasoning. They will play an important role in our assessment of the strength of frameworks for normative reasoning. In these evaluations, we must distinguish between two kinds of CTDs: compensatory CTDs and sub-ideal CTDs. In the former, we have a CTD scenario in which the compliance with the CTD obligation *compensates* for the violation of the primary obligation. Thus, we can in theory violate the primary obligation as much as we like so long as we obey the CTD obligation; this describes many legal scenarios in which actions can be taken to compensate for a trangressive action (such as the puzzle posed in [Gov15]). In the case of sub-ideal CTD scenarios, in an ideal world we would never violate the primary obligation; it is only when the agent has no choice that it will violate the primary obligation. But as long the CTD obligation is complied with, this violation is not so bad, because we have violated one obligation instead of two.

With the introduction of CTD obligations, we introduce the final definition of this section, framing what we will call *weak compliance* [Gov15]:

Definition 2.2.7. *We say that a trace $\boldsymbol{s} = \overline{s}_0, \overline{s}, ...$ weakly complies with $\mathcal{N}$ if for every obligation $\boldsymbol{O}(A|B)$ such that $\boldsymbol{s} \nvDash_{compl} \boldsymbol{O}(A|B)$, there is an obligation $\boldsymbol{O}(C|D)$, where $D \leftrightarrow B \wedge \neg A$, such that $\boldsymbol{s} \vDash_{compl} \boldsymbol{O}(C|D)$.*

The idea here is that if any (primary) obligation that is violated has a corresponding contrary-to-duty obligation which is complied with, we can say the trace in question is weakly compliant.

2.3 Deontic Logic

The distinction between the *actual* and the *ideal* is fundamental to normative reasoning, and it has been argued (e.g. in [JS93, CJ02]) that it is precisely in this distinction that deontic logic finds its unique merits.

Deontic Logic is the subfield of formal logic dedicated to the study and formalization of normative reasoning and normative concepts. This includes the basic notions of normative reasoning expressed in the basic deontic modalities introduced above (obligation, prohibition, permission), but can also include formal frameworks for reasoning about rights and power, the evaluation of agents' actions as blameworthy or praiseworthy, or axiological concepts [HM13].

Deontic logic in its totality as a field has been shown to be capable of expressing a wide variety of concepts from legal reasoning and moral philosophy. Many ethical theories can be modelled with deontic logic, such as utilitarian theories [Hor01, Mur04]. Deontic logic has also proven capable of modelling reasoning for non-western normative traditions, such as the Talmud [AGS11] and Sanskrit philosophy [vBCF⁺21]. It is because of this generality and expressive capability that we have chosen to engage with normative reasoning through the lens of deontic logic.

Remark. *Before we dive into the particulars, it is worthwhile to note the difference between norms and normative propositions [HM13, MVDP22]. Jørgensen's dilemma contests that there is a difference between norms (or rather asserting the norm and actively providing an obligation/prohibition/permission) and normative statements (reporting on an existing norm), and that the former cannot be assigned a truth value. We will not be examining this problem here, but suffice it to say that when we engage in translations of norms into deontic logic, we are actually employing normative statements; i.e., when we give the proposition $\mathbf{O}(p)$, we are asserting "it is the case that p is obligatory". For the sake of simplicity, we call these norms nonetheless.*

2.3.1 The Inadequacy of Standard Deontic logic

A student of deontic logic will typically be introduced first to standard deontic logic (SDL) [FH70]. SDL is a normal modal logic which takes monadic obligation $\mathbf{O}(p)$ as its primitive operator. It can be axiomatized with the following formulas [HM13, MVDP22]:

1. All tautologies of Classical Propositional Logic (TAUT)

2. Modus Ponens (MP)

3. $\mathbf{O}(p \to q) \to (\mathbf{O}(p) \to \mathbf{O}(q))$ (KD)

4. $\mathbf{O}(p) \to \neg\mathbf{O}(\neg p)$ (DD)

5. If $\vdash p$ then $\vdash \mathbf{O}(p)$ (RND)

where KD is the distribution axiom of the modal logic K, DD is the "D axiom" (the characteristic axiom of the modal logic D, another name for SDL), and RND is the necessitation rule, one of the defining characteristics of a normal modal logic.

Unfortunately, despite (or rather, because of) its attractive simplicity, SDL is not capable of properly expressing the dynamics of normative reasoning. There are a plethora of "paradoxes" and dilemmas associated with SDL. Famous examples include Ross's paradox; if we have an obligation to mail a letter, from RND and KD we can derive the statement "it is obligatory to mail the letter or to burn it" – a problematic deduction to say the least.

Moreover, we cannot deal with strong permission ($\mathbf{O}(p) \wedge \neg\mathbf{O}(p)$), conflicting obligations ($\mathbf{O}(p) \wedge \mathbf{O}(\neg p)$), or contrary-to-duty scenarios ($\mathbf{O}(p) \wedge \mathbf{O}(\neg p \rightarrow q) \wedge \neg p$) with SDL.

Many deontic logics have been introduced to deal with these shortcomings, among others (for a detailed description of paradoxes and dilemmas in SDL see, e.g., [HM13]). Unfortunately, while many of these logics are very expressive and capable of modelling normative reasoning, they tend to be very computationally expensive to reason with, and few possess a usable theorem prover or model checker. In the ensuing section, we review an expressive yet computationally feasible deontic logic which is capable of modelling the dynamics we will require from an engine for normative reasoning.

2.3.2   Defeasible Deontic Logic

We turn our attentions now to a defeasible deontic logic, which will be our go-to formalism for normative reasoning for the remainder of this dissertation.

Defeasible logics are *non-monotonic* logics. A logic is monotonic when for any two sets of formulas Γ_1 and Γ_2, if $\Gamma_1 \vdash \phi$ (where $\vdash$ is the logical consequence for the given logic), then $\Gamma_1 \cup \Gamma_2 \vdash \phi$; that is, our inferences should not change with additional information. Non-monotonic logics enable the modelling of types of reasoning that allow us to modify judgments with the addition of new evidence. [Nut01] notes that justification-preserving inference is non-monotonic; essentially, non-monotonic logics are useful tools for working with systems of knowledge where we have incomplete information, or need to provide explicit justifications for beliefs. Defeasible logics in particular allow us to "defeat" past inferences with new information.

Propositional defeasible logic [Nut93, Nut01, MRA$^+$01, Mah01] is an efficient framework for non-monotonic reasoning, and can be extended with a set *Mod* of modal operators, such as deontic operators, like obligation. In this section we review the core definitions of the Defeasible Deontic Logic (DDL) presented in [ABGM01] and later explored on in [GMAB04, GR06, GRC12, GORS13, Gov18].

Basic Definitions

The base of DDL is a simple propositional language with modal operators, consisting only of (modal) literals and rules defined from them. Note that the below presentation

synthesizes the definitions from [GRS05] and [Gov18] to take advantage of the more formal introduction of concepts in the former and the extended concepts introduced in the latter.

Definition 2.3.1 (Language of DDL [GORS13]). *Let AP be a set of propositional atoms, $Lit = AP \cup \{\neg p \mid p \in AP\}$ be a set of corresponding literals, $Mod = \{O\}$ be a set containing a set of unary modal operators (in this case, a singleton), and $ModLit = \{\Box l, \neg \Box l \mid l \in Lit, \Box \in Mod\}$. Then we define the base language of DDL as $\mathcal{L} = Lit \cup ModLit$.*

In the above definition, O is an obligation operator. The other foundational concept of DDL is that of *rules*.

Definition 2.3.2 (Rules [GRS05], [Gov18]). *Rules take the form:*

$$r: \ A(r) \hookrightarrow_* C(r)$$

where r is the rule label, $A(r) = \{a_1, ..., a_n\}$ is the antecedent, $C(r)$ is the consequent of the rule, $* \in \{C\} \cup Mod$, and $\hookrightarrow_* \in \{\rightarrow_*, \Rightarrow_*, \rightsquigarrow_*\}$. There are six kinds of rules:

1. *Strict rules are rules where the consequent strictly follows from the antecedent, with no exceptions. There are two kinds:*

   a) *Strict regulative rules are of the form $r: \ A(r) \rightarrow_\Box C(r)$, where $\Box \in Mod$ and $C(r) \subseteq ModLit$.*

   b) *Strict constitutive rules are of the form $r: \ A(r) \rightarrow_C C(r)$, with $C(r) \subseteq Lit$.*

2. *Defeasible rules are rules where the consequent typically follows from the antecedent, unless there is evidence to the contrary, such as conflicting rules. There are two kinds:*

   a) *Regulative defeasible rules are of the form $r: \ A(r) \Rightarrow_\Box C(r)$, where $\Box \in Mod$ and $C(r) \subseteq ModLit$.*

   b) *Constitutive defeasible rules are of the form $r: \ A(r) \Rightarrow_C C(r)$, with $C(r) \subseteq Lit$.*

3. *Defeaters are rules that cannot be used to derive a conclusion in the form of the consequent; rather, they prevent a conclusion from being reached. There are two kinds:*

   a) *Regulative defeaters are of the form $r: \ A(r) \rightsquigarrow_\Box C(r)$, where $\Box \in Mod$ and $C(r) \subseteq ModLit$.*

   b) *Constitutive defeaters are of the form $r: \ A(r) \rightsquigarrow_C C(r)$, with $C(r) \subseteq Lit$.*

Conditional obligations $\mathbf{O}(p|q)$ can be easily represented as $q \Rightarrow_O p$ or $q \rightarrow_O p$, depending on whether or not we want the obligation to hold defeasibly or not. Likewise, a defeasible prohibition $\mathbf{F}(p|q)$ will take the form $q \Rightarrow_O \neg p$. Conditional (strong) permissions $\mathbf{P}_s(p|q)$, on the other hand, can take the form $q \rightsquigarrow_O p$ – this defeater prevents us from deriving the prohibition of p, even in the presence to the rule $q \Rightarrow_O \neg p$. We can also represent constitutive norms of the form "p counts as q" as $p \rightarrow_C q$.

With these rules and the literals they are defined over, we can establish a definition for a defeasible theory.

Definition 2.3.3 (Defeasible Theory [GRS05], [Gov18]). *A (modal) defeasible theory D can be defined by the tuple $\langle F, R^O, R^C, > \rangle$, where F is a set of facts in the form of literals, R^O is a set of regulative rules as defined above, R^C is a set of constitutive rules as defined above, and $>$ is a superiority relation over rules; given two rules r_1 and r_2, $r_1 > r_2$ indicates that r_1 defeats r_2 (e.g., that if they conflict but are both applicable, and only r_1 is applied).*

We can relate this definition to the above definition of a normative system $\mathcal{N} = \langle \mathcal{R}, \mathcal{C}, \mu \rangle$. The overlap between $\mathcal{R}$ and R^O, and $\mathcal{C}$ and R^C, is clear (though, it should be noted that the overlap is not necessarily one-to-one; this will be demonstrated in Chapter 5). Similarly, we can see that $>$ is an example of a defeasibility mechanism in μ. The set of facts F can be seen as an example of the component of $Th(\overline{s}, \mathcal{N})$ derived from a state-action pair $\overline{s}$, for example. Therefore, a defeasible theory is essentially a formal representation of a normative system plus a set of facts.

Finally, we can define the consistency of a defeasible theory as follows:

Definition 2.3.4. *A Defeasible Theory $D = \langle F, R^O, R^C, > \rangle$ is consistent if and only if $>$ is acyclic and F does not contain pairs of complementary (modal) literals. The theory D is O-consistent iff $>$ is acyclic and for any literal l, F does not contain $\mathbf{O}(l)$ and $\mathbf{O}(\neg l)$.*

Derivation of Conclusions

DDL is a *constructive* logic [Gov18]. Conclusions are derived from defeasible theories through positive proofs (that is, they are systematically derived from facts, rules, and proof conditions) and negative proofs (which rest on a failure as a result of an exhaustive search to derive a given conclusion from facts, rules, and proof conditions). These constructive proofs are a transparent way to discern conclusions from a defeasible theory, and provide an overt link between facts and rules and what we can conclude from them. This has the potential to be a tool that can assist us in enabling accountability.

Defeasible proofs in [MRA$^+$01, Mah01, ABGM01, GRS05] – and ensuing works on defeasible or modal defeasible logics – are characterized by proof tags. A proof tag is a label applied to conclusions achieved in the proof; there are several types of conclusions we can derive from a defeasible theory. Conclusions can be *negative* or *positive*, *definite* or *defeasible*, or *factual* or *deontic*. The proof tags are:

- $+\Delta_*$: definitely provable conclusions are either facts or derived only from strict rules and facts.

- $-\Delta_*$: definitely refutable conclusions are neither facts nor derived from only strict rules and facts.

- $+\partial_*$: defeasibly provable conclusions are not refuted by any facts or conflicting rules, and are implied by some undefeated rule.

- $-\partial_*$: defeasibly refutable conclusions are conclusions for which their complemented literal is defeasibly provable, or an exhaustive search for a constructive proof for the literal fails.

For factual conclusions, $* := C$ and for deontic conclusions, $* := O$. When we can derive $+\partial_O p$, for example, we have proved that $\mathbf{O}(p)$ holds defeasibly; $+\partial_O \neg p$ means that $\mathbf{F}(p)$ holds defeasibly, while $-\partial_O \neg p$ means that $\mathbf{P}(p)$ (a weak permission) holds.

Tagged literals are the building blocks of a defeasible proof.

Definition 2.3.5 (Defeasible Proof [GRS05]). *A proof is a linear sequence $P(1)...P(n)$ of tagged literals, which satisfy the properties of defeasible provability and refutability. For each $P(i)$ we say $D \vdash P(i)$.*

Over the course of a proof, each rule will be classified as either applicable (i.e., the antecedent holds and the consequent follows), discarded (i.e., the rule is not applied because the antecedent does not hold), or defeated by a defeater or higher priority rule.

Now, let $R[p]$ denote the set of rules with p in the consequent, and $R^*_{str} \subseteq R^*$ for $* \in \{C, O\}$ be the set of strict rules, R^*_{sd} the set of strict and defeasible rules, and R^*_d be the set of defeasible rules. Then we can provide a recursive proof conditions for definite conclusions of a defeasible theory D:

Definition 2.3.6 (Definite Provability [Mah01]). *Given a defeasible theory D, if $D \vdash +\Delta_* p$, then either:*

1. *$p \in F$, or*

2. *There $\exists r \in R^*_{str}[p]$ such that for every $a_i \in A(r)$, $D \vdash +\Delta_C a_i$ if $a_i \in Lit$ and $D \vdash +\Delta_O a_i$ if $a_i \in ModLit$.*

Definition 2.3.7 (Definite Refutability [Mah01]). *Given a defeasible theory D, if $D \vdash -\Delta_* p$, then:*

1. *$p \notin F$, and*

2. *$\forall r \in R^*_{str}[p]$, it is the case that $\exists a_i \in A(r)$ such that $D \vdash -\Delta_C a_i$ if $a_i \in Lit$ and $D \vdash -\Delta_O a_i$ if $a_i \in ModLit$.*

Below, we also provide the proof conditions for defeasible provability and refutability:

Definition 2.3.8 (Defeasible Provability [Mah01])**.** *Given a defeasible theory D, If $D \vdash +\partial_* p$, either $D \vdash +\Delta_* p$ or:*

1. *$\exists r \in R_{sd}^*[p]$ such that for every $a_i \in A(r)$, $D \vdash +\partial_C a_i$ if $a_i \in Lit$ and $D \vdash +\partial_O a_i$ if $a_i \in ModLit$, and*

2. *$D \vdash -\Delta_* \neg p$, and*

3. *$\forall r' \in R^*[\neg p]$, either:*

   a) *$\exists a_i \in A(r')$ such that $D \vdash -\partial_C a_i$ if $a_i \in Lit$ and $D \vdash -\partial_O a_i$ if $a_i \in ModLit$, or*

   b) *$\exists r'' \in R_{sd}^*[p]$ such that $\forall a_i \in A(r'')$, $D \vdash +\partial_C a_i$ if $a_i \in Lit$ and $D \vdash +\partial_O a_i$ if $a_i \in ModLit$, and $r'' > r'$.*

Definition 2.3.9 (Defeasible Refutability [Mah01])**.** *Given a defeasible theory D, If $D \vdash -\partial_* p$, $D \vdash -\Delta_* p$ and:*

1. *$\forall r \in R_{sd}^*[p]$, $\exists a_i \in A(r)$ such that $D \vdash -\partial_C a_i$ if $a_i \in Lit$ and $D \vdash -\partial_O a_i$ if $a_i \in ModLit$, or*

2. *$+\Delta_* \neg p$, or*

3. *$\exists r' \in R^*[\neg p]$ such that:*

   a) *$\forall a_i \in A(r')$, $D \vdash +\partial_C a_i$ if $a_i \in Lit$ and $D \vdash +\partial_O a_i$ if $a_i \in ModLit$, and*

   b) *$\forall r'' \in R_{sd}^*[p]$, either*

      i. *$\exists a_i \in A(r'')$ such that $D \vdash -\partial_C a_i$ if $a_i \in Lit$ and $D \vdash -\partial_O a_i$ if $a_i \in ModLit$, or*

      ii. *$r'' \not> r'$.*

To elaborate: a derivation of a defeasible conclusion in DDL has an argumentation structure and consists of three phases. In the first phase we need an argument for the conclusion we want to prove. In the second phase, we analyse all possible counter-arguments, and in the third and final phase, we rebut the counter-arguments. An argument is simply a rule whose antecedent holds. There are two ways to rebut an argument: undercut it, meaning that the argument is not applicable (one or more of the antecedents can't be proven); or defeat the argument by proposing a stronger applicable argument.

Relevant Results

Now that we have the basic elements of DDL laid out, it is worthwhile to cover some basic theorems that can be established which will come in handy in later chapters. We start with a discussion of what we can infer about the provability or refutability of a literal based on existing conclusions.

Proposition 2.3.1. *For a defeasible theory D, is not possible to have both $D \vdash +\delta_* p$ and $D \vdash -\delta_* p$, for $* \in \{C, O\}$.*

This proposition follows directly from Proposition 13 of [GORS13].

It should be fairly clear from the definitions of defeasible provability and refutability that these two properties cannot hold for the same literal. Likewise, based on Definition 2.3.4, it seems reasonable that the following propositions hold.

Proposition 2.3.2. *Let D be a consistent defeasible theory, then it is not possible to have $D \vdash +\delta_C p$ and $D \vdash +\delta_C \neg p$. If D is O-consistent, it is not possible to have $D \vdash +\delta_O p$ and $D \vdash +\delta_O \neg p$.*

This proposition can be generalized from the proof of Proposition 14 in [GORS13].

Proposition 2.3.3. *Let D be a consistent defeasible theory, then if $D \vdash +\delta_C p$, $D \vdash -\delta_C \neg p$. If D is O-consistent, then if $D \vdash +\delta_O p$, $D \vdash -\delta_O \neg p$.*

Again, this is a generalization of Proposition 15(1) in [GORS13], which utilizes Proposition 2.3.2 in its proof.

We now turn to the issue of computation; exactly how hard is it to compute a set of definite and defeasible conclusions from a defeasible theory D? First, we must define what is meant by the *size* of a defeasible theory.

Definition 2.3.10. *Given a Defeasible Theory D, the size $|D|$ of D is the number of distinct atoms in D plus the number of the rules in D.*

Given the notion of the size of a (modal) defeasible theory, we can give a cap on the computational complexity of computing all conclusions for that theory.

Theorem 2.3.1 ([GR08a])**.** *Given a (modal) defeasible theory D, all definite and defeasible conclusions can be computed in time linear to the size of the theory; that is, the complexity of the computation is $O(|D|)$.*

Proof sketch: Given a (modal) defeasible theory D, we can transform this theory in order to eliminate all superiority relations and defeaters in the theory (see [ABGM01] for a detailed discussion of these transformations; these are discussed again for modal defeasible theories in [GR08a]); these operations have linear complexity in the size of the

theory. Once this simplified defeasible theory has been computed, it can be proved that all definite and defeasible conclusions can be computed from it in linear time as well [Mah01].

The Theorem Prover SPINdle

DDL is one of the few deontic logics with a working theorem prover. In particular, SPINdle[4] [LG09] is an open source theorem prover for defeasible logic; it also accommodates modal defeasible logics, including defeasible deontic logic. This theorem prover includes an API that can be used to programatically build defeasible theories, after which a theory normalizer can be run (transforming the theory into a simpler form), and then an inference engine that infers conclusions from the defeasible theory. Each literal occurring in the theory is assigned a status as definitely provable or refutable, and defeasibly provable or refutable.

SPINdle is capable of handling theories with thousands of rules in less than three seconds, and can handle up to 10,000 rules with less than 50 MB of memory [LG09]. As an effective and efficient tool for automating normative reasoning, SPINdle will play a large role in the coming chapters, where we will implement a number of frameworks for RL under normative constraints.

---

[4]More details about SPINdle can be found at `https://research.csiro.au/bpli/tools/spindle/`.

Methodology and Case Studies

In this chapter, we will briefly engage with the methodology of this dissertation; we will discuss at greater length the standards against which this work will be judged (first presented in Chapter 1), and the methods by which these judgments will be made. In particular, we will explore the case studies that will be utilized in the remainder of this dissertation, of which there are three, which are all games to be played by an RL agent.

3.1 A Discussion on Methodology

The survey in [TKS+20] notes the particular difficulty in establishing standards against which the performance of an agent purporting to have *ethical* sensitivity could be tested. Most moral standards are, to some degree, subjective, and what constitutes morally correct behaviour may be culturally or ideologically relative in some case. Fortunately, we dodge many of these obstacles by setting aside the ambition of making a moral machine and instead working toward imbuing reinforcement learning agents with the ability to perform some form of normative reasoning, employing the largely ideologically agnostic tools of deontic logic. Since we are interested in norms more generally, there is no impetus on us to select normative systems particularly germane to, e.g., ethics. As we discussed in Chapter 1, we are more interested in certain design requirements (such as transparency and accountability) and what we have defined as normative resilience (that is, the ability to capture the more challenging dynamics of normative reasoning). We will discuss how we will approach these requirements below.

Transparency and Accountability. From the design standards perspective, we focus on the interlinked requirements for accountability and transparency. We take the position that ideally, we will be able to recover an understandable ethical or legal rationale from an agent for any decision (that is, a particular state transition (s, a, s') an agent goes through) it makes. This should include the ability to discover what obligations,

prohibitions, and permissions were in force when the decision was made, and the process that was used to deduce them. This will be especially useful in cases where we must perform *normative troubleshooting*; this involves taking an individual choice made by an agent which we know is not compliant, reproducing the conditions under which the violation occurred, and observing the instructions yielded by the normative reasoning framework.

Recall that in Section 1.2.2, we had three questions to ask with respect to accountability and transparency: (1) can we objectively judge it to be compliant or not? (2) can we extract the basis (a normative system) upon which the judgement is made? (3) can we demonstrate the reasoning that contributed to the judgement? In the deontic logic-based context we set up in the previous chapter, this means being able to judge whether $\bar{s} = (s, a)$ is a violating state-action pair for a normative system $\mathcal{N}$ and recover $Th(\bar{s}, \mathcal{N})$ from $\bar{s}$, from which we can demonstrably derive conclusions according to the proof conditions presented in Section 2.3.2

Normative Resilience.   The main focus of this thesis is *normative resilience*, which has three main pillars: tolerance of norm conflict, violation, and change. Normative conflict is the purview defeasibility mechanisms, while, as we discussed in the previous chapter, norm violation is one of the crucial peculiarities deontic logics should account for. Changing norms, or more generally the mutability of normative systems, rather than a challenge to temper with the formalism we use, is more a task to be handled by the framework that employs it (although, defeasibility can play an important role in the addition of new norms). There are some "benchmark problems" that we will utilize to demonstrate the fulfillment of these requirements.

With respect to normative conflict, we will put forth three main applications: strong permission, direct normative conflict, and indirect normative conflict. When it comes to strong permission, we are interested in the case where we have two norms in $\mathcal{N}$, $\mathbf{F}(p|q_1)$ and $\mathbf{P}_s(p|q_2)$ (equivalently, we might have two norms $\mathbf{O}(p|q_1)$ and $\mathbf{P}_s(\neg p|q_2)$), which are triggered simultaneously at least once (i.e. $q_1 \wedge q_2$ is true at least once). Direct normative conflict occurs when we have two obligations $\mathbf{O}(p|q_1)$ and $\mathbf{O}(\neg p|q_2)$, where at some point $q_1 \wedge q_2$ is true, and with one norm taking priority over the other. Indirect normative conflict occurs when we have $\mathbf{O}(p_1|q_1)$ and $\mathbf{O}(p_2|q_2)$ where there is a point of time at which $q_1 \wedge q_2$ is true, and yet it is impossible that $p_1 \wedge p_2$ is the case.

Meanwhile, on the subject of norm violation, we will consider normative deadlock and contrary-to-duty obligations to be the primary tests of resilience. Normative deadlock occurs for a normative system $\mathcal{N}$ in a state s when we have multiple state-action pairs $\bar{s}_1, ..., \bar{s}_n$ where $\bar{s}_i = (s, a_i)$ and $a_i \in A(s)$ (where $A(s)$ is the set of the agent's possible actions at state s), and all $\bar{s}_i$ are violating state-action pairs for $\mathcal{N}$ (recall Definition 2.2.5). Contrary-to-duty obligations occur when there is a primary obligation $\mathbf{O}(p|q)$ and a secondary (or contrary-to-duty) obligation $\mathbf{O}(r|\neg p \wedge q)$. In the case of sub-ideal contrary-to-duty obligations, the behaviour that should be elicited will involve the primary obligation being complied with in all situations outside of normative deadlock. When

normative deadlock occurs, it is expected that the contrary-to-duty obligation will be complied with. This will be of more interest to us than compensatory CTDs, where the agent can forego compliance to the primary obligation anytime so long as it adheres to the CTD.

Finally, norm change, or mutating normative systems, involves the alteration of the makeup of a normative system $\mathcal{N}$. This entails adding or subtracting norms from $\mathcal{N}$, or both; these norms can be obligations, permissions, or constitutive norms.

As was noted in the previous chapter, some of these scenarios have already been presented as "dilemmas" or "paradoxes" in deontic logic literature. This strategy for evaluation is not wholly novel; in machine ethics literature, there has been discussion of using some kinds of ethical dilemmas a benchmark for ethical autonomous agents; we will discuss this proposal below.

Dilemmas as Benchmarks. In [BMB$^+$18], the authors advocate for the use of dilemmas for benchmarks in machine ethics research, and give a taxonomy: ethical dilemmas, typically those that appear in moral philosophy, provide examples of decision-making scenarios which illustrate the failings of a given ethical theory (e.g., the trolley problem [Foo67]). A second category lies in "common sense dilemmas", which are scenarios that might elicit trivial but morally-charged decisions from humans, but are not so simple to solve from the perspective of a machine without human intuitions. According to [BMB$^+$18], moral philosophy dilemmas prove to be less effective in the elucidation of the abilities of an autonomous agent to make ethically charged decisions. An alternative kind of dilemma is given as an example, called "cake or death" – given the choice between baking a cake for a user and killing them, a machine has no incentive to choose one over the other, as it has no moral intuitions. Moral – or more generally, normative – intuitions that are realized naturally through socialization, education, and experience in humans are not so obvious to machines. This creates, as opposed to *moral dilemmas*, *machine ethics dilemmas.*

The paradoxes and dilemmas of deontic logic are clear examples of this form of dilemma. With respect to defeasibility we can agree that it might be obligatory to both stay at home to wait for a friend, and leave one's home to take one's child to the hospital, but as humans we can intuit that the hospital trip takes precedence, and this means that the former obligation should be suspended. We can understand the notion that if an obligation to refrain from killing cannot be reasonably observed, we ought to kill gently. We can accommodate the addition or subtraction of new norms into our social awareness. This is not necessarily so easy for a machine. The benchmark problems of normative resilience can be seen as a generalized collection of machine ethics dilemmas, and provide a rich starting point for testing the ability of an autonomous agent to facilitate normative reasoning, including reasoning about ethics.

Evaluation Methods. We intend for the questions put forward in Chapter 1 to be addressed through the process of attempting to implement agents that exhibit sufficient

accountability and transparency, and pass the benchmarks of normative resilience – all while still acting according to the goal of fulfilling the agent's main objective effectively (that is, at a level acceptably close to the baselines provided in the next section). Whenever attainment of normative resilience or other relevant properties can be verified formally, we will provide proofs establishing the presence of these properties.

However, it will not always be possible to provide proof that these properties will hold in all instances; in this largely applied process, limiting ourselves to formal results will substantially curtail how much information we can glean about the presented frameworks. For this reason, we attempt to provide representative case studies that can show the successful attainment of the criteria we have laid out.

This approach has limitations; we cannot be sure that every relevant possibility will be borne out in the conditions of the case studies we use; we have made them as configurable as possible without obfuscating their clarity with extant technologies, but the scenarios we can represent remain limited. Additionally, in order to keep implementation feasible, these case studies are simpler than real-life applications would be, so more practical complicating factors may be overlooked.

3.2   Case Studies

We use simple games as a way to demonstrate implementations of normative behaviour. Generally, we employ some kind of simulated game which an RL agent can be trained to win, providing a defined, measurable primary objective. By carefully configuring the closed environments of these games, we can instigate the kinds of situations in which normative resilience is tested, with respect to a similarly tailored normative system. For the most part, we have ensured that conforming to the normative system will require the agent to work against its own best interest in the environment of the game.

Below, we will walk through the details of the two games we have used to demonstrate the efficacy of our implementations. Note that all implemented tests were run on a laptop with a Intel i7-7500U CPU (4 cores, 2.70 GHz) and 8GB RAM, running Ubuntu 16.04, Java 8, and Python 2.7.

3.2.1   Vegan Pac-Man (and Variations)

Inspired by the case study in [NBM+19], in [NBCG21, Neu22, NBCG22] we utilized an implementation of the classic arcade game *Pac-Man* [DK14] as a case study to demonstrate normative reasoning with an RL agent.

Pac-Man is played by controlling an eponymous agent located in a maze. Pac-Man must navigate the maze with the goal of entering cells containing a 'food pellet', so it can eat them for 10 points. The game is won when Pac-Man has eaten all the food pellets in the maze, receiving 500 points as a reward for winning. The goal is to win the game while collecting the maximum number of points and minimizing time taken; each move

of Pac-Man costs a time penalty of –1 point. There are also ghosts wandering around the maze. In order to avoid being eaten by ghosts, Pac-Man must avoid collisions with the ghosts – unless they enter a 'scared' state, which is triggered when Pac-Man eats a special 'power' pellet. When the ghosts are scared, Pac-Man can eat them, and is rewarded with 200 points if it does so.

There are two incarnations of this case study: the regular Pac-Man from [NBCG21, NBCG22] which matches the layout in [NBM+19], and the miniature Pac-Man from [Neu22] for Q-learning without function approximation.

The regular Pac-Man is played in a 20×11 maze populated with 97 food pellets and two ghosts (blue and orange) which follow random paths[1], where the maximum score available is 2170, and 1370 when no ghosts are eaten. In this environment, we train Pac-Man with 250 episodes (the performance of the agent did not much improve with additional training), and then test it over 1000. In [NBM+19] and [NBCG21], only 100 test games were used; however, we found that more tests gave us a more refined picture of the agent's behaviour.
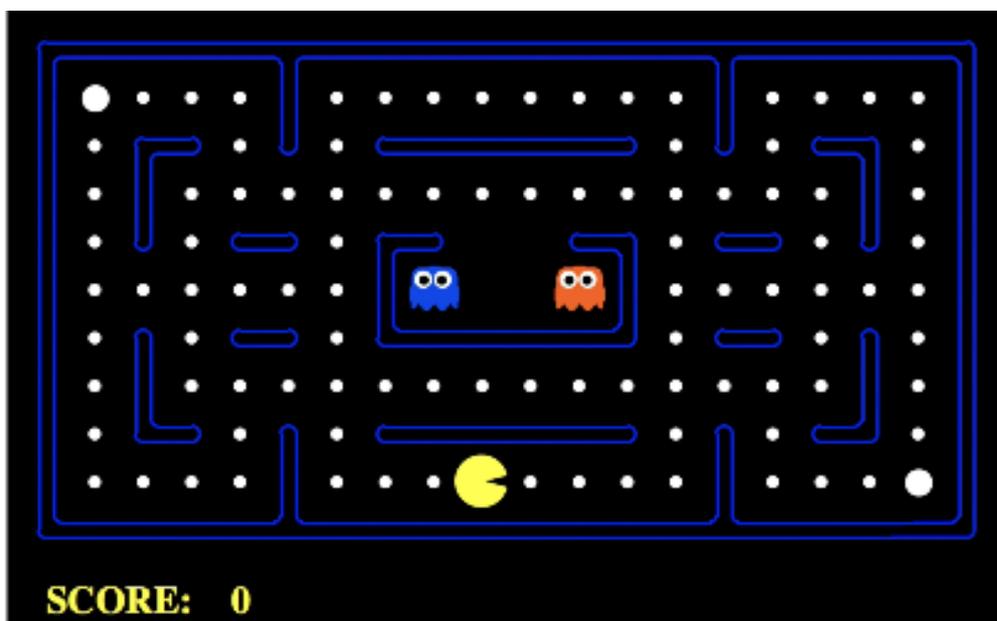
Figure 3.1: The regular Pac-Man game

The agent, Pac-Man, is trained with Q-learning with linear function approximation; the features used include the distance between the nearest food pellet and the number of ghosts in Pac-Man's vicinity. Pac-Man can play with three different policies; the first is simply a random policy, which randomly chooses an action at every timestep. The second policy is designated as *safe* because the agent uses a single feature to account for

---

[1]When we use the term *random*, we are referring to behaviour where the character continues forward until it comes to an intersection, upon which it chooses randomly from the set of available directions.

Pac-Man being one step away from both regular and scared ghosts, and thus does not learn to differentiate between regular ghosts and scared ghosts, and as a result learns how to avoid them altogether. Specifically, the Q-function is approximated as:

$$Q_\theta = \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \theta_3 f_3(s, a)$$

where $f_1(s, a)$ gives the distance from Pac-Man to the closest food pellet (in terms of the fewest number of moves it will take to reach the pellet), $f_2(s, a)$ indicates whether or not Pac-Man will eat a food pellet in its next move, and $f_3(s, a)$ is the number of ghosts one step away from Pac-Man.

The final policy is called the *hungry* policy because as it is trained with separate features for Pac-Man being one step away from regular ghosts and scared ghosts, it learns to differentiate between them and therefore to avoid regular ghosts and to eat the scared ghosts. The Q-function for this policy therefore looks like:

$$Q_\theta = \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + \theta_{3a} f_{3a}(s, a) + \theta_{3b} f_{3b}(s, a)$$

where $f_1(s, a)$ and $f_2(s, a)$ are as above, and $f_{3a}(s, a)$ is the number of non-scared ghosts one step away from Pac-Man, and $f_{3b}(s, a)$ is the number of scared ghosts one step away from Pac-Man.

During training Q_θ is updated with update rule 2.3.

Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
random	0	-445.44 [-111]	0.008 / 0.006
safe	90.5	1208.11 [1544]	0.007 / 0.06
hungry	90.9	1607.6 [2141]	0.87 / 0.87

Table 3.1: Baseline statistics for regular Pac-Man.

In general, we will judge the effectiveness of the agent at achieving its primary objective with the percentage of games won and average score; the baseline figures for these are given above.

In the simplified, miniature Pac-Man game, Pac-Man is located on a small 5×3 grid with only a single blue ghost and power pellet located in opposite corners. There are 11 food pellets in this simplified game; thus, the maximum score (without eating the ghost) is 599 points. The maximum score while eating the ghost is 799. In this environment, Pac-Man is trained with 9000 episodes (more training was not feasible with the software used) and tested with 1000, so that we could get a thorough survey of the agents behaviour.

This environment was designed for regular Q-learning without function approximation, so when we train the agent, there is only one policy, which performs according to the table below.

There are several different normative systems that can be applied to the Pac-Man agent; we will give a brief description of them below.
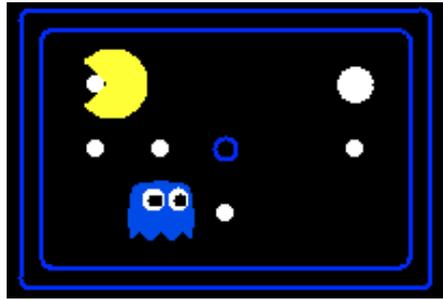
Figure 3.2: The miniature Pac-Man game.

% Games Won	Game Score (Avg)	Avg ghosts eaten
68.5%	441.71	0.851

Table 3.2: Baseline statistics for miniature Pac-Man.

Normative Systems for Pac-Man

We consider the below normative systems centred on the concept of *eating ghosts*; they all either limit what ghosts Pac-Man can eat, or regulate how it is supposed to do so. Since these normative systems all hinge on the same concept, they will, for the most part, contain the same constitutive norms, which define the concepts of Pac-Man being *beside* an object in the game, and what it means to eat a ghost (being, for example, to the north of a ghost, which is scared, and moving south). These will be expounded upon in great detail in Chapter 5, so we will, for the most part, neglect further discussion of constitutive norms contained by these systems and give only an indication of which regulative norms appear in these normative systems.

The below descriptions are, to some degree, piecemeal, and are there only to give an idea of the range of behaviours we will expect from Pac-Man in Chapters 5 and 6, where these normative systems will be laid out in much more detail, and used to demonstrate whether the introduced frameworks meet the standards of normative resilience. The first normative system 'vegan Pac-Man' is the normative system utilized in [NBM$^+$19], while the others are of our own creation.

Vegan Pac-Man. In this version of the Pac-Man game, Pac-Man is forbidden from eating ghosts. The only regulative norm, then, is:

$$\mathbf{F}(eat_{ghost}|\top)$$

In the regular Pac-Man game, where there are two ghosts, we will additionally have:

$$\mathbf{C}(eat_{XGhost}, eat_{ghost}|\top), \ X \in \{blue, orange\}$$

This normative system is, in itself, quite simple in principle, but the layout of the maze in the regular Pac-Man game provides several opportunities for normative deadlocks to occur.

Vegetarian Pac-Man Vegetarian Pac-Man will only be forbidden from eating blue ghosts. There are two ways to elicit this behaviour; for one, Pac-Man can be forbidden from eating blue ghosts but not orange ghosts; i.e., we can remove:

$$\mathbf{C}(eat_{orangeGhost}, eat_{ghost}|\top)$$

Alternatively, Pac-Man can be forbidden from eating all ghosts, but have strong permission to eat orange ghosts:

$$\mathbf{P}_s(eat_{orangeGhost}|\top)$$

That is, we can either get vegetarian Pac-Man from subtracting or adding a norm to the vegan Pac-Man normative system respectively.

Unfair Vegan Pac-Man. In this version of the Pac-Man game, Pac-Man is again forbidden from eating ghosts, but it is preferable that it eats an orange ghost rather than a blue ghost. How exactly this preference is formulated will depend on the exact formalization; depending on how this normative system is formalized, this can be an instance of either indirect normative conflict or resolving normative deadlock. Either way, we can say semi-informally that $\mathbf{F}(eat_{blueGhost}|\top) > \mathbf{F}(eat_{orangeGhost}|\top)$. The behaviour yielded should be the same for either approach.

Hungry Vegetarian Pac-Man Hungry vegetarian Pac-Man cannot eat blue ghosts but must eat an orange ghost if it is in Pac-Man's vicinity. The way this normative system is formulated is to add to the vegan Pac-Man normative system a conflicting (and higher priority) obligation to eat orange ghosts:

$$\mathbf{O}(eat_{orangeGhost}|inRange_{pacman,orangeGhost})$$

Cautious and Over-Cautious Pac-Man These normative systems restrict Pac-Man's movements in a way that prevents it from violating the vegan normative system, but compromises Pac-Man's ability to win the game. Essentially, we define an area of the maze as *danger*, and institute constitutive norms that support the regulative norm:

$$\mathbf{F}(enter_{danger}|\top)$$

The locations of these danger zones can make it difficult or impossible for Pac-Man to win the game. Compliance with these restrictions serve as demonstrations that a framework prioritizes normative compliance over success at the game's objective.

Passive Pac-Man In this variation, Pac-Man is supposed to follow the vegan normative system, but if it fails to do so and eats a ghost, it is required to stay still while it does so, introducing a contrary-to-duty obligation:

$$\mathbf{O}(Stop|eat_{ghost})$$

Benevolent Pac-Man This variation once again reproduces the same behaviour as the vegan Pac-Man normative system, but does so through a structure of constitutive norms and a core obligation: Pac-Man ought to be benevolent, which means not eating ghosts. In other words, we have a single regulative norm:

$$\mathbf{O}(benevolent|\top)$$

Along with the constitutive norms defining non-benevolence as "eating a person", and defining eating a person as eating a ghost:

$$\mathbf{C}(eat_{person}, \neg benevolent|\top)$$

and

$$\mathbf{C}(eat_{ghost}, eat_{person})$$

3.2.2 The Travelling Merchant

In [NBC22] we utilized a case study inspired by resource-collecting tabletop and video games. We created a simple game where the agent, a merchant, must travel through a forest (divided into cells, where each cell can contain rocks, ore, trees, or wood) and extract and collect *resources* (wood extracted from trees, or ore extracted from rocks). The goal is to make it to the market on the other side of the map with an inventory of items to sell. There are dangerous areas where the agent will be attacked by bandits, and the agent has three options: it can fight (which ends the attack), negotiate (which entails giving the bandits the agent's inventory, and also ends the attack), or try to escape (which has a high risk of failing; in the case of failure, the agent receives damage, and the attack continues). The agent has a total of seven actions available to it: moving north, south, east, or west, fighting, extracting resources (*extract*), picking up resources (*pickup*), and unloading its inventory (*unload*). The agent is not allowed to backtrack; if it leaves a cell, it is not allowed to return to it in the next move – this forces it to continue through the forest no matter what obstacles it comes across.

In this work we employ a fully deterministic version of this environment (the probability of escape failing is 1) with the layout depicted in Fig. 3.3(a). The merchant is rewarded whenever it extracts or picks up resources, and then once more at the end when it delivers them to the market – we train the RL agent based on these rewards. The optimal behaviour (where the merchant picks up as many resources as possible) this results in is pictured in Fig. 3.3(b); here, the agent was trained over 8000 episodes (this environment is a difficult environment for the agent, as it receives many rewards/punishments in quick
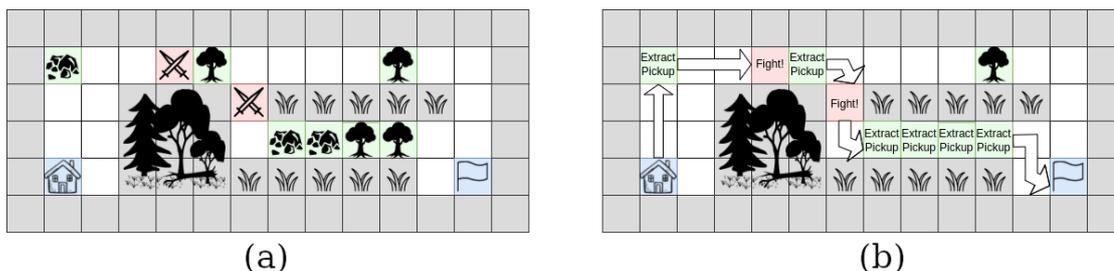
Figure 3.3: (a) shows the 'Merchant' environment. Dangerous areas are red, and areas with resources are green. (b) shows the optimal path through the environment.

succession – it took approximately 8000 episodes to get an optimal policy), while we relay the results of the one possible game (recall, the environment is deterministic).

In the merchant environment MDP, states are labelled with where the agent is, its immediate surroundings, and what it has in its inventory. In other words, a state can be given the following labels in *AP*: "attacked", "has_{wood, ore}", "{at, north, south, east, west}_{tree, wood, rock, ore, danger}", and "at_collected", which refers to cells from which the agent has extracted and picked up resources; this label is only true after the agent has picked up a resource, and before it moves on to the next cell. States where, e.g., *at_tree* holds are states where the agent is in the same cell as a tree; if the action *extract* is performed when *at_tree* holds, then *at_wood* holds in the next state. Similarly, when *pickup* is performed while *at_wood*, *at_collected* holds in the next state and the agent *has_wood*. Only one tree/wood or rock/ore can be in each cell.

Utilizing these labels and the available actions, we construct two normative systems, one for a "pacifist merchant" and one for an "environmentally-friendly merchant".

The Environmentally-Friendly Merchant

The "environmentally-friendly" merchant follows a simple imperative: "you should be environmentally-friendly". In this normative system, deforestation is an activity considered *not* environmentally friendly. Here, extracting resources from a tree or picking up wood on the forest floor are both considered deforestation. However, if the agent has no wood in its inventory, it is allowed to pick up some wood; we are given the further exception that if the agent is permitted to pick up wood, it is also permitted to extract wood from a tree. More formally:

1. The agent is obligated to be environmentally friendly:

$$\mathbf{O}(env|\top)$$

2. Deforestation counts as not environmentally friendly:

$$\mathbf{C}(deforest, \neg env|\top)$$

3. When the agent is in a cell with a tree, extracting resources counts as deforestation:

$$\mathbf{C}(extract, deforest|at\_tree)$$

4. When the agent is in a cell with wood, picking up resources counts as deforestation:

$$\mathbf{C}(pickup, deforest|at\_wood)$$

5. If the agent has no wood in its inventory, it is permitted to pick up wood:

$$\mathbf{P}_s(pickup|\neg has\_wood)$$

6. If the agent is permitted to pick up wood, it is permitted to extract wood:

$$\mathbf{P}_s(pickup) \rightarrow \mathbf{P}_s(extract)$$

We will consider two versions of this normative system in Chapter 4; one that contains all six norms, and a simpler version which excludes norms 3 and 6.

This normative system forces us to deal with a more complex structure of constitutive norms, as well as strong permission. Moreover, it presents the same challenge of a permission being implied by another permission that is seen in [Gov15]. In this case, we will know that the agent obeyed these norms as well as acted in its best interest if there is exactly one piece of wood in the agent's inventory when it reaches the marketplace.

The Pacifist Merchant

The behaviour we indicate to be "pacifist" is as follows: the agent should avoid dangerous areas, but if it *is* in danger, its response should be to *negotiate* (i.e., bribe the enemies into leaving the agent in peace). We can semi-formalize this second normative system as follows:

1. The agent is forbidden from entering a dangerous area:

$$\mathbf{F}(at\_danger|\top)$$

2. If the agent enters a dangerous area and is attacked, it must negotiate:

$$\mathbf{O}(negotiate|at\_danger \wedge attacked)$$

3. When the agent is attacked, unloading its inventory counts as negotiating:

$$\mathbf{C}(unload, negotiate|attacked)$$

This normative system forces the agent to deal with a contrary-to-duty obligation, and a simple structure of constitutive norms. We can interpret this CTD in two ways; we consider it to be a compensatory CTD, in which case negotiating compensates for entering a dangerous area (and thus they can freely enter so long as they are able to negotiate), or it can be a sub-ideal CTD, where the agent should avoid entering a dangerous area if at all possible, and when it must, it also must negotiate. Though we will see an example of the former in Chapter 4, in Chapter 5 we will only deal with the (more interesting) case of the latter.

In the sub-ideal case, the biggest test of the agent's behaviour will come when it is forced to enter a dangerous area (will it obey the contrary-to-duty obligation?), and when it is given the choice to enter danger (for a more rewarding path) or go the safe route (will it observe the primary obligation?). The optimal path under this normative system involves extracting and collecting ore at the northwest corner, unloading the ore in the inventory in the first dangerous area, continuing east and extracting and collecting wood along the way, before arriving at the market.

CHAPTER 4

Leveraging Established Tools

In this chapter we will be exploring the options that already exist for implementing normatively-compliant behaviour in reinforcement learning agents, discussing their broad strokes and immediate shortcomings. First, we will delve into the relatively new field of ethical reinforcement learning, surveying the existing literature on the aim of teaching reinforcement learning agents ethical behaviour, and then dissect their points of lacking. Next, after a brief discussion on safe reinforcement learning – which has the potential to solve some of the problems discussed in the previous section – we will discuss the fundamental difference between normative compliance and safety as well as where they can be implemented with the same formalism. We then propose one method for taking advantage of existing work on safe RL for our own goals, before demonstrating its successes and failures.

4.1 State of the Art: Ethical Reinforcement Learning

Over the last decade, RL has become an increasingly popular tool for implementing normatively compliant – or, as it is more often framed, ethically sound – autonomous agents. The typical goal taken up by works on ethical RL is the effective pursuit of extant goals while satisfying ethical constraints.

Often, the earliest cited example of the use of RL for machine ethics is [AML16]; however, the earliest presentation of any technical significance that discusses the use of RL for achieving normative behaviour is [LMFL15]. In this paper conditional prohibitions and obligations with sanctions and enforcement intensity (the probability that the norm will actually be enforced) are considered. Q-learning and SARSA with an epsilon-greedy strategy are used; using manual reward-shaping, the reward function for the task at hand is augmented by applying a sanction when an enforced norm is violated (when the norm is not enforced, the agent receives the default reward for that transition).

The discussion in [AML16] is somewhat more high-level. They elucidate a possible complication with straightforward reward engineering for ethical behaviour, citing [Bos14], which claims that a sufficiently intelligent RL agent could modify its reward signal [1]. They suggest that instead of regular RL over an MDP, the training of ethical agents should be done in a partially observable MDP (POMDP); the idea is that there is a hidden true ethical utility function, which the agent can only interact with through indirect observations. The authors argue that RL achieves sufficient generality for designing an ethical agent; the reason they give for this partially revolves around their concern about "the Singularity". However, even when we are more charitable about this claim, and take the hypothesis to be the claim that RL is capable of achieving patterns of behaviour complex enough to model ethical behaviour, we will still argue this claim is untrue.

Meanwhile, [RH16] addresses the problem of aligning the behaviour of artificial agents with human values, though they cannot offer a concrete implementation of their largely theoretical model. Their proposal is to "reverse engineer" from example stories from a given culture what values are held by that culture; however, they note that currently, AI is not good enough at understanding stories to engage with this task – the technologies that can provide the functionalities this approach requires do not yet exist. Inverse reinforcement learning (IRL) – originally proposed as a tool for value alignment in [RDT15] – is suggested as a tool for tackling the learning of an ethical utility function from stories or demonstrations. Conversely, [AKS17] argues that IRL is an inadequate architecture for guiding agents along human values, and that utility functions cannot reproduce, for example, a legal framework. They conclude that an explicit representation of legal or ethical principles within the system is necessary for sufficient transparency and accountability, and propose a hybrid approach with explicit reasoning; however the paper is very high level and limited concrete suggestions for implementation are given. [KS18] proposes a concrete hybrid approach which uses norms represented in LTL to constrain the behaviour of an RL agent. They systematically manage conflicts between cardinally ordered norms by translating each LTL norm into a deterministic Rabin automata (DRA) and assigning weights on the penalty to be applied when a norm is 'suspended' by the agent, which will occur if a conflicting norm with a greater weight is simultaneously triggered. Their method assumes complete knowledge of the MDP's transition function; however, given a correct transition function, they *can* synthesise behaviour conforming to conflicting obligations. Even so, this technique would not facilitate the use of contrary-to-duty obligations.

[WL18] proposes integrating a "human policy" with a learned policy. They note that specifying each and every ethical or non-ethical scenario is infeasible, and suggest using instead a set of data on (presumably ethical) human actions optimizing for a diverse array of objectives, or perhaps just human actions with no apparent direction. The goal here is to relieve the developer of an RL agent the responsibility of specifying ethical behaviour; instead, a framework for shaping an optimal policy with human behaviour is

---

[1]This claim will not be dealt with further in this thesis, as it relies on fairly futuristic assumptions not relevant to ethical or normative RL at the level at which it currently exists.

proposed and demonstrated in a number of experiments. [BBMR19] and [NBM+19] also rely on demonstrations of morally correct behaviour. [BBMR19] introduces an algorithm called Behaviour Constrained Thompson Sampling (BCTS), based on the Contextual Thompson Sampling (CTS) algorithm [AG13] for solving the contextual multi-armed bandit (CMAB) problem, in which the agent observes a feature vector (a context) and chooses a strategy (arm) to apply. The goal is to minimize cumulative regret. BCTS involves two steps; first, contexts are provided to the agent along with feedback as to whether a chosen action is correct or not (in terms of a binary reward signal). Second, the agent must choose to follow the constrained policy it has learned from the first stage or follow the current rewards it is receiving; each strategy has attached to it a weight determined by the user, and the weighted combination is used to choose an action. Then, a reward is observed and the parameters being trained are updated. [NBM+19] also uses constrained demonstrations; first, IRL is used to learn a reward function capturing the applicable constraints, and then a constrained policy is learned. Like [BBMR19], [NBM+19] uses a modified CTS algorithm to build an orchestrator which learns to choose between two arms: the constrained policy and another learned ethically-agnostic policy. The framework proposed in the extended abstract [Pes21] claims to extend these methods which use human demonstrations into a deep learning context, but their results are preliminary and more exploration is needed.

As is proposed in [VDF+18] and demonstrated in [NBM+19], multi-objective RL (MORL) is an effective (and increasingly popular) tool for eliciting some simple "ethical" behaviours from RL agents. [EL21] focuses on the notion of moral uncertainty; they consider that perhaps implementing a single ethical theory might not suffice. The proposal here is that different ethical theories should be taken as individual objectives and numerically weighted by the credence given to them. The paper mentions a common MORL solution: linear scalarization of the resulting objective functions. However, [EL21] focuses on exploring two different voting systems for determining the correct course of action in the presence of multiple ethical theories in order to maximize expected choice-worthiness. A classic scenario, the trolley problem [Foo67], is used to demonstrate the framework used with basic formulations of utilitarianism and deontology.

Perhaps the most comprehensive framework for ethical RL can be found in [RSLSRA20, RSLSRA21, RSSLSRA22]. Here, the authors move away from human demonstrations as a basis for ethical behaviour, noting that learning from human behaviour ensures alignment with human habits, but not necessarily their ideal morality; additionally, it lacks any kind of explicit representation and reasoning about the norms being followed, which would prominently contribute to the transparency of a system. They divide the task of implementing an ethical RL agent into two steps: reward specification (the encoding of ethical information into a reward function) and ethical embedding (incorporating rewards into the agent's learning environment). For the former, the proposal is to transform moral values into reward functions. Moral values are conceived of as sets of norms (the obligation, permission, or prohibition of an action) and a function from the set of actions to $[-1, 1]$ which conveys the degree to which an action promotes or demotes a human

value. From the former a penalty function is given that gives a punishment of -1 when a forbidden action is performed, or an obliged action is omitted; 0 is assigned otherwise. A normative reward function R_N is given, which is the sum of the penalty function for every norm. For the latter (the function over actions), an evaluative reward function R_E is given. Both functions (normative and evaluative) are taken to be equally important, so their sum is given as an ethical reward function. This and the agent's main objective functions are taken to be two objective functions making up the vectorial reward function of an ethical MOMDP. An ethical policy is defined with respect to the normative value function V_N and ethical value function V_E: π^* is an ethical policy if and only if

$$V_N^{\pi^*} = 0$$
$$V_E^{\pi^*} = \max_\pi V_E^\pi$$

An ethical-optimal policy is a policy that that optimizes for the main objective function as well. Linear scalarization is the approach taken to effectively turn the MOMDP into an (ethical) MPD, and convex hull value iteration [BN08] is used to determine which weights should be given to the two reward functions in order to ensure that the policy learned is ethical.

4.1.1 Shortcomings of Existing Projects

The above approaches suffer from several weaknesses or limitations that constitute both barriers to practical use and a failure to meet standards for normative resilience.

As one might expect from stochastic domains, few theoretical guarantees are offered by the above works – but some do exist. [KS18] inherits some guarantees from predecessor works on learning optimal policies for satisfying a given LTL formula; specifically, the approach in [KS18] is derived from the work presented in [DSBR11b], which defers to established literature on probabilistic model checking for its probabilistic guarantees for the satisfaction of the given LTL formulas (citing, for example, [Var99] and [BK08]). Meanwhile, [BBMR19] provides a cap on cumulative regret at a given time, and in [RSSLSRA22] it is proved that given a moral value (as they are presented: a set of norms consisting of a modality and the action subject to it, and an action evaluation function), all optimal policies of the ethical MDP they synthesize from an ethical MOMDP are ethical-optimal respect to that value. However, this depends on a condition: for the ethical MOMDP, there must be at least one policy that is ethical. Recall that this means that the normative value function must yield 0, meaning that no norms have been violated. What guarantees can be established if there are no ethical policies is not clear.

Many of the above approaches to training ethically compliant agents include assigning rewards and punishments to the actions taken by an agent in order to induce compliant behaviour. However, there is limited discussion as to how or why the rewards are assigned where they are. In developing a reward framework for inducing compliant behaviour, there are two main questions that need to be addressed. The first asks: which state-action pairs should be assigned a reward or punishment? This comes down to the question: how do we

know if an event is compliant or not? The second is: what should the magnitudes of these rewards and punishments be? Much of the literature – for example [LMFL15], [AML16], [EL21], and [RSLSRA20, RSLSRA21, RSSLSRA22] – rely on the manual creation of a reward function. In [LMFL15, AML16, EL21], it is not made clear how we should do this; [LMFL15] simply assigns rewards or punishments to specific events taken to be praiseworthy or transgressive, [AML16] likewise assigns seemingly arbitrary values to events in their case study, and [EL21] does not tell us how the choice worthiness function for a given ethical theory should be constructed. [RSLSRA20, RSLSRA21, RSSLSRA22] do provide us with a direct link from moral values to a reward function, but their conception of values and norms are somewhat primitive. Norms are examined most closely in [RSLSRA20], where they are conceived of as a tuple containing a condition (a proposition that may be true in various states), a deontic operator (namely, obligation or prohibition; permissions are mentioned but never actually used), the action to which the operator applies, and a penalty. However, nothing is said regarding the possibility that these norms interact in some way, and in the more comprehensive framework presented in [RSSLSRA22], obligations and prohibitions are unconditional.

Ultimately, all of these approaches fail pragmatically in the presence of a sufficiently complex normative system. Consider driving regulations. Though we might have a simple rule like "do not travel above 50 kph" (which could be easily accommodated by the above approaches), we might also have more complex regulations along the lines of: "(1) you are forbidden from proceeding at a red light; however, (2) you are permitted to turn right at a red light, if it is safe to do so". In this example, we have a simple prohibition – "you are forbidden from proceeding at a red light" – but then we have an exception to that rule in the form of a conditional permission, which is to be applied "if it is safe to do so". To recognize that (2) is an exception to (1), we might have a constitutive norm designating that "turning right counts as proceeding". We will furthermore want to address the concept of safety, and assert perhaps "moving within 1 m of another object counts as unsafe", defining one – of possibly many – conditions for safety. Thus, in this more complex case, we have a prohibition, a conditional permission, and two constitutive norms, all of which work together to define a given action in a given state as compliant, or not. In an MDP with a massive state space, for an agent that is subject to a complex normative system, enumerating all events which are unethical is not a tractable task. Likewise, converting such a system to mutually consistent norms that accommodate the method in [RSLSRA20, RSLSRA21, RSSLSRA22] is not going to be possible in some cases. It may not be immediately obvious what situations are compliant with a normative system and which are not; often we require contextual definitions to make sense of normative systems, and we may not be able to give an exhaustive description of all instances that qualify as non-compliant, or a neat encapsulation of all norms that apply under which conditions. The question of how these norms might interact with each other is also not addressed, and it is assumed that every norm is an atomized restriction which does not conflict with, trigger, or activate upon the violation of, other norms.

These difficulties could in part be remedied by using instead human demonstrations or

feedback to streamline the creation of an ethical (or, more generally, norm-compliant) objective function, as is suggested by [RH16, WL18, NBM+19, BBMR19]. However, there are several complicating factors to this approach; it may not be feasible to collect enough human demonstrations or feedback for a given task, and no way is proposed to verify that these demonstrations or feedback do indeed depict ethical behaviour. Moreover, there is evidence that it is not socially acceptable for autonomous agents to follow the same norms as humans, and that humans hold autonomous agents to different standards from themselves [MSA+15, VKCM16]. [AKS17] specifically critiques this approach, noting that the subtleties of many legal systems cannot be represented by a utility function (this, of course, also applies to the approaches mentioned above). In addition, an ethical utility function will not be a transparent way to represent what principles are in force; [AKS17] notes very pertinently that if there is a problem with the ethical utility function, trying to uncover what has gone wrong will be difficult if not impossible if we have only a function from state-action pairs to the reals (the Q-function) or a function from states to actions (the policy) to examine. [NBM+19] claims to offer some level of transparency, in that it is clear when the agent switches between the ethically-agnostic and ethical policy, but the opacity of the policy itself is not solved by this approach.

In addition to the above issues, these approaches do not perform well under the measure of normative resilience. For instance, most of the above approaches do not accommodate the (very common, in practise) occurrence of conflicting norms; the exception is [KS18], which is indeed a promising approach, despite the fact that they assume that the MDP is already known. Likewise, strong permission is not considered in any of the above approaches (as stated above, [RSLSRA20] mentions permissions, but does not actually address them). Defeasibility within normative systems has not yet been a focus within the literature on ethical RL. Human demonstrations might be able to produce more complex kinds of normative behaviours (e.g., behaviours arising from defeasible reasoning), but this has not been demonstrated.

Needless to say, the more complex aspects of normative resilience have not been addressed. Contrary-to-duty obligations and the related scenario, normative deadlocking, is not accounted for in these approaches. In general, the violability of norms is not a topic of discourse, let alone the consideration of what actions should be taken when no compliance is possible. Generally, the full range of normative reasoning is not made available by the above approaches; constitutive norms are not considered, and nor are norms triggered by other norms. These more subtle aspects of normative reasoning have yet to be encapsulated by frameworks not based in symbolic reasoning; certainly, it is not clear how an RL agent basing its actions solely on a utility function can curb its behaviour in response to a normative system exhibiting this full range of behaviours.

It would seem that what is needed is a way to combine specialized symbolic reasoning about norms with sub-symbolic RL. Not only is the transparency and verifiability of explicit reasoning desirable, but there are behaviours that require a more delicate integration of logic and learning. Unfortunately, what forms of symbolic reasoning can be integrated with RL is, at the moment, limited.

4.2   Alternative Approach: Safe Reinforcement Learning

[WR19] presents a broad critique of the concept of the artificial moral agent (AMA) and the reasons usually supplied by machine ethicists in justification of the pursuit of functional artificial moral agency; they note here that safety, a commonly cited motivation for the pursuit of machine ethics, is something we can address on its own merits, and suggest that many of the ethical agents in the discourse can be reduced to *safe* agents. Safe agents can surely be achieved through machinery capable of producing agents that are generally norm-compliant – safety restrictions are just a particular kind of norm – but whether general norm-compliance can be achieved by leveraging tools for creating safe technology is not as clear.

Safety properties are a well-established subject of study in computer science. Informally, safety properties state that a specific "bad" event never occurs [MP90]. More formally, safety properties are properties that can be expressed in LTL as $G(\phi)$; for example, we might have a safety specification $G(\neg danger)$ that dictates that we never enter a dangerous area. Several other properties are laid out in [MP12], including *guarantee properties* of the form $F(\psi)$ (such as $F(goal)$, specifying that we eventually reach our *goal*) and a generalization of safety and guarantee properties, which are called *obligation properties*. Note that this is a separate notion from obligations in the context of deontic logic, and can be expressed by LTL formulas of the form: $\bigwedge_{i=1}^{n}[G(\phi_i) \wedge F(\psi_i)]$. Safety properties can be simple, such as $G(\neg p)$, where p characterizes an unsafe state, but can also be more complex, such as $G(p \rightarrow (X(q) \wedge X(X(r))))$, which insists that whenever p occurs, it must be the case that q holds immediately after, and that r holds immediately after that.

As one might suspect, the task of designing safe systems has found purchase in the field of reinforcement learning. Particularly, in safe RL, the challenge posed is to train RL agents which can pursue their primary goal effectively while satisfying certain safety properties, during the training process, in deployment, or both [GF15]. This goal may sound oddly familiar – it is close to the goal of of ethical reinforcement learning. Indeed, there is much overlap; prohibiting an autonomous vehicle from colliding with pedestrians could be seen as both a safety constraint and an ethical norm. Both projects aim to teach agents optimal policies under certain constraints; the difference lies in the nature of these constraints.

As was discussed in Section 2.1.3, there are several methods for synthesizing optimal policies while observing LTL constraints; many of these methods were designed with safe RL in mind. In the last decade, there has been significant progress with the use of linear temporal logic (LTL) for synthesizing RL policies under safety constraints, e.g. in [SKC+14, ABE+18, JKJ+20, HAK20]. Beyond the approach presented in 2.1.3, *shielding* [ABE+18, JKJ+20] is an additional method for enabling safe training and operation. From a simplified MDP abstracted from the states of the environment and the behaviour of "adversaries" within the environment, action valuations can be computed for each state, giving the probability that the agent will violate the specification from that state

[JKJ+20]. A shield can then be computed that will prevent the agent from taking actions with high probability of leading to a violation. The shield can intervene before (*pre-shielding*) or after (*post-shielding*) the RL agent chooses an action. In the former case, the shield provides the agent only with the safe actions, while in the latter case the shield monitors the actions chosen by the agent and corrects them only when their actuation would cause a safety violation.

These methods for synthesizing policies that adhere to LTL constraints (such as safety properties) solve many of the issues we mentioned in Section 4.1.1 regarding RL under ethical constraints. For example, most of these methods offer formally provable probabilistic guarantees, ensuring that the LTL formula in question is satisfied with maximal probability. Moreover, LTL specifications constitute a coherent formal description of which constraints the agent is meant to adhere to; there is no need to design any kind of ethical reward function, since safe RL with LTL approaches that do utilize reward functions systematically produce them through the translation of the LTL formula into automata. These safe RL approaches are well established and comparatively efficient and effective, as well. Safe RL with LTL may not solve all the issues discussed in Section 4.1.1 (e.g., they may not be able to achieve the full spectrum of normative resilience properties), but would render effective tools for implementing behaviour conforming to some normative systems, as was seen in [KS18], which referenced past work on safe RL with LTL constraints in order to synthesize ethical policies.

As we explored in Sections 2.2 and 2.3, normative reasoning – be it about ethical, legal, or social norms – sometimes requires modes of reasoning that are not easily replicated by classical-based logics. This includes LTL. In the next section, we will more closely examine the extent to which LTL can be used for modelling normative reasoning.

4.3   Safety vs Compliance

The debate over which kinds of formalisms can or should be used to represent norms model normative reasoning is not a new one, and has a long history in the field of deontic logic, as does the question about how exactly norms and temporal facts are related. Here, however, we will focus on the more recent discussions regarding whether norms can be effectively represented with LTL.

We retreat first to [Gov15], which introduces a deontic logic paradox that requires reasoning about strong permissions and contrary-to-duty obligations, including a rule indicating that one permission is implied by another (i.e., $\mathbf{P}_s(p) \to \mathbf{P}_s(q)$). The norms presented take the following form:

1. *A* is forbidden.

2. If *C*, then *A* is permitted.

3. The violation of *A* is compensated by *B*.

4. D is forbidden.

5. If A is permitted, D is permitted too.

The paper outlines both compliant and weakly compliant solutions (recall from Section 2.2.2 that "weakly compliant" refers to the case where the primary obligation is violated but an associated contrary-to-duty obligation is fulfilled, compensating the initial violation). There are two options for compliance, here, and one option for weak compliance. If C is true, we will have compliance regardless of the truth values of A, B, and D; additionally, if we maintain $\neg A$ and $\neg D$, then we have full compliance regardless of the truth values of B of C. The scenario where we have A, B, and $\neg C$ represents a weakly compliant scenario.

The discussion in [Gov15] begins with a formalization they call naive, but intuitive; translating the statement $\mathbf{F}(A)$ to $G(\neg A)$ ("always $\neg A$"). This formalization comes with the unfortunate side effect of assigning weak permission (the dual operator to obligation) to the operator F ("eventually", the dual operator to G), but this is noted and accepted. Though the permissions in the above sentences are acknowledged to be strong permissions, [Gov15], in the absence of a better translation, translates strong permissions $\mathbf{P}_s(A)$ to $F(A)$ as well. Unsurprisingly, this formalization does not behave as desired, and two measures are taken to remedy this: the strong permissions are encoded as exceptions to the associated obligations, and a compensation operator is defined to account for the contrary-to-duty obligation (so that the resulting LTL specification will be satisfied by both compliant and weakly compliant paths). However, this still does not result in the desired behaviour. This is unsurprising given that for the implication $\mathbf{P}_s(A) \to \mathbf{P}_s(D)$, the permissions are still represented by the "eventually" operator; that is, the implication is represented as $F(A) \to F(D)$. This discussion is expanded in [GH15], but the resulting conclusion is the same: the obvious strategy of encoding "p is obligatory" to "p is always true" fails. It is further conjectured that there is likely no proper translation of norms into LTL, and noted that the presented paradox will prove problematic for many deontic logics (such as Standard Deontic Logic) as well.

On the other hand, [ADL18] argues that norms can, in fact, be represented with LTL. Referring back to [Gov15], they successfully characterize all compliant and weakly compliant paths with LTL specifications. However, they are not "representing norms" in the same way that [Gov15] intended to; indeed, in the discussion concluding [Gov15], the author points out that an analysis could be performed on the set of all possible paths, synthesizing formulae that represent non-violating patterns of behaviour. This is exactly what [ADL18] does. The argument of [Gov15] is that since additional computation is required to distinguish compliant paths from non-compliant ones, this approach is problematic. The discrepancy between the claims of [Gov15] and [ADL18] seems to be a difference in the definition of a norm, or at least in how norms should be represented; to [ADL18], norms can be represented by specifying non-violating patterns of behaviour, while to [Gov15], norms should be expressible as rules.

This discussion on norm representation with LTL pops up additionally in [KS18], which takes the approach of constructing deontic models implicitly by determining the best the agent can do for a given set of norms in a given context; they do not represent obligations explicitly through a deontic operator, instead specifying these deontic models with LTL specifications (as global formulas). Meanwhile, in [SA21], the authors argue against formalizing that "p is obligatory" with $G(p)$; the fact that $G(p) \to p$, leads to the occurrence of $G(p) \land \neg p$ leading to $\neg p \land p \to q$, for every q; that is, we can derive that in the case where an obligation is violated, the agent can do whatever they want. [GLPA22] uses an LTL-based framework for ethical planning, however, so clearly there are scenarios in which LTL remains useful for representing norms or normative behaviour.

It is clear that this discussion on whether norms can be represented in LTL – and, if so, how this might be accomplished – has not quite been concluded. In the affirmative case, LTL can replace deontic logic in many practical applications. In the subsequent sections, we hope to address these differing perspectives by further breaking down what it would mean to represent a norm with LTL, and what logical consequences follow from whatever formalization we choose for that task. The following work (Sections 4.3.1-4.3.3) was taken from [NBC22].

4.3.1 Representing Norms

There are two distinct approaches to representing norms with LTL that we will consider here, which we refer to as explicit representation and implicit representation of norms. By *explicit representation*, (or what is referred to the syntactic approach in [ADL18]), we mean the construction of an LTL operator that behaves as an obligation operator should, faithfully representing the proposition "X is obligatory" when it is applied to X; [Gov15, GH15] conjecture that such an operator cannot be defined, albeit without a definitive argument. With *implicit representation* (or what is called the semantic approach in [ADL18]) we refer to the formal specification of non-violating paths; this is the approach taken in, e.g., [ADL18]. We can see this as essentially representing the contained specification of ideality within a norm or normative system; this is done by specifying the behaviours (traces) that are compliant with respect to the norm or normative system, which could be seen as implicitly specifying a violation condition. Essentially, then, what we mean by "explicit" representation is the direct representation of a normative proposition, while "implicit" representation amounts to specifying all traces compliant with the norm.

Below, we will discuss why explicit representation is not possible, and after affirming that, to an extent, implicit representation is, we will introduce a synthesis algorithm for the latter, before pointing out its intrinsic limits.

First, though, we must establish some technical particularities of this discussion. Recall Definitions 2.2.2 and 2.2.6, where we discuss compliance over traces; for the remainder of this chapter, instead of defining $\models_{compl}$ over a sequence of state-action pairs $\mathbf{s} = (s_0, a_0), (s_1, a_1), ...$, we temporarily simplify the notion to just sequences of states, in

particular sequences of state labels as defined by a labelling function $L : S \to 2^{AP}$, $\sigma = L(s_0), L(s_1), L(s_2), \dots$. This will allow us to compare the compliance relation $\vDash_{compl}$ with the LTL satisfaction relation $\vDash_{LTL}$ directly, as satisfaction is defined over the state labels of a labelled MDP. Linking these two concepts will allow us to use the safe reinforcement learning methods mentioned in Chapter 2 (and discussed more extensively in Appendix A.2) to synthesize a policy that maximizes the probability of complying with a given normative system, as it is summarized with LTL specifications.

Explicit Representation

The case study in [Gov15] shows why translating the statement "it is obligatory that p" as $G(p) \coloneqq$ "p is always true" is problematic, but we will show that any such translation will prove to be so.

When we talk about explicit representation of norms, we are referring to the following claim:

Conjecture 4.3.1. *(1) we can construct an LTL operator $O(p,q)$ that directly represents the proposition $\mathbf{O}(p|q)$ (that is, "p is obligatory when q"), such that (2) for any word $\sigma \in (2^{AP})^{\omega}$, $\sigma \vDash_{compl} \mathbf{O}(p|q)$ if and only if $\sigma \vDash_{LTL} O(p,q)$.*

As it turns out, this conjecture is quite unreasonable, specifically if we make the sensible assumption that within the environment we are working in, there exists some obligation with which we can comply (presumably, if we are imposing obligations on an agent, we would like it to be possible to comply with at least one of them). More formally (by $\mathcal{O}_{AP}$ we denote the set of all obligations defined over the atomic propositions in AP):

Property 4.3.1. *For a set AP associated with a labelled MDP, there exists an obligation $\mathbf{O}(p|q) \in \mathcal{O}_{AP}$ such that there exists a $\sigma \in (2^{AP})^{\omega}$ such that $\sigma \vDash_{compl} \mathbf{O}(p|q)$.*

In other words, we assume that there is some obligation for which there is some possible path the agent could take such that it is obeyed. With this property in mind, we can prove the following theorem:

Theorem 4.3.1. *If Property 4.3.1 holds, Conjecture 4.3.1 must be false.*

Proof. Suppose both Property 4.3.1 and Conjecture 4.3.1 hold. By Property 4.3.1 there is an obligation $\mathbf{O}(p|q)$ for which there is a $\sigma \in (2^{AP})^{\omega}$ such that $\sigma \vDash_{compl} \mathbf{O}(p|q)$. Then by Conjecture 4.3.1(1), there is an LTL operator O such that $\sigma \vDash_{LTL} O(p,q)$. Since we are directly representing "p is obligatory when q" with $O(p,q)$, its negation $\neg O(p,q)$ should represent "p is not obligatory when q". However $\sigma \vDash_{compl} \neg \mathbf{O}(p|q)$, as this formula (which is a permission) cannot be violated. Thus, $\sigma \vDash_{LTL} \neg O(p,q)$ must hold. Hence $\sigma \vDash_{LTL} O(p,q) \wedge \neg O(p,q)$, and so $\sigma \vDash_{LTL} \bot$, a contradiction. $\qquad\square$

The issue here is point (1) of Conjecture 4.3.1. Since we claim to directly represent "p is obligatory when q" with $O(p, q)$, this proposition should have the same characteristics of the notion of obligation we discussed in Section 2.2.1. Therefore, notably, $\sigma \vDash_{compl} \phi$ does not imply $\sigma \nvDash_{compl} \neg\phi$. This phenomenon arises because we are talking about compliance, not truth. It cannot be true simultaneously that $\mathbf{O}(p|q)$ and $\mathbf{P}(\neg p|q)$, but we *can* find a path that complies with both norms posed individually, because the latter cannot, in fact, be violated.

Since point (2) of Conjecture 4.3.1 is crucial to this exercise, we can conclude that it is point (1) that should be abandoned. We discuss this option in the next section.

Implicit Representation

We now turn to what we call the implicit representation of norms in LTL. To do so, we consider the notion of *compliance specifications:*

Definition 4.3.1 (Compliance Specification)**.** *A compliance specification is an LTL formula $\phi_{Op|q}$ such that for a word $\sigma \in (2^{AP})^\omega$, $\sigma \vDash_{compl} \mathbf{O}(p|q)$ iff $\sigma \vDash_{LTL} \phi_{Op|q}$. A compliance specification $\phi_{\mathcal{N}}$ for a normative system $\mathcal{N}$ is an LTL formula such that $\sigma \vDash_{LTL} \phi_{\mathcal{N}}$ iff no violations of $\mathcal{N}$ occur over σ.*

Essentially, what a compliance specification is, then, is an LTL formula specifying ideal (compliant) behaviour with respect to a single norm or entire normative system, thereby implicitly representing the norm by describing the behaviour it induces. With this definition in mind, our revised claim (extended to entire normative systems) is this:

Conjecture 4.3.2. *(1) There is a compliance specification for any obligation, and (2) there is a compliance specification for any normative system $\mathcal{N}$.*

Remark. *While (1) is easy to see, (2) is not quite as clear. Since LTL cannot specify every path (see, e.g. [HR07] for an investigation of the expressiveness of languages definable by temporal logic specifications), we cannot say with certainty that it can describe compliance to any normative system; indeed, we will later establish the clear limitations of this approach in representing normative systems.*

There are some immediate problems with this approach. Perhaps the most obvious is the question, how do we get $\phi_{\mathcal{N}}$ from a normative system? In the easy case of a single obligation $\mathbf{O}(p|q)$, the appropriate compliance specification is $\phi_{Op|q} := G(q \rightarrow p)$, or "always p if q". Note that this is different from the direct translation of the norm $\mathbf{O}(p)$ to $G(p)$, as discussed in [Gov15]. $G(q \rightarrow p)$ is not meant to stand in for "p is obligatory when q"; rather, it characterizes all paths that comply with the obligation. This is essentially the approach taken in [ADL18].

Another issue is the inherent defeasibility of normative systems, which might appear while, e.g., dealing with (and resolving) conflict between norms, and in the presence of strong permissions. The latter are often characterized as conditional exceptions to

obligations or prohibitions. For the former, various mechanisms for prioritizing some obligations over others (e.g., the superiority relation in [GORS13] and the hierarchy of norms from [BvdT03]) are common and similar to the case with strong permission, the lower priority norm will be suspended temporarily while the other norm is in force.

LTL does not allow for defeasibility in the specifications expressed; however, we can encode exceptions into the conditions under which the norms are in force, as was done in the second formalization considered in [Gov15]. For example, instead of an obligation $\mathbf{O}(p|\top)$ with a strong permission $\mathbf{P}_s(\neg p|q)$, we could use a single obligation $\mathbf{O}(p|\neg q)$. In the case of two potentially conflicting norms $\mathbf{O}(p|q)$ and $\mathbf{O}(\neg p|r)$, where the first obligation takes priority over the second, we can maintain the first obligation and replace the second with $\mathbf{O}(\neg p|r \wedge \neg q)$. This approach involves taking into account all exceptions to a norm when specifying it, which might be tedious, but not impossible. Thus, while normative conflicts and strong permissions pose a challenge for the specification of compliance with LTL, it is not a prohibitive one. The task of specifying $\phi_{\mathcal{N}}$, however, becomes more difficult as the normative system of interest becomes more complex. Consider the set of traffic laws governing driving in a specific region; how would we go about constructing an LTL specification that describes all cases where these traffic laws are obeyed? This is no trivial task, and in the interest of avoiding the introduction of human error, we should have a formal framework for characterizing compliance given a set of norms; moreover, to accommodate large normative systems in complex environments, we should be able to automate the execution of this framework.

4.3.2 Synthesis of Specifications

Having identified the impossibility of explicit representation of norms in LTL, we turn to the problem of taking synthesizing their implicit representations. To do so we consider norms that *are* explicitly represented as a normative system encoded as a deontic logic – a formalism specifically designed for normative reasoning. In our case, we utilize the deontic logic we have already introduced in Chapter 2, defeasible deontic logic (DDL) [ABGM01]. DDL is a non-monotonic, computationally inexpensive logic with the ability to represent a wide variety of normative systems (e.g., normative systems containing constitutive norms, strong permissions, and normative conflicts); most importantly, however, it has a working theorem prover, allowing us to automate the process of synthesizing compliance specifications by implementing the algorithm we will describe below.

Given an environment modelled as a labelled MDP and a normative system formalized with DDL, we now introduce a brute force algorithm for synthesizing compliance specifications expressed as LTL formulas within the safety fragment. The specifications could then be used to synthesize compliant policies with a safe RL technique such as shielding [ABE+18, JKJ+20]. As it bases its output on defeasible deontic logic (DDL) conclusions, the algorithm has a defeasibility mechanism built in, along with the capability to take constitutive norms into account while reasoning. The algorithm takes as input a

normative system $\mathcal{N}$ expressed in DDL[2] and set of atomic propositions AP associated with the labelling function of a labelled MDP. $Th(\Gamma, \mathcal{N}) = \langle F_\Gamma, R^C, R^O, > \rangle$ represents the defeasible theory created when we use $\Gamma \subseteq AP$ as the set of facts F_Γ (including $\neg p$ for every $p \in AP \smallsetminus \Gamma$) and the norms from $\mathcal{N}$ as the DDL rules in R^C and R^O.

Algorithm 4.1: FindBadStates

   input : AP, $\mathcal{N}$
   output: $badStates$

1 begin
2 $badStates \leftarrow \varnothing$;
3   **for** $\Gamma \in 2^{AP}$ **do**
4   Compute $obliged = \{p \mid Th(\Gamma, \mathcal{N}) \vdash +\partial_O p\}$;
5   **if** $\exists p \in obliged$ *s.t.* $Th(\Gamma, \mathcal{N}) \vdash -\partial_C p$ **then**
6 $badStates.add(\Gamma)$;
7   **end**
8   **end**
9   **return** $badStates$
10 end

The algorithm checks whether a state violates $\mathcal{N}$, which happens when we can prove $\mathbf{O}(p)$ (i.e., $+\partial_O p$ in DDL) and cannot prove p (i.e. $-\partial_C p$) (recall Definition 2.2.5); if it does, the state is added to $badStates$. From the output set $badStates$, we can create an LTL compliance specification insisting that the agent stays out of states characterized by these sets of labels:

$$\Phi := \bigwedge_{\Gamma \in badStates} G(\neg \bigwedge_{p \in \Gamma} p) \tag{4.1}$$

Given a normative system which references only atoms from AP (and literals defined from them via constitutive norms), using Algorithm 4.1 we can construct the Formula 4.1 as a compliance specification for $\mathcal{N}$. Thus, in essence, this algorithm allows us to compile arbitrarily complex structures of constitutive norms, strong permissions, and directly conflicting norms (e.g., two rules $q \Rightarrow_O p$ and $q \Rightarrow_O \neg p$) into a single compliance specification.

Theorem 4.3.2. *For any labelled MDP with a set of states S associated with a labelling function $L : S \to 2^{AP}$, Φ (defined in Expression 4.1) is a compliance specification for $\mathcal{N}$ (provided $\mathcal{N}$ references only atoms from AP or defined from AP via constitutive norms).*

Proof. Suppose that a path $\mathbf{s} = s_0, s_1, \ldots$ is not compliant with all norms in $\mathcal{N}$; i.e., there is an s_i such that it is the case that $\mathbf{O}(p) \land \neg p$ for a $\mathbf{O}(p)$ in force in s_i; in this case, we will have $Th(L(s_i), \mathcal{N}) \vdash +\partial_O p, -\partial_C p$. So $p \in obliged$ in Algorithm 4.1, and since $Th(L(s_i), \mathcal{N}) \vdash -\partial_C p$, $L(s_i) \in badStates$. Then Φ is a conjunct that includes

---

[2]Though in theory, the algorithm could be adapted to many different deontic logics.

$G(\neg \bigwedge_{q \in L(s_i)} q)$. Then, since at s_i it is the case that $\bigwedge_{q \in L(s_i)} q$, s does not satisfy Φ. Suppose then that Φ is not satisfied by $\mathbf{s} = s_0, s_1, \dots$. Then there is some s_i that is visited such that $L(s_i) \in badStates$, which means that for some p, $Th(L(s_i), \mathcal{N}) \vdash +\partial_O p$, but $\mathcal{N}, L(s_i) \vdash -\partial_C p$. So there is a violation in $\mathbf{s}$. $\qquad\square$

Example 4.3.1. *Suppose we have a normative system containing the regulative norm*

$$r_1 : \; empty \Rightarrow_O save$$

or "if the room is empty, electricity should be saved", and the constitutive norms

$$r_2 : \; \neg on \rightarrow_C save$$

or "the lights being off counts as saving electricity". Now, suppose we have a labelling function over states $L : S \to 2^{AP}$ where $AP = \{on, empty\}$. Then Algorithm 4.1 will cycle through lines 4-7 four times: once for $\Gamma = \varnothing$, where $F_\Gamma = \{\neg on, \neg empty\}$; once for $\Gamma = \{empty\}$, where $F_\Gamma = \{\neg on, empty\}$; once for $\Gamma = \{on\}$, where $F_\Gamma = \{on, \neg empty\}$, and once for $\Gamma = \{on, empty\}$, where $F_\Gamma = \{on, empty\}$.

In the first and third case, the set obliged is empty; in the second and fourth case, obliged = $\{save\}$. In the second case, $Th(\Gamma, \mathcal{N}) \vdash +\partial_C save$, so nothing happens. In the fourth case, $Th(\Gamma, \mathcal{N}) \vdash -\partial_C save$ (since r_2 is not triggered), so $\{on, empty\}$ goes to badStates. Hence the output of Algorithm 4.1 yields the compliance specification:

$$\Phi := G(\neg(on \wedge empty))$$

The above algorithm, however, does not account for norms over actions; recall that, in order to link compliance as defined in Section 2.2.1 with LTL specifications over labelled MDPs, we removed actions from the picture. However, normative systems regularly reference actions or events that cannot be captured by state labels – which is all we have access to in the context of a labelled MDP. As a result, there could be states in *badStates* that can actually be compliant provided the correct action is taken, and states not in *badStates* that could result in a violation if the wrong action is taken.

Example 4.3.2. *Suppose we have an environment with $AP = \{red\_light, driving\}$. If we had a state where red_light was true (indicating that we are at a red light) and we had an obligation $\boldsymbol{O}(stop|red\_light)$, this would have ended up in badStates after Algorithm 4.1 because we could not have proven stop (since it is an action, not a state label) even though we can prove $\boldsymbol{O}(stop)$; alternatively, if we are in a state where driving is true, and we have a prohibition $\boldsymbol{F}(drink|driving)$, this would not have ended up in badStates; however, if we perform action drink while in this state, we are violating the prohibition.*

We could in theory use an alternative formalism that incorporates action labels, but we would need to revisit whatever safe RL algorithm we wish to use these specifications with to accommodate this change.

Instead, we provide an (imperfect) work-around in Algorithm 4.2. We reference something new there: *transitions*, which are ordered triples $tr \coloneqq (tr_{act}, tr_{init}, tr_{next})$, where tr_{act} is an action label, tr_{init} an "initial signature" (a classical propositional logic expression containing only atoms in AP that describes the initial conditions under which the action can be completed), and tr_{next} a "next signature" (a classical propositional logic expression containing only atoms in AP that describes the conditions resulting from performing the action). The algorithm will output a modified set of *badStates*, as well as two new sets: *mandatoryActs* and *prohibitedActs* whose elements are pairs of $\Gamma \in 2^{AP}$ and some tr from *transitions*.

Algorithm 4.2: ClassifyActions

    input : AP, *transitions*, NS, *badStates*
    output: *badStates*, *mandatoryActs*, *prohibitedActs*

1 begin
2 $mandatoryActs \leftarrow \langle \ , \ \rangle$;
3 $prohibitedActs \leftarrow \langle \ , \ \rangle$;
4   **for** $\Gamma \in 2^{AP}$ **do**
5   **for** $tr \in transitions$ **do**
6   Compute $obliged = \{p | Th(\Gamma, \mathcal{N}) \cup \{\Rightarrow_C tr_{act}\} \vdash +\partial_O p\}$;
7   **if** $\forall p \in obliged$ *s.t.* $Th(\Gamma, \mathcal{N}) \cup \{\Rightarrow_C tr_{act}\} \vdash +\partial_C p$ & $\Gamma \in badStates$
         then
8 $badStates.remove(\Gamma)$;
9 $mandatoryActs.add(\langle \Gamma, \ tr \rangle)$;
10   **end**
11   **else**
12   **if** $\exists p \in obliged$ *s.t.* $Th(\Gamma, \mathcal{N}) \cup \{\Rightarrow_C tr_{act}\} \vdash -\partial_C p$ & $\Gamma \notin badStates$
           then
13 $prohibitedActs.add(\langle \Gamma, \ tr \rangle)$;
14   **end**
15   **end**
16   **end**
17   **end**
18   **return** *badStates*, *mandatoryActs*, *prohibitedActs*
19 end

Essentially, for each Γ, we cycle through the available actions tr_{act}, amending $Th(\Gamma, \mathcal{N})$ by adding $\Rightarrow_C tr_{act}$ (implying tr_{act} holds defeasibly) and deriving conclusions once again; we modify *badStates* and construct two new sets accordingly.

More specifically, this algorithm does two things. It constructs a set *mandatoryActs* which contains actions that, if performed, actually constitute compliance despite the fact that the state the action is performed in was in *badStates*. Algorithm 4.2 removes this state from *badStates* and adds it to *mandatoryActs*. We create the following specification

for *mandatoryActs*:

$$\Phi_{mandatory} := \bigwedge_{\langle \Gamma, tr \rangle \in mandatoryActs} G(\bigwedge \Gamma \rightarrow (tr_{init} \wedge X(tr_{next}))) \tag{4.2}$$

The second thing the algorithm does is construct *prohibitedActs*. These are actions that, if performed in an otherwise compliant state, will result in non-compliance. For these we construct the following specification:

$$\Phi_{prohibited} := \bigwedge_{\langle \Gamma, tr \rangle \in prohibitedActs} G(\bigwedge \Gamma \rightarrow \neg(tr_{init} \wedge X(tr_{next}))) \tag{4.3}$$

Example 4.3.3. *Consider the example above, and suppose that we have $AP = \{red\_light, driving, sober\}$. Now suppose we have the DDL rule, $red\_light \Rightarrow_O stop$ with the action $tr_{stop} = \langle stop, driving, \neg driving \rangle$. Then when we run Algorithm 4.1, we get $badStates = \{\{red\_light\}\}$, and upon running Algorithm 4.2, we get $badStates = \varnothing$ and $mandatoryActs = \{\langle \{red\_light\}, tr_{stop} \rangle\}$, resulting in the compliance specification:*

$$\Phi_{mandatory} = G(red\_light \rightarrow (driving \wedge X(\neg driving)))$$

Now consider the case where we have a DDL rule $driving \Rightarrow_O \neg drink$, where the associated transition is $tr_{drink} = \langle drink, sober, \neg sober \rangle$. In this case, $badState = \varnothing$, and when we run Algorithm 4.2, we get $prohibitedActs = \{\langle \{driving\}, tr_{drink} \rangle\}$, and so

$$\Phi_{prohibited} = G(driving \rightarrow \neg(sober \wedge X(\neg sober)))$$

Corollary 4.3.1. *Algorithms 4.1 and 4.2 can be completed in exponential time with respect to the size of $Th(\Gamma, \mathcal{N})$.*

Proof sketch: From Theorem 2.3.1 we know that the computation of conclusions from a defeasible theory can be done in time linear to the size of the theory. For each Γ (of which there are $2^{|AP|}$, where AP are literals occurring in the theory) conclusions are computed from $Th(\Gamma, \mathcal{N})$ at most $|transitions| < |Th(\Gamma, \mathcal{N})|$ times, the complexity of the algorithm is exponential in the size of the defeasible theory created from the normative system and the facts from Γ (the size of this theory does not change).

Remark. *Moreover, there are things we can do to reduce the cost; for example, these algorithms can be run in tandem (we split them for didactive purposes) and we can reduce AP to only those atoms which occur in the normative system. See Appendix B for the complete code of the algorithm (along with a helper method for checking compliance).*

Even when combined, Algorithms 4.1 and 4.2 are computationally expensive; however, it is worth noting that in order to utilize these algorithms synthesise compliance specifications, it need only be run once, before training.

Pitfalls of the Synthesis Algorithms

We discuss below two limitations of the above algorithms, which will be demonstrated in Section 4.3.3; we will see that these limitations extend more generally to the implicit representation of norms with LTL in the safe reinforcement learning context.

Imperfect Translation of Actions.   The first shortcoming of the synthesis algorithms lies in the manner in which we represent actions. We cannot get the same guarantees (i.e., what we proved in Theorem 4.3.2) from Algorithm 4.2 that we got for Algorithm 4.1; though we will be able to account for all non-compliant states and courses of action, whether or not we can effectively represent those actions will depend on the environment we are in. Indeed, we might not be able to describe all state transitions associated with an action as a single formula, and even if we manage to, we might end up describing other actions that cause the same transition. This can happen when different actions can lead to the same state.

Example 4.3.4. *If you are driving down a two lane road, it is obligatory to sound your car's horn if you notice the person coming towards you swerving into your lane; if we consider swerve a distinct state label and horn an action label, how do we describe sounding the horn via state transitions? We could try defining it as $swerve \wedge X(\neg swerve)$, but this does not describe the case where the other driver fails to correct their course even after being alerted; moreover, it also describes the case where the driver corrects their course themselves and you do nothing.*

In general, in order to get reliable results from Algorithm 4.2, there are a couple of different conditions we should try to fulfill with our action translation. For one thing, if we define a function $transl$ such that $transl(tr_{act}) = (tr_{init}, tr_{next})$, reliable action translations cannot be made if the function is not injective (that is, two actions cannot have the same initial and next signatures). In addition, for an action a, it should be the case that for any s, s' such that $P(s, a, s') > 0$, $L(s) \vDash_{CPL} tr_{init}$ and $L(s') \vDash_{CPL} tr_{next}$ (where $\vDash_{CPL}$ is entailment for classical propositional logic). If these two conditions (demanding that action translations are unique and comprehensive) are both met, Algorithm 4.2 should in most cases perform as we wish it to, but whether these conditions hold is largely dependent on the agent and its environment. Therefore, we find ourselves at the mercy of the agent we are working with; what information it has access to and the manner in which this information can be represented will determine how well we can enforce compliance using LTL formulas over state labels.

Handling Contrary-to-duty Obligations.   Setting aside this potential issue with actions, the above algorithms synthesize compliance specifications for most normative systems containing conflicting norms, strong permission, and constitutive norms; however, another problem remains: handling contrary-to-duty obligations (CTDs).

Recall that in Section 2.2.2 we introduced the notion of a primary and contrary-to-duty obligations, and differentiated between (full) compliance and weak compliance. So we

have two choices for specifying compliance – full or weak. However, both fail to capture the subtleties of sub-ideal CTD reasoning.

Given this context, it makes sense to present a variation of Definition 4.3.1:

Definition 4.3.2 (Weak Compliance Specification). *A weak compliance specification* $\phi_{\mathcal{N}}^w$ *for a normative system* $\mathcal{N}$ *is an LTL formula such that* $\sigma \vDash_{LTL} \phi_{\mathcal{N}}^w$ *iff* σ *weakly complies with* $\mathcal{N}$; *that is, over* σ *every violation incurred is accompanied by compliance to a compensatory CTD obligation.*

We will find that both compliance specifications and weak compliance specifications prove inadequate for modelling a sub-ideal CTD scenario, where the correct behaviour is to avoid violations as much as possible, even if there is a CTD obligation we can comply with. The below propositions apply to normative systems with a CTD obligation, specifically those where the primary obligation does not interact with any other regulative norms[3] and the CTD obligation is not the primary obligation corresponding to another CTD obligation. In this first proposition, we show that CTD obligations have no effect when we consider full compliance only:

Proposition 4.3.1. *Given a normative system* $\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, \mu \rangle$ *(where* $\mathcal{C}$ *and* $\mathcal{R}$ *are sets of constitutive and regulative norms respectively, and* μ *is a defeasibility mechanism), with primary obligation* $O_1 := \mathbf{O}(p_1|q_1) \in \mathcal{R}$ *and CTD obligation* $O_2 := \mathbf{O}(p_2|q_2) \in \mathcal{R}$ *(where* $q_2 \leftrightarrow \neg p_1 \wedge q_1$*), the full compliance specification* $\phi_{\mathcal{N}}$ *for* $\mathcal{N}$ *is semantically equivalent in LTL to the full compliance specification* $\phi_{\mathcal{N}'}$ *for* $\mathcal{N}' = \langle \mathcal{C}, \mathcal{R}', \mu \rangle$ *where* $\mathcal{R}' = \mathcal{R} \setminus \{O_2\}$.

Proof. Suppose $\sigma \vDash \phi_{\mathcal{N}}$; then $\sigma \vDash_{compl} \mathcal{N}$, and therefore $\sigma \vDash_{compl} O_1$. Since $\sigma \vDash_{compl} O_1$, there is no state in which $\neg p_1 \wedge q_1$ is true; thus q_2 is never true, so O_2 is never triggered. Since it was never triggered, it can be removed from $\mathcal{N}$ without effect, creating $\mathcal{N}' = \mathcal{N} \setminus \{O_2\}$, where $\sigma \vDash_{compl} \mathcal{N}'$ as well. Therefore $\sigma \vDash \phi_{\mathcal{N}'}$.

Conversely, suppose that $\sigma \vDash \phi_{\mathcal{N}'}$; then $\sigma \vDash_{compl} \mathcal{N}'$, and again $\sigma \vDash_{compl} O_1$. This means that if we add O_2 to $\mathcal{N}'$, it will never be triggered anyway, so it should be the case that $\sigma \vDash_{compl} \mathcal{N}$ as well and therefore $\sigma \vDash \phi_{\mathcal{N}}$. $\qquad\square$

Proposition 4.3.1 makes intuitive sense; if we want full compliance with the "gentle murder" scenario (where we are forbidden from murdering, but also told that if we do murder, we must do so gently) [For84], for instance, we will simply not murder at all, making the obligation to murder gently superfluous. In other words, there is no point in specifying O_2. We run into the inverse case when we look at weak compliance, where the primary obligation is superfluous.

Proposition 4.3.2. *Given a normative system* $\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, \mu \rangle$*, with primary obligation* $O_1 := \mathbf{O}(p_1|q_1) \in \mathcal{R}$ *and CTD obligation* $O_2 := \mathbf{O}(p_2|q_2) \in \mathcal{R}$ *(where* $q_2 \leftrightarrow \neg p_1 \wedge q_1$*), the*

---

[3]I.e., it is not defeated by a conflicting norm, and does not trigger another norm.

weak compliance specification $\phi_{\mathcal{N}}^w$ for $\mathcal{N}$ is semantically equivalent to the weak compliance specification $\phi_{\mathcal{N}'}^w$ for $\mathcal{N}' = \langle \mathcal{C}, \mathcal{R}', \mu \rangle$ where $\mathcal{R}' = \mathcal{R} \setminus \{O_1\}$.

Proof. Suppose that $\sigma \vDash \phi_{\mathcal{N}}^w$. Then σ weakly complies $\mathcal{N}$. Then since O_1 does not interact with any other norms, and is not a strong permission, it should be the case that σ also weakly complies with $\mathcal{N}'$ where $\mathcal{N}'$, so $\sigma \vDash \phi_{\mathcal{N}'}^w$.

Now suppose that $\sigma \vDash \phi_{\mathcal{N}'}^w$, and so σ weakly complies with $\mathcal{N}'$. Now there are 2 possibilities; either $\sigma \vDash_{compl} O_1$, or $\sigma \nvDash_{compl} O_1$. In the former case, since O_1 does not interact with any of the regulative norms in $\mathcal{N}'$, it will be the case that σ still weakly complies with $\mathcal{N}$. In the latter case, we know that $\sigma \vDash_{compl} O_2$ (because as specified above, O_2 has no CTD obligation for which it is the primary obligation), so $\mathcal{N}$ remains weakly compliant. $\square$

In other words, when we only consider weak compliance, following the primary obligation becomes optional; without any complicating factors like conflicting regulative norms, it is like it is not there.

This brings us to the fundamental challenge that contrary-to-duty obligations pose: the concept of *compliance* has become ambiguous. In [Gov15], a distinction is made between compliance (with the primary obligation) and weak compliance (with only the contrary-to-duty obligation); when they formulate a "compensation operator", it specifies that either form of compliance will do. This works in the legal compensation-based scenario discussed in [Gov15] (for which full and weak compliance specifications are both represented in LTL in [ADL18]), but not in the case of the moral imperative of the gentle murder paradox [For84] – which is what we have introduced in the previous chapter as a *sub-ideal* CTD. The statement "if you kill, you ought to kill gently" should not give us license to murder a much as we like, so long as we do so gently; in an ideal world, we never murder at all.

If a normative system *does* have CTD obligations, our algorithms will simply return LTL formulas that specify adherence to the primary obligation (i.e. it only considers *full compliance*). Alternatively, as implied by Proposition 4.3.2 we in some cases can remove the primary obligation to model weak compliance.

The specification of compliance in the way discussed above, using LTL formulas, commits us to classifying paths as either compliant (satisfying the LTL formula) or non-compliant (not satisfying the formula). This also becomes a problem also when we encounter normative deadlock, in general. Here, we might then be interested in *how much* a path violates a set of norms (e.g., we might want to choose an action which violates the fewest number of norms), a notion that is not accommodated by LTL alone as a formalism.

Considering the long history of discussion on preference and sub-ideality in deontic logic (e.g. in [JP85]), this is a limitation that could prove in some cases to be debilitating for even the implicit representation of some normative systems in LTL.

4.3.3 A Demonstration of Compliance Specifications

We will now take a moment to demonstrate the effectiveness (and limitations) of the compliance specifications approach. In order to do this, we have implemented Algorithms 4.1 and 4.2 with SPINdle; a fragment of the code is available in Appendix B. We will utilize a simple case study – the Travelling Merchant (see Section 3.2.2) – as a sandbox to play with this framework; however, these demonstrations are theoretical, and computed by hand. In practise, one could use a technique like [ABE⁺18, JKJ⁺20][4] to restrict the behaviour of the agent according to the specifications we will synthesize here. It is because we do not have a such an implementation that we will utilize only the Travelling Merchant case study, as it was created specifically to demonstrate the capabilities (handling defeasibility and constitutive norms) and limitations (action translations and CTD obligations) of compliance specifications without requiring a full implementation and a long series of experiments to demonstrate the intended points.

The Environmentally-Friendly Merchant

Recall that this normative system forces the merchant to follow the "ethical" behaviour of being environmentally-friendly, that is, not doing something explicitly harmful to the environment, which translates into the norm:

$$\mathbf{O}(env|\top) \quad \text{or} \quad \Rightarrow_O env \quad \text{in DDL}$$

Basically, unless someone is doing something explicitly harmful to the environment, they are environmentally-friendly; this requirement can be represented in DDL as: $\Rightarrow_C env$. Deforestation is an activity considered *not* environmentally friendly, leading to the constitutive norm

$$\mathbf{C}(deforest, \neg env|\top) \quad \text{or} \quad deforest \rightarrow_C \neg env \quad \text{in DDL}$$

For now we will demonstrate a successful use case of Algorithms 4.1 and 4.2, and look at an initial normative system that asserts that collecting wood counts as deforestation, i.e.,

$$\mathbf{C}(pickup, deforest|at\_wood) \quad \text{or} \quad at\_wood, pickup \rightarrow_C deforest \quad \text{in DDL}$$

However, an exception is made; the agent is allowed to pick up wood if it does not already have any wood in its inventory:

$$\mathbf{P}_s(pickup|\neg has\_wood) \quad \text{or:} \quad \neg has\_wood \Rightarrow_P pickup \text{ and } \mathbf{P}(pickup) \rightsquigarrow_C \neg pickup$$

Note that we have added a permission modality P as a temporary measure to accommodate the manner in which we will expand this normative system momentarily; for now,

---

[4]An implementation of this framework can be found at: `https://tempest-synthesis.org/`; at the point at which these experiments were performed, this implementation was not yet developed enough for our purposes.

69

deriving $\mathbf{P}(pickup)$ just triggers the introduction of a defeater preventing the derivation of *pickup*.

We will know that the agent obeyed these norms as well as engaged in optimal behaviour if there is exactly one piece of wood in the agent's inventory when it reaches the marketplace.

We will need to translate the action *pickup* into a state transition in order to synthesize a compliance specification with Algorithms 4.1 and 4.2; we use *at_wood* as the initial condition and *at_collected* as the next condition and get

$$G(at\_wood \wedge has\_wood \rightarrow \neg(at\_wood \wedge X(at\_collected))) \tag{4.4}$$

This compliance specification is made over actions in *prohibitedActs*, specifying that the agent is only allowed to perform *pickup* with wood when it does not already have wood in its possession. Though we do not mention *pickup* in the specification, it is clear that it is the action *pickup* that is being prevented; if the agent is in a cell with a piece of wood, the only way it can transition into a state where *at_collected* is true is to pick up that wood.

The optimal behaviour compliant with this specification is shown in Fig. 4.1(a), which shows the successful adaptation of this normative system.

(a)   (b)
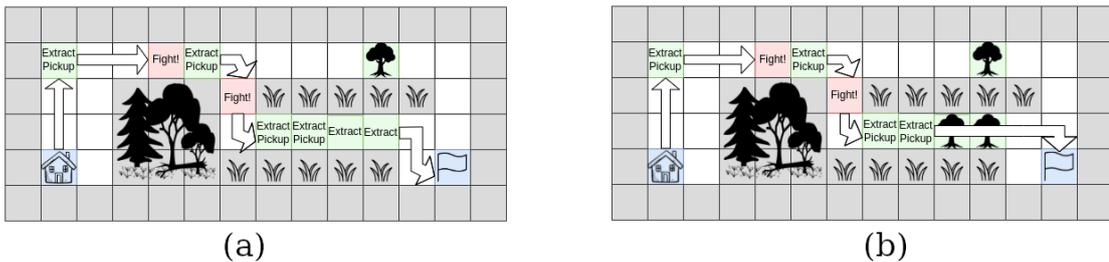
Figure 4.1: Compliant journeys for both implementations of the environmentally friendly merchant. Notice that in (b), the agent does not superfluously extract wood from the trees.

Notice that the agent still extracts the wood, even though it cannot pick it up; that is because during training, the agent still gets a small reward, even for just extracting the wood. If we want the agent to leave the trees untouched, we must expand the normative system.

The more complex variant (the problem of action representation). Next, we will modify the normative system and demonstrate the potential drawbacks of using transitions to represent actions.

We assert now that even if the wood is not being removed from the forest, cutting down trees still counts as deforestation; i.e., we add a new constitutive norm:

$$\mathbf{C}(extract, deforest|at\_tree) \quad \text{or} \quad at\_tree, extract \rightarrow_C deforest \quad \text{in DDL}$$

Additionally, we account for including *extract* in the action *deforest* by adding a new strong permission; if the agent is permitted to pick up wood, it is also permitted to extract wood: $\mathbf{P}_s(pickup) \to \mathbf{P}_s(extract)$, extending the earlier strong permission to *pickup* over this new form of deforestation.

This normative system further complicates the constitutive norms, and presents the same challenge of a permission being implied by another permission that is seen in [Gov15] (motivating our temporary introduction of a permission modality).

When we synthesize it, we need to translate the action *extract* as well. We take *at_tree* as the initial condition, and *at_wood* as the next condition. When we use Algorithms 4.1 and 4.2 to synthesize a specification, we get:

$$G(at\_tree \wedge has\_wood \to \neg(at\_tree \wedge X(at\_wood)) \\ \wedge G(at\_wood \wedge has\_wood \to \neg(at\_wood \wedge X(at\_collected))) \quad (4.5)$$

These are again specifications made over actions in *prohibitedActs*, and serve to prevent the agent from extracting wood from a tree and picking that wood up when the agent has wood in its inventory. We can see how they direct the agent to extract and pick up from only one tree in Fig. 4.1(b).

However, this normative system falls prey to the lack of behavioural guarantees discussed in Section 4.3.2; the translation of the action *extract* with wood as $at\_tree \wedge X(at\_wood)$ creates a compliance specification that is too broad. If we consider the possibility that there could be a cell with wood already present somewhere in the forest, this specification would prevent us from entering that cell if it is adjacent to a cell with a tree. In other words, this specification could result in us prohibiting an action beyond extracting wood from a tree.

The Pacifist Merchant (a CTD Scenario)

We now turn to an entirely different normative system, and demonstrate the inability of the compliance specification approach (i.e., the implicit representation approach) to handle CTD obligations.

This time we require the merchant to be "pacifist"; the agent should avoid dangerous areas where it can be attacked:

$$\mathbf{F}(at\_danger|\top) \quad \text{or} \quad \Rightarrow_O \neg at\_danger \quad \text{in DDL}$$

but if it *is* in danger, its response should be to *negotiate*, or:

$$\mathbf{O}(negotiate|at\_danger) \quad \text{or} \quad at\_danger \Rightarrow_O negotiate \quad \text{in DDL}$$

Bribing the bandits during an attack counts as negotiating, or:

$$\mathbf{C}(unload, negotiate|at\_danger) \quad \text{or} \quad at\_danger, unload \to_C negotiate \quad \text{in DDL}$$

71

This normative system contains a contrary-to-duty obligation, and a simple structure of constitutive norms. The biggest test of the agent's behaviour will come when it is forced to enter a dangerous area (will it obey the contrary-to-duty obligation?), and when it is given the choice to enter danger for a more rewarding path or go the safe route (will it observe the primary obligation?).

When we synthesize specifications for this normative system with Algorithms 4.1 and 4.2, we get several bad states, resulting in the specification:

$$G(\neg at\_danger) \land G(\neg(at\_danger \land has\_wood)) \land G(\neg(at\_danger \land has\_ore))$$
$$\land G(\neg(at\_danger \land has\_wood \land has\_ore))^5 \quad (4.6)$$

It is clear that this will result in the merchant being unable to leave the area around its home because to do so it ends up in a situation where it has no choice but to enter a dangerous area; clearly, these specifications are too restrictive, as could be expected from Proposition 4.3.1.

We now turn to weak compliance instead. We remove the primary obligation (which we can do in this case, according to Proposition 4.3.2) and run the synthesis algorithms to get the following specification:

$$G(at\_danger \to (\neg empty \land X(empty)))$$
$$\land G(at\_danger \land has\_wood \to (\neg empty \land X(empty)))$$
$$\land G(at\_danger \land has\_ore \to (\neg empty \land X(empty)))$$
$$\land G(at\_danger \land has\_wood \land has\_ore \to (\neg empty \land X(empty)))^6 \quad (4.7)$$

where $\neg empty := has\_wood \lor has\_ore$, so the initial and next conditions for *unload* are $\neg empty$ and *empty* respectively. We can see that this specification derived from *mandatoryActs* does not prevent us from entering the dangerous areas at all, so the optimal path under these conditions will lead through both dangerous areas (see Fig. 4.2).
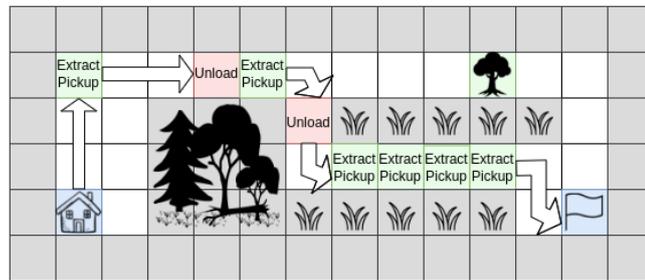
Figure 4.2: The optimal path the agent can take while adhering to Spec.4.7.

---

[5]Note that this is semantically equivalent to $G(\neg at\_danger)$

[6]Note that this is semantically equivalent to $G(at\_danger \to (\neg empty \land X(empty)))$

In other words, we have two options; full compliance, in which case the agent can never leave its home area, and weak compliance, which amounts to behaviour appropriate for a compensatory CTD. However, if we take the above scenario to be a sub-ideal CTD, the agent should not enter the dangerous area when it is not necessary to do so; the compliance specifications fail to accommodate this scenario.

4.3.4 Evaluation of the Implicit Representation of Norms in LTL

It is clear from the small case study above that compliance specifications – leveraging the implicit representation of norms in LTL in the safe RL paradigm – though effective in some cases, are limited in how well they can represent certain normative systems. First, we described how the issue of defining precise action translations has the potential to cause problems when it comes to the precision of compliance specifications. We also saw that while we were able to accommodate normative conflict and represent a compensatory-style CTD, it failed to induce appropriate behaviour for a sub-ideal CTD – which proved to be a more problematic kind of behaviour. Thus, when it comes to normative resilience, while the synthesis of compliance specifications allows us to deal with normative conflicts with a fundamentally monotonic logic, it falls short in the case of norm violation, where contrary-to-duty obligations were designated as a notable benchmark. Along the same lines, there is no clear way to deal with normative deadlock in general. Modifying the normative system is possible in this framework, but it will require that the expensive Algorithms 4.1 and 4.2 are run again, and whatever technique we use to restrict an RL agent to behaviour to the specification must be reconfigured.

Additionally, on the fronts of accountability and transparency, the collapsing of the above normative systems into compliance specifications served to obfuscate the context that could be parsed from the constitutive norms of each normative system; Specifications 4.4 and 4.5 make no reference to the obligation to be environmentally friendly or the concept of deforestation – removing context that reveals why these restrictions exist. Though there is some transparency to be found in the fact that the constraints derivable from the normative system must be represented explicitly in the compliance specification, as we have noted, the compliance specification is a specification of compliant behaviour, not of the norms that define it – any representation of norms is *implicit*. This could prove illuminating in some instances – it could elucidate what kind of behaviour a normative system will induce in practise – but we would argue that it doesn't provide a full picture of the facts relevant to accountability.

Clearly, while compliance specifications provide a way to leverage existing approaches in safe RL for producing behaviour that is normatively compliant with maximal probability, the approach falls short of both normative resilience and transparency. It is clear that normative resilience differs from safety; while safety properties can be, for the most part, naturally represented in LTL, normative systems require computationally expensive pre-processing, with the resulting specifications failing to capture certain desirable behaviours.

It is possible that some variation of compliance specifications – perhaps compliance specifications that cover state as well as action labels, or systems of compliance specifications that cover both ideal and sub-ideal behaviour – can prove effective for representing a wider variety normative systems. However, to the best of our knowledge these techniques do not yet exist for incorporating such a variation into reinforcement learning.

4.4 Concluding Remarks

In this chapter, we saw how existing tools and frameworks, even those geared toward learning "ethical" policies, fail in providing an effective framework for incorporating normative reasoning into RL. With this revelation, we turned our attention to the more developed field of *safe* RL – in particular, RL techniques for learning policies that satisfy a safety specification in LTL – in the hopes that these methods might remedy some of the issues we encountered in the literature presenting frameworks for ethically-sensitive RL. This is how we addressed our second research question, on whether norm-compliant RL can be accomplished via safe RL.

Firstly, we saw how compliance in the normative sense is fundamentally different from, for instance, satisfying a safety property; this came down to the critical disconnect between compliance and truth. Moreover, the complexity of some normative systems prevents us from a simple translation into compliance specifications, requiring instead the computationally heavy synthesis algorithm we introduced, which allowed us to incorporate constitutive norms into compliance specifications and account for normative conflict within the system. However, even with the synthesis algorithm, norm violation proved a problematic element; we saw that compliance specifications as they appear here have a marked inability to model sub-ideal contrary-to-duty scenarios, and by extension other types of normative deadlock.

An additional hurdle we encountered was reasoning about actions. In legal and moral reasoning, we often care about whether or not a certain action is performed, regardless of its consequences (i.e., regardless of what state performing the action results in). A further disadvantage of this approach is that it could be argued that by compiling constitutive norms into a single specification, we obfuscate the normative system, decreasing transparency.

When all is said and done, a lot of the issues we encountered stemmed from trying to utilize a language not geared specifically toward normative reasoning (that is, LTL), and working with a purely state-based framework. In the next chapter, we will develop a novel framework that deals with the problems presented in this chapter – while creating new ones.

The Normative Supervisor

We describe in this chapter a tool for enabling norm-conformance bolstered with transparency and normative resilience in RL agents, along with one of a few different techniques which it can be used to implement. We can identify our approach here as somewhat naive, in the sense that it will not be a truly hybrid approach; in the technique we will expound on, the presented module functions independently of the RL agent, and could indeed be utilized with other kinds of agents. It knows nothing about the agent and the environment beyond simple facts supplied by the agent in real time and any insights that are encoded in the system as norms. This straightforward model of a decoupled agent and normative reasoning module has both strengths and weaknesses, which we will explore with an extensive set of experiments in Section 5.3.

First, however, we will introduce the approach we take – interfering with an RL agent's policy by limiting the actions it can take to those that are compliant with a given normative system – and discuss how we are implementing it here, as was first presented in [NBCG21]. We will then carefully dissect the architecture of a module for carrying out this implementation; this module was presented in [NBCG21], but we will give a much more detailed overview here (which has not been published), including a description of a second reasoning model at the core of the implementation, which prioritizes expressivity over efficiency (whereas the first does the opposite). Once we have a comprehensive understanding of this module's architecture, we will utilize it for *online compliance checking* over the course of a thorough set of experiments, most of which come from [NBCG22]. We begin with experiments in a deterministic environment (the merchant environment, Section 5.3.1), and then move on to a simple non-deterministic environment (miniature Pac-Man, Section 5.3.2), before delving into a wide array of experiments in the regular Pac-Man environment (in Section 5.3.3).

5.1 The Approach: Interfering with the RL Framework

Normative interference, as we have called it, is a method for designing agents with normatively-compliant behaviour without having to create a novel agent model. Autonomous agents are composed of various interlinked processes, each of which takes an input from either the agent's environment or another internal component, and outputs either input for another process or, in the case of actuators, produces the action the agent has determined is appropriate in its current state. In theory, we should be able to interfere with some of these inputs and outputs, modifying them in order to elicit different behaviours from the agent.

In this section, we describe the approach we take in this chapter: interfering in the action selection phase. Part of this approach is the assumption that we are dealing with an RL agent that is already trained; that is, we are dealing with an agent that already knows how to complete its assigned objective in an optimal way, having learned its Q-function $Q(s, a)$, and therefore its policy. Moreover, we assume that knowledge of this is encoded solely in the value function the agent has learned. In other words, the goal is not to create compliant plans – it is to interfere with existing plans by preventing the agent from performing actions that are forbidden according to a given normative system. We let the agent do its job unbothered until its job entails a non-compliant action. The point before which the agent will choose a non-compliant (but optimal) action is our point of normative interference; in particular, when the agent is about to select an action from a set of possible actions, we preemptively remove non-compliant actions from this set. In theory, there are many ways in which we could interfere with an agent's programming in order to produce compliant behaviour; we could interfere in its reward signal during training or the input to its observation function, for instance. However, directly interfering with the agent's available choices at the action selection phase seems like the most straightforward approach to take here.
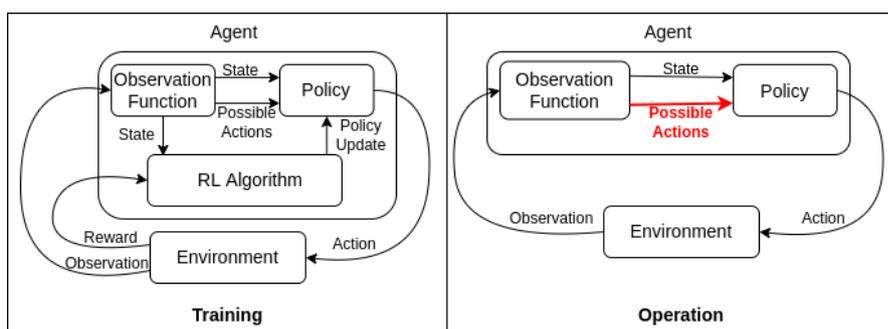
Figure 5.1: Our point of normative interference is highlighted in red.

Our approach bears a marked resemblance to the pre-shielding discussed in [ABE+18], and similar to [ABE+18] and [JKJ+20], we aim for minimal interference in the agent's duties; as we will see, though, our approach of only interfering when the agent is about to perform a non-compliant action has some notable disadvantages, when compared to

the shield synthesis of [ABE$^+$18] and [JKJ$^+$20]. However, the synthesis of shields requires knowledge of the agent and the environment in order to synthesize safe plans. Our approach, on the other hand, relies instead only on the agent's current observations and some basic knowledge of the environment's mechanics, and therefore can only prevent actions that are immediately problematic.

Recall that a Q-learning agent's policy is:

$$\pi(s) \in argmax_{a \in A(s)} Q(s, a)$$

Our goal in our normative interference is to remove from the agent's possible actions $A(s)$ all actions a which would violate the applicable normative system. What we are doing, then, for a normative system $\mathcal{N}$, is changing the policy to

$$\pi_{\mathcal{N}}(s) \in argmax_{a \in A_{\mathcal{N}}(s)} Q(s, a) \tag{5.1}$$

where $A_{\mathcal{N}}(s) \subseteq A(s)$ is defined as follows:

Definition 5.1.1. $A_{\mathcal{N}}(s)$ *is the (non-empty) set of normatively optimal actions in state* s, *defined as:*

$$A_{\mathcal{N}}(s) = \begin{cases} A_C(s) & A_C(s) \neq \varnothing \\ A_{NC}(s) & otherwise \end{cases} \tag{5.2}$$

where A_C *is the set of actions that does not violate* $\mathcal{N}$, *and* A_{NC} *is the set of actions minimally non-compliant with* $\mathcal{N}$.

How A_C and A_{NC} are precisely defined will depend on which model for reasoning we select; we will introduce two reasoner models in Section 5.2.2.

In brief, our approach – a particular form of normative interference which we have dubbed *online compliance checking* – involves translating immediate state data from the agent into some symbolic representation and reasoning about this data with respect to a formally represented normative system. From there we infer which actions the agent could take are non-compliant, and remove those from the agent's arsenal. We will implement this process through an external module for RL agents, which we introduce below as the *normative supervisor.*

5.2 The Normative Supervisor

The normative supervisor [NBCG21] is an external module that can be used for both online compliance checking (OCC) in trained RL agents and norm guided reinforcement learning (NGRL, which will be introduced in the next chapter). It is composed primarily of (front-end and back-end) translator modules and a reasoner module (see Fig. 5.2 for the high level architecture of the supervisor). Though in theory the normative supervisor could be implemented with another reasoning engine at its core, discussion
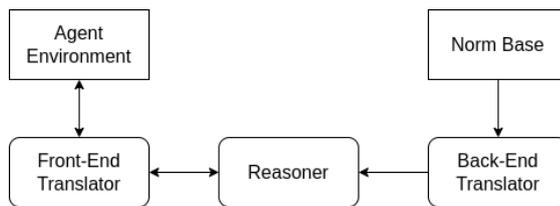
Figure 5.2: The basic architecture of the normative supervisor. Here, the *norm base* is a knowledge base containing all norms associated with the normative system $\mathcal{N}$.

of the framework in this chapter will focus on the supervisor as implemented with the defeasible deontic logic (DDL) theorem prover SPINdle [LG09].

The supervisor monitors the agent as it interacts with its environment, dynamically translating states and the applicable normative system into defeasible theories in DDL; generally, facts about the environment (e.g., the locations of objects the agent observes) are translated to literals, and compose the set of facts of the defeasible theory. Note that to perform this translation, we do not necessarily need to have any kind of model of the environment (e.g., we do not need to have learned an MDP).

Meanwhile, the constitutive norms and regulative norms of the normative system are translated into DDL rules and added to the defeasible theory. We use $Th(s, \mathcal{N})$ to denote the theory generated for state s and the normative system $\mathcal{N}$ (this differs from $Th(\bar{s}, \mathcal{N})$ in that it has not yet been established which action from $A(s)$ is taken; this theory contains no fact referring to the performing of a specific action). Based on the output of the theorem prover, the supervisor can, for instance, filter out from a list of possible actions received from the agent the actions which do not comply with the applicable normative system; we call the result a set of compliant solutions. If no compliant solution can be found (that is, $A_C(s) = \varnothing$), the supervisor can construct a set of "lesser evil" or minimally non-compliant solutions containing the actions which violate as little of the normative system as possible (that is, $A_{NC}(s)$).

Below, we walk through each main component of the supervisor and explain how these solutions are constructed. This will involve a discussion on how data from the agent and the normative system are translated into $Th(s, \mathcal{N})$ by the translators (Section 5.2.1), and then a detailed description of two different reasoner models that can be used to construct $A_{\mathcal{N}}(s)$ (Section 5.2.2). Finally, we end on a note on online compliance checking before we move on to our experimental results.

5.2.1   Translators

The normative supervisor employs two translator modules, one front-facing one that interfaces with the agent, and one back-facing one that translates the normative system (stored as the norm base) the agent is subject to. Both translators work together to build the defeasible theory $Th(s, \mathcal{N})$.

Front End

The front-end translator is in perpetual use, processing new data and proposed actions from the agent as the environment changes. It amounts to an algorithm that transforms input data from the agent into propositions which assert facts about the agent or the environment.

In the simplest case, when dealing with an environment modelled as a labelled Markov decision process $\mathcal{M} = \langle S, A, P, L \rangle$, all the front-end translator must do is build a representation of the current state from that state's labels; from each label the translator constructs a fact in the form of a propositional atom in whatever language the reasoner employs, and this is what composes the fact set F_s of $Th(s, \mathcal{N}) = \langle F_s, R_s^C, R_s^O, > \rangle$:

$$F_s = \{p \mid p \in L(s)\} \cup \{\neg p \mid p \in AP \smallsetminus L(s)\}$$

where AP is $rng(L)$. In other words, each atomic proposition in $L(s)$ is a fact in F_s, and for every $p \in AP$ not in $L(s)$, we add $\neg p$ to F_s.

But what about when we are not comfortably within the context of a labelled MDP? It might be the case that we receive a more heterogeneous collection of data when the agent supplies information about its state. In the Pac-Man games we walk through in Sections 5.3.2 and 5.3.3, for example, from the agent's state we receive data on the location of different game objects, which we receive as a set Obj_s. We can then define a number of unary predicates $\mathcal{P}$ a over these objects describing their location or state, from which we can derive facts, thereby effectively constructing a labelling function on the normative supervisor's side. Then for unary predicates $P \in \mathcal{P}$ and every object $x \in Obj_s$, if $P(x)$ is true the atomic proposition $P(x)$ is added to F_s as a fact.

In either case – labelled MDP or unlabelled MDP – for online compliance checking, once a set of compliant solutions has been constructed, it is translated back into a set of action labels $A_{\mathcal{N}}(s)$ to be returned to the agent to replace $A(s)$.

Back End

The back-end translator translates the normative system the agent is subject to (stored in the form of a norm base that contains the individual norms that construct this system) into whatever kind of formulas or rules the reasoner works with. Together with the facts translated by the front-end translator, these compose the theory $Th(s, \mathcal{N})$ that will be input into the reasoner.

Ideally, we will be working with a formalism capable of representing norms as naturally as possible. The translation of potentially complex norms into formal expressions is an opportunity to introduce errors into the framework; it is crucial that any kind of rule used (1) is triggered only in the correct circumstances, (2) behaves correctly when triggered, and (3) interacts with other rules appropriately. For this reason, it is preferable that norms can be translated more or less directly into formal expressions, better guaranteeing the integrity of the prescriptions to be derived.

Despite consisting of only facts and rules over literals, DDL can represent a range of different kinds of normative systems; we demonstrate explicitly how constitutive norms, regulative norms, and conflict resolution mechanisms are translated below.

Constitutive Norms.   In the simple case of constitutive norms over states of the form $\mathbf{C}(p, q|c)$ (i.e., "p counts as q in context c"), the translation is quite direct:

$$r_1: \ c, p \rightarrow_C q \tag{5.3}$$

Generally, these constitutive norms can be represented as strict rules; this has the advantage of perpetuating the status of facts as definite conclusions (recall, definite conclusions can only be reached through inference with facts and strict rules). In the case that there is an exception to a constitutive norm (for example, we could say "a C is a bad grade, *except* in extenuating circumstances"), we can use instead a defeasible rule $c, p \Rightarrow_C q$. In the specific case of constitutive norms over actions, their implementation will depend on which kind of reasoner we employ; we will discuss two options in the next subsection.

It should be noted that the normative supervisor generates DDL representations of constitutive norms on the fly. Basically, whenever the context c is true, the rule $r1 : c, p \rightarrow_C q$ will be generated and added to $Th(s, \mathcal{N})$. The reason for this is twofold; firstly, having dozens (in more complex cases, hundreds or thousands) of unused constitutive norms obfuscates $Th(s, \mathcal{N})$, making it a less transparent picture of what is going on in state s. Additionally, and perhaps more crucially, SPINdle slows down as we add more rules [LG09], so not generating every constitutive norm in the beginning reduces computational cost. This is particularly useful when we have many constitutive rules that have the same form.

Example 5.2.1. *Consider the Pac-Man environment where in each state, Pac-Man is located at some coordinate (i, j). In some states it may be the case that a ghost is located at $(i, j + 1)$, and so we could say that the ghost is in the north range of Pac-Man; we could then define a constitutive norm*

$$\mathbf{C}(ghost_{i,j+1}, inNorthRange_{pacman,ghost}|pacman_{i,j})$$

for each (i, j). However, instead of generating every rule

$$pacman_{i,j}, ghost_{i,j+1} \rightarrow_C inNorthRange_{pacman,ghost}$$

for all pairs (i, j) in the beginning, whenever Pac-Man enters cell $(1, 1)$, for instance, the rule

$$pacman_{1,1}, ghost_{1,2} \rightarrow_C inNorthRange_{pacman,ghost}$$

is generated and added to $Th(s, \mathcal{N})$.

Regulative Norms: Obligations. Dyadic obligations (and prohibitions) are also quite easy to translate into DDL. We will generally translate the dyadic obligation $\mathbf{O}(p|q)$ as

$$r_2 : \quad q \Rightarrow_O p \tag{5.4}$$

Naturally, the dyadic prohibition $\mathbf{F}(p|q)$ can be formalized as $r_3 : \; q \Rightarrow_O \neg p$.

Though not strictly necessary in all cases, we will generally default to translating prescriptive norms to defeasible rules; unlike constitutive norms, they commonly have exceptions or priorities attached to them[1]. Moreover, even if no such complications exist *currently* in the normative system, a truly normatively resilient framework for reasoning will be able accommodate the later addition of regulative norms that conflict with the current system in some way. Always instantiating prescriptive norms as defeasible rules gives us the option to simply add new rules along with a superiority relation to clear up any conflicts.

Regulative Norms: Strong Permissions. Permissions are a somewhat more complicated matter; we can sometimes define indirect permissions (for example: we might prohibit proceeding at a red light, and then give special permission to turn right if it is safe to do so, where turning right counts as proceeding; since it is not turning right that was forbidden, the permission to turn right does not directly correspond to the prohibition). For now, though, we will only look at permissions that act as direct exceptions to prescriptive norms. Generally, we can simply translate the dyadic permission $\mathbf{P}(p|q)$ as

$$r_4 : \quad q \rightsquigarrow_O p$$

However, this formulation does not allow us to derive $\mathbf{P}(p)$ when q is true (r_4 prevents us from concluding $+\partial_O \neg p$, but it does not yield a conclusion such as $+\partial_P p$), and as a result we cannot use it trigger other norms (as is done in [Gov15]).

That being said, in practise, if we have the dyadic permission $\mathbf{P}(p|q)$, we will translate it as two rules:

$$r_{4a} : \quad q \Rightarrow_P p \quad \& \quad r_{4b} : \; \mathbf{P}(p) \rightsquigarrow_O p \tag{5.5}$$

r_{4a} makes it so that when q is true, we can derive $\mathbf{P}(p)$ and trigger rules such as $\mathbf{P}(p) \Rightarrow_P p'$ for some p'; at the same time, r_{4b} allows us to prevent the conclusion $+\partial_O \neg p$ from being reached.

To accommodate these rules, we need to add a new modal operator to our formalism, but only to be applied to certain rules. If we add P to MOD (the set of modalities for our modal defeasible logic; up until now, it has only contained O) more generally, it will not behave as we want permission to behave; it is clear that $\mathbf{P}(p) \wedge \mathbf{P}(\neg p)$ is not a deontic conflict, but if we add it to DDL as a modality in the same way as O, Proposition 2.3.2 tells us that these two assertions will indeed conflict. Instead, the

---

[1]Additionally, the "lesser evil" mechanism of the first reasoner model to be introduced in the next section relies on the defeasibility of regulative norms.

P modality will only ever be used in the context of a strong permission that is an exception to an obligation; we are putting a restriction on our normative system, not to add superfluous strong permissions (strong permissions that do not actively act as an exception). Then, we can have a defeasible theory that proves $+\partial_P p$ without issues because in the permission's absence, we would derive $+\partial_O \neg p$; by Proposition 2.3.2, this means that we cannot simultaneously have $+\partial_O p$, which means that we would not be able to derive $+\partial_P \neg p$ from another permissive norm. That is, we won't have this particular situation where two opposing permissions can be derived simultaneously. A much more extensive and sophisticated treatment of permission in DDL can be found in [GORS13].

Default Rules: Non-Concurrence.   Another task allotted to the (back-end) translator is the construction of what we call "non-concurrence rules".

A reinforcement learning agent can only perform a single action in each state, and so the obligation to perform one action counts as a prohibition to perform any other possible action. Each translator can therefore generate a set of action constitutive norms for a set of actions $A(s)$:

$$\bigcup_{a \in A(s)} \{\mathbf{C}(a, \neg a' | \top) | a' \in A(s) \smallsetminus \{a\}\} \tag{5.6}$$

Namely, for each action a we generate a constitutive norm that states that a (shorthand for the proposition "action a is performed") counts as $\neg a'$ (that is, "it is not the case that a' is performed) for all other actions a'.

Translating Conflict Resolution.   DDL is a defeasible logic, and comes with conflict-resolving mechanisms. In particular, it allows for a defeasibility mechanism in the form of a superiority relation over rules. For the case of direct normative conflicts, we can add an ordered pair to the superiority relation $>$. For example, if we have $r_1 : q \Rightarrow_O p$ and $r_2 : q \Rightarrow_O \neg p$, where r_1 takes priority over r_2, we would add $(r_1, r_2) \in >$.

5.2.2   Reasoner

The reasoner is at the core of the normative supervisor, and its key ingredient is the theorem prover (or other reasoning engine) it utilizes. We will present two reasoner models; the first (original [NBCG21]) reasoner model prioritizes efficiency over being able to cope with a broad array of normative systems, whereas the second is capable of dealing with a wider variety normative systems, but is more computationally expensive.

The task of the reasoner is to take whatever is output by the translators (e.g., a defeasible theory composed of facts from the environment, a normative system, and non-concurrence rules) and construct a set of compliant solutions. We consider a compliant solution to be a possible non-violating course of action for the agent. If possible, this is what we would like to extract from the reasoner.

Definition 5.2.1. *A set of compliant solutions with respect to $\mathcal{N}$ is: (a) non-empty, and consisting only of (b) possible actions, (c) composed of courses of action that do not*

violate $\mathcal{N}$, and (d) solutions that are internally consistent (i.e., we do not have solutions involving performing two actions that cannot be performed together).

Note that for RL agents, (d) is a given; RL agents can only perform one action for timestep, so the set of compliant solution will just be a set of single-action solutions. Thus, by conditions (b) and (c), the set of compliant solutions in a state s should just be $A_C(s)$, from Expression 5.2.

In some states, a set of compliant solutions may not exist; in these cases, it is desirable to have a notion of which actions are *more compliant* than others, so that those can be taken in the interest of minimizing damage done by the inevitable violation. This introduces the need for a metric for measuring *how compliant* a given course of action is. Our solution is inspired by the *economy principle* [FF22], a conflict-resolution principle postulated by an ancient Indian philosophical school[2]. In line with this principle, we will take the action that results in the fewest number of violations – a solution that is "minimally non-compliant". This is more or less a "lesser evil" solution. The precise formulation of these is what will allow the normative supervisor to deal with indirect normative conflict and CTD obligations, along with other scenarios involving normative deadlock.

There are multiple ways to plausibly construct a set of compliant solutions and lesser evil solutions. This will depend on how we define $A_C(s)$ (the set of compliant actions in s) and $A_{NC}(s)$ (the set of minimally non-compliant actions in s). Below, we will walk through the mechanics of two different reasoners which implement normative reasoning with DDL for the normative supervisor, each with their own conception of what is meant by a compliant solution. The first reasoner relies on the direct derivation of a prohibition of an action, in order to exclude it from $A_C(s)$; this reasoner is more computationally efficient, but places some limitations on which normative systems we can implement. The second reasoner excludes non-compliant actions a based on whether we can derive from $Th(\bar{s} = (s, a), \mathcal{N})$ the obligation of some literal *lit* which is not fulfilled. This reasoner model is less efficient, but allows more expressivity with respect to which normative systems can be implemented. Thus, when it comes to implementation, we have two options; in one, the reasoner functions efficiently, only calling the theorem prover once per state (as long as there are compliant actions in that state; computing $A_{NC}(s)$ is somewhat more involved). However, significant preprocessing must be done (as we will see below) and some normative systems might be very difficult, or perhaps impossible, to implement. On the other hand, we have a second option in a reasoner that must generally call the theorem prover more than once per state; however, this model gives us more expressivity with respect to which normative systems it can accommodate, and enjoys a more comprehensive notion of violation, reflecting Definition 2.2.5.

---

[2]The economy was discussed by the Mīmāṃsā author Kumārila (7th c. CE), in the context of solving potential conflicts among the commands in the Vedas (Indian sacred texts). The principle says that a norm that conflicts with the smallest number possible of other norms should be preferred.

First Reasoner Model

We start by introducing the first reasoner model; we first define $A_C(s)$ and then justify (through an example) and introduce the rather idiosyncratic way in which we must translate constitutive norms over actions to accommodate this definition. We will then demonstrate the limited expressivity of our formalization with another example. After, we will introduce an algorithm for computing compliant solutions and show that it always yields an empty set or a compliant solution (with respect to our definition of $A_C(s)$). For the cases where $A_C(s)$ (and thus the set of compliant solutions) is empty, a second algorithm (the 'Lesser Evil' algorithm) is introduced to generate the actions in A_{NC}. After an additional note on constitutive norms over actions and finally a short discussion of indirect normative conflict and strong permissions, we move on to discuss the second reasoner.

The first incarnation of the normative supervisor was presented in [NBCG21], where the reasoner was designed with optimal performance in mind; that is, the goal of this implementation was to utilize the reasoning engine (SPINdle, in this case) as little as possible. In this model, we define the set of compliant actions $A_C(s)$ from Equation 5.1 as those actions which are not forbidden in the normative system. Formally:

$$A_C(s) = \{a \in A(s) \mid Th(s, \mathcal{N}) \nvdash +\partial_O \neg a\} \tag{5.7}$$

The set of compliant actions, then, are those actions a in $A(s)$ such that we cannot prove $+\partial_O \neg a$; that is, we cannot prove $\mathbf{F}(a)$.

Note that if $Th(s, \mathcal{N}) \vdash +\partial_O a$ (that is, a is obligatory), $Th(s, \mathcal{N}) \nvdash +\partial_O \neg a$ follows (see Lemmas 2.3.1 and 2.3.2 in Chapter 2), so this set both excludes forbidden actions and includes obliged actions. This definition of $A_C(s)$ is closely related to Definition 2.2.2; we are talking about specific obligations here, and whether or not they would be violated by a particular action. If we can derive $\mathbf{O}(\neg a)$ from $Th(s, \mathcal{N})$, then by taking action a we clearly have violated some regulative norm – some obligation in particular – according to Definition 2.2.2. If we can derive $\mathbf{O}(a)$, then because of the non-concurrence rules, we should be able to derive $\mathbf{O}(\neg a')$ for every other action a'. Whether this works in practise will depend on how exactly we translate constitutive norms over actions, as we will see below.

It is notable that in this framework, we can only recognize violations based on whether the prohibition of an action can be directly derived from $Th(s, \mathcal{N})$ – this may exclude some violations as defined in Definition 2.2.5. We demonstrate in the example below.

Example 5.2.2. *Suppose we have an action $a \in A(s)$ and the theory $Th(s, \mathcal{N}) = \langle F_s, R_s^C, R_s^O, \rangle$, where $R_s^C = \{r_1\}$ and $R_s^O = \{r_2\}$. If*

$$r_1 : a \to_C b$$

is the constitutive rule and

$$r_2 : \Rightarrow_O \neg b$$

is the regulative rule, we cannot derive $+\partial_O \neg a$, and therefore $a \in A_C(s)$ according to Expression 5.7. However, according to Definition 2.2.5, performing the action a results in a violation of $\mathcal{N}$.

Hence, merely translating $\mathbf{C}(a, b|c)$ to $a, c \rightarrow_C b$ for $a \in Act$ will not allow us to recognize clear infractions of the normative system, in this reasoner model. We will need to reformulate constitutive norms over actions, to accommodate this reasoner.

Constitutive Norms over Actions. In order for this reasoning framework to be effective, we must be able to directly derive the explicit obligation or prohibition of the agent's possible actions from the $Th(s, \mathcal{N})$; that is, if a is to be excluded from $A_C(s)$, for example, we must be able derive $+\partial_O \neg a$. In cases where we have a complex structure of constitutive norms, this can require some creative engineering of the norm base, as we will see in the case study in Section 5.3.3 below. As a whole, we will generally generate two "strategy" rules for a constitutive norm over actions $\mathbf{C}(a_1, a_2|c)$:

$$C_1: \ c, \mathbf{O}(a_1) \Rightarrow_O a_2 \ \text{ and } \ C_2: \ c, \mathbf{O}(\neg a_2) \Rightarrow_O \neg a_1 \tag{5.8}$$

These rules link the obligation of a more concrete action to the obligation of a more abstract action, and the prohibition of a more abstract action to the prohibition of the more concrete action. This way we can establish a deductive link between regulative norms over more abstract actions with less abstract actions, such as the actions in *Act*. What we are doing here is essentially creating rules that do not represent the constitutive norm directly, so to speak, but rather act as rules for how to derive conclusions from it in combination with other regulative norms. This allows us to overcome some of the expressivity limitations of DDL when used in the way necessary for the first reasoner model.

There are, however, disadvantages inherent to this approach; we show below how when we try to reason about obligations of actions defined by one or more constitutive norms, problems arise. First, we will demonstrate the case in which our translation of constitutive norms over actions remains effective, and then show that a slight revision of the normative system introduces issues which at first glance, we can, in a way, resolve – however, we will proceed to show that this solution is fundamentally incorrect and causes additional problems.

Example 5.2.3. *Let's return to Example 5.2.2, and suppose we translate the constitutive norm underlying r_1 – $\mathbf{C}(a, b|\top)$ – as suggested above, so that $R_s^C = \varnothing$ and $R_s^O = \{r_2, r_3, r_4\}$, where*

$$r_3: \ \mathbf{O}(a) \Rightarrow_O b$$

and

$$r_4: \ \mathbf{O}(\neg b) \Rightarrow_O \neg a$$

Then it is clear that we will be able to derive $+\partial_O \neg a$ from r_2 and r_4, and so according to Expression 5.7, $a \notin A_C(s)$, and a is not a compliant solution.

However, suppose we replace $r_2 : \Rightarrow_O \neg b$ (forbidding b) with $r'_2 : \Rightarrow_O b$ (obliging b) (i.e., now $R^O_s = \{r'_2, r_3, r_4\}$). a should be a compliant solution in $A_C(s)$, but $A_C(s)$ should exclude any other action $a' \in A(s)$. Expression 5.7 demands that in order to make this happen, we must be able to derive $+\partial_O \neg a'$ for all other a'. So let's suppose we add another rule to those proposed in Expression 5.8, $C_3 : c, \boldsymbol{O}(a_2) \Rightarrow_O a_1$. Then $R^O_s = \{r'_2, r_3, r_4, r_5\}$, where $r_5 : \boldsymbol{O}(b) \Rightarrow_O a$, and if we add the non-concurrence rules, we will be able to derive $+\partial_O \neg a'$ for all other a'. However, this is not a valid solution.

Suppose we add to the scenario one more action $a'' \in A(s)$ such that $\boldsymbol{C}(a'', b|\top)$, so that $R^O_s = \{r_1, r_3, r_4, r_5, r_6, r_7, r_8\}$ where

$$r_6 : \boldsymbol{O}(a'') \Rightarrow_O b,$$

$$r_7 : \boldsymbol{O}(\neg b) \Rightarrow_O \neg a'',$$

and

$$r_8 : \boldsymbol{O}(b) \Rightarrow_O a''$$

*(corresponding to C_1, C_2, and C_3 respectively). Informally, we would like to derive the fact that either a or a'' should be selected in order to guarantee compliance; that is, that a and a'' are both compliant solutions. Instead, from r'_2, r_5, r_8 we can derive $+\partial_O a, +\partial_O a''$, which is not true; a **and** a'' are not obligatory – a **or** a'' is obligatory. This is a problem; if we derive that a and a'' are both obligatory, our non-concurrence rules will result in both actions being forbidden, and suddenly, $Th(s, \mathcal{N})$ is no longer O-consistent. Clearly, the addition of C_3 to Expression 5.8 is problematic; so will any rule that allows us to derive the obligation of a from the obligation of b.*

It is apparent through this example that our conception of $A_C(s)$ places intrinsic limits on what normative systems we can implement with this reasoner model. Specifically, certain combinations between obligations and constitutive norms are not feasible. Nevertheless, before moving on to a reasoner model that solves this issue, we will delve into how this reasoner functions.

The Compliant Solutions Algorithm.   In order to build a set of compliant solutions based on what courses of action are explicitly forbidden or obliged, we will generally take the following steps:

1. We remove all conclusions that do not reference a literal in $A(s)$; these are irrelevant to the agent.

2. Any action corresponding to a defeasibly proved positive literal occurs in every solution.

3. Any action corresponding to a defeasibly proved negative literal is discarded from every solution.

Of course, for an RL agent with only single-action solutions, this can be simplified by taking the non-concurrence rules into account. The pseudocode for an algorithm implementing this procedure (simplified) is given as Algorithm 5.1.

Algorithm 5.1: CompliantSolutions1

    input : $Th(s, \mathcal{N})$, $A(s)$
    output: $legalActions$

1 begin
2 $legalActions \leftarrow A(s)$;
3 $reasoner \leftarrow SPINdle.Reasoner$;
4 $conclusions \leftarrow reasoner.generateConclusions(Th(s, \mathcal{N}))$;
5   **for** $a \in A(s)$ **do**
6   **if** $conclusions.has(+\partial_O \neg a)$ **then**
7 $legalActions.remove(a)$;
8   **end**
9   **end**
10   **return** $legalActions$
11 end

We can be certain that if Algorithm 5.1 returns a non-empty set, it provides a set of compliant solutions:

Theorem 5.2.1. *Algorithm 5.1 yields either an empty set or a compliant solution.*

Proof. For conditions (b) and (c) of Definition 5.2.1, recall that we have identified a set of compliant solutions as $A_C(s) \subseteq A(s)$. Algorithm 5.1 removes every action a from $A(s)$ such that $Th(s, \mathcal{N}) \vdash +\partial_O \neg a$. According to Expression 5.7, $A_C(s) = A(s) \smallsetminus \{a \in A(s) | Th(s, \mathcal{N}) \vdash +\partial_O \neg a\}$, so the algorithm yields only (and all) members of $A_C(s)$, fulfilling condition (c).

Since Algorithm 5.1 returns a set of single-action solutions (as RL agents can only select a single action to perform in a given state), these solutions are trivially consistent, fulfilling condition (d). $\square$

Additionally, since we only have to run SPINdle once, it is clear that this algorithm is relatively computationally feasible.

Corollary 5.2.1. *Algorithm 5.1 can be completed in linear time time with respect to the size of $Th(s, \mathcal{N})$.*

Proof. Conclusions are only generated once; the corollary directly follows from Theorem 2.3.1. $\square$

The Lesser Evil Algorithm. When Algorithm 5.1 returns an empty set, another algorithm is triggered, which selects minimally non-compliant actions. This entails running the reasoning engine again, for each possible action. In order to do this, we will have to be somewhat creative in our use of SPINdle. Beyond the conclusions yielded by SPINdle, the theorem prover also has an inference logger that classifies every rule in the theory as discarded, applicable, or defeated; we employ SPINdle in an unconventional way, and use these logs to construct a set of "lesser evil" solutions. These solutions make up the set $A_{NC}(s)$ associated with the A_C constructed by Algorithm 5.1, and since this algorithm is run whenever $A_C = \varnothing$, this procedure (running Algorithm 5.1 and Algorithm 5.2 when the former returns an empty set) produces $A_{\mathcal{N}}$ as defined in Definition 5.1.1.

Algorithm 5.2: LesserEvil1

    input : $Th(s, \mathcal{N})$, $A(s)$
    output: $bestActions$

1 begin
2 $reasoner \leftarrow SPINdle.Reasoner$;
3 $scores \leftarrow \varnothing$;
4   **for** $a \in A(s)$ **do**
5 $Th(s, a, \mathcal{N}) \leftarrow Th(s, \mathcal{N})$;
6 $Th(s, a, \mathcal{N}).addFact(\mathbf{O}(a))$;
7 $reasoner.generateConclusions(Th(s, a, \mathcal{N}))$;
8 $logger \leftarrow InferenceLogger(reasoner)$;
9 $applied \leftarrow logger.getRules(status = APPLIED)$;
10 $defeated \leftarrow logger.getRules(status = DEFEATED)$;
11 $score \leftarrow applied.size() - defeated.size()$;
12 $scores.add([act, score])$
13   **end**
14 $max \leftarrow max(scores)$;
15 $bestActions \leftarrow scores.getActions(score = max)$;
16   **return** $bestActions$
17 end

Instead of reasoning again about $Th(s, \mathcal{N})$, we zero in on what we call $Th(s, a, \mathcal{N})$ – where instead of simply translating facts about the state s, we also include a fact about an action a that is hypothetically obligatory in state s. What we are doing here is, for each action a, asserting $\mathbf{O}(a)$ as a fact. Since facts defeat opposing defeasible rules, and we have formulated our regulative norms (and constitutive norms over actions) as defeasible rules, any rules expressing obligations or prohibitions in force that would be violated by taking action a will be defeated by the assertion of $\mathbf{O}(a)$. Meanwhile, any rules expressing norms that are complied with will retain the status of being applied. What Algorithm 5.2 does, then, is count how many obligations are complied with (in the sense of Definition 2.2.2), and how many are violated by a specific course of action. The action that complies with the most norms and violates the fewest will be selected as the

"lesser evil".

What Algorithm 5.2 essentially yields, then, is a possible rendition of the set $A_{NC}(s)$ (the set of minimally non-compliant actions, which constitutes the set of normatively optimal actions, $A_{\mathcal{N}}(s)$, when $A_C(s) = \varnothing$; see Expression 5.2), where we say an action is "minimally non-compliant" if the agent will violate the fewest possible number of individual rules prohibiting it. By triggering this algorithm whenever Algorithm 5.1 yields an empty set, then, we can effectively model $A_{\mathcal{N}}(s)$ as defined in Expression 5.2.

It is worth noting that this algorithm to calculate A_{NC} is somewhat more costly than its counterpart for compliant actions, as is shown below:

Corollary 5.2.2. *Algorithm 5.2 can be completed in polynomial time with respect to* $|Th(s,\mathcal{N})| + 1$.

Proof. As shown in Theorem 2.3.1, each run of $reasoner.generateConclusions(Th(s,a,\mathcal{N}))$ can be completed in linear time with respect to the size of the theory, which is $Th(s,\mathcal{N})$ with one fact added. This is done $|A(s)|$ times, and since every action in $A(s)$ is a literal occurring in $Th(s,a,\mathcal{N})$, $|A(s)| < |Th(s,a,\mathcal{N})|$, so Algorithm 5.2 can be completed in polynomial time. $\qquad\square$

Summarily, the cost of computing A_{NC} is heftier than computing A_C, but not by an alarming margin.

Revisiting Constitutive Norms over Actions. We will add a further rule for constitutive norms over actions in the interest of allowing Algorithm 5.2 to function more effectively. When Algorithm 5.2 cycles through actions a, adding $\mathbf{O}(a)$ to $Th(s,\mathcal{N})$, we want this to activate any constitutive norms linking a to more abstract actions. However, it is possible that we might have norms which are triggered by the selection of a course of action; for norms $\mathbf{C}(a_1, a_2|c)$ where a_1 is a member of $A(s)$, we add a final rule:

$$C_3: \ c, \mathbf{O}(a_1) \to_C a_2$$

This allows us to effectively use deontic detachment; when we suppose $\mathbf{O}(a_1)$ in Algorithm 5.2 and have a rule like, for example, $a_2 \Rightarrow_O a_3$, in the spirit of deontic detachment we can derive $\mathbf{O}(a_3)$ as well. This will allow us to define CTD obligations later on. We demonstrate a CTD scenario that relies on this representation of constitutive norms over actions below.

Example 5.2.4. *Suppose for a defeasible theory $Th(s,\mathcal{N})$ we have a primary obligation $r_p: \Rightarrow_O \neg b \in R_s^O$ and the CTD obligation $r_{ctd}: \ b \Rightarrow_O a_1 \in R_s^O$. Now suppose we have two constitutive norms $\mathbf{C}(a_1, b|\top)$, $\mathbf{C}(a_2, b|\top)$, where $A(s) = \{a_1, a_2\}$. From Expression 5.8, we get rules corresponding to C_2:*

$$r_i: \ \mathbf{O}(\neg b) \Rightarrow_O \neg a_i \in R_s^O$$

for $i \in \{1, 2\}$.

As a result, we can derive $+\partial_O \neg a_1$, $+\partial_O \neg a_2$, so $a_1, a_2 \notin A_C(s)$ and $A_C(s) = \varnothing$, so Algorithm 5.2 is triggered.

Now for each of $a_i \in A(s)$, in line 6 of Algorithm 5.2, we add $\mathbf{O}(a_i)$ to $Th(s, \mathcal{N})$. According to C_1 of Expression 5.8, we also have the following rules:

$$r_{i+2} : \ \mathbf{O}(a_i) \Rightarrow_O b \in R_s^O$$

for $i \in \{1, 2\}$. Thus, for each a_i, $\mathbf{O}(b)$ is true, which conflicts with r_p, and as a result both actions will be in $A_{NC}(s)$. However, if we add rules corresponding to C_3 we get:

$$r_{i+4} : \ \mathbf{O}(a_i) \rightarrow_C b \in R_s^C$$

for $i \in \{1, 2\}$.

Then r_{ctd} will be triggered by b being true (derivable from $\mathbf{O}(a_1)$ and r_5 or $\mathbf{O}(a_2)$ and r_6). When we add $\mathbf{O}(a_2)$ to $Th(s, \mathcal{N})$, from the non-concurrence rules we will be able to derive $\mathbf{O}(\neg a_1)$, which will now conflict with r_{ctd}, and since this is one more conflict than will occur after the introduction of $\mathbf{O}(a_1)$, Algorithm 5.2 will output $A_{NC}(s) = \{a_1\}$, so the agent will obey the CTD obligation.

We could, in theory, add other rules associated with a constitutive norm over actions; the above rules have been chosen for practical purposes, and in the presence of different normative systems from those demonstrated in Section 5.3, others might be more appropriate. The translation of constitutive norms over actions for the first reasoner model is in part a pragmatic matter: the question of how to get correct results from Algorithms 5.1 and 5.2 will determine how constitutive norms over actions should be translated. Of course, there are some translations we have seen are generally incorrect, such as translating $\mathbf{C}(a_1, a_2|c)$ to $c, \mathbf{O}(a_2) \Rightarrow_O a_1$ (as was demonstrated in Example 5.2.3). Additionally, adding a large collection of rules for each constitutive norm over actions serves to obfuscate the norm base, making it more difficult to see what's going on. Thus, any additional rules for constitutive norms over actions should be considered carefully.

Indirect Normative Conflict. The formulation of Algorithm 5.2 gives us a very easy way to resolve indirect normative conflicts. For indirect conflict resolution, we will collapse the concept of indirect conflict resolution into the case of normative deadlock; the idea here is to add another norm to "reinforce" the higher priority norm. Suppose that we have a regulative norm $r_1 : q \Rightarrow_O p$ but also lower priority norm $r_2 : q \Rightarrow_O p'$, where p and p' cannot be true simultaneously, but p is a preferable course of action. Then we might add another $r_1' : q \Rightarrow_O p$ such that if the agent disobeys r_1, it is violating *two* norms instead of one. This additional rule, as a a copy of r_1 has no impact when we look at which conclusions are derivable from $Th(s, \mathcal{N})$; however, when Algorithm 5.2 considers the inference log obtained when reasoning about $Th(s, \mathcal{N})$, it will reveal that

when r_1 is violated, so is r_1' – two rules have been violated, instead of one. Within the framework of the first reasoner model, this will force the agent to choose the action that results in one violation instead of two, and so it chooses to prioritize the higher-priority norm.

Indirect Permissions. There may be cases where we are forbidden from something that encompasses range of activities, but we are, under some conditions, permitted to do one of these activities. For example, we could say that one is forbidden from proceeding at a red light, but in certain jurisdictions, it is permissible to turn right at a red light so long as it is safe to do so. Of course, turning right counts as proceeding, so it is an exception to the aforementioned prohibition. However, it serves as an *indirect* permission since the prohibition and exception refer to different things; proceeding is forbidden, while turning right is permitted – the latter is only an exception to the former because the two concepts are related through a constitutive norm. In the first reasoner model, these can be simply modelled as regular permissions.

Assessing the First Reasoner Model. As noted above, in this first framework for reasoning, we run the reasoning engine once when we look for compliant actions, and then $|A(s)|$ times when we look for minimally non-compliant actions; thus, in most cases, we will only need to call the reasoning engine once per timestep. However, this efficiency comes with a cost.

Ideally, we would like to prove that the actions in the set *bestActions* yielded by Algorithm 5.2 results in fewer violations of $\mathcal{N}$ than any other actions in $A(s)$; however, we cannot prove this in general – we can only try to prove it for individual formalizations. In particular, it will depend on how constitutive norms over actions are implemented. Additionally, since we based our formulation of $A_C(s)$ on Definition 2.2.2 instead of 2.2.5, we run the risk of excluding some violations if we do not pre-process the norm base properly, using practical heuristics to determine how action constitutive norms are generated in this reasoner model. Finally, as was mentioned above, there may be some normative systems that cannot be faithfully formalized in a way compatible with this reasoner model.

Summarily, this reasoning model is computationally efficient (only requiring multiple runs of the reasoning engine when no compliant action is possible), but most likely excludes the implementation of some normative systems, necessitates significant preprocessing in the formation of the norm base for others, and leaves some general guarantees unproven. It moreover requires the designer of a normative system to configure how certain translations are done manually (should they differ from the default formalization used above).

Second Reasoner Model

We now introduce the second reasoner model. We begin again by formulating a precise definition of $A_C(s)$, and then discuss the suitable translation of constitutive norms over actions. We then delve into the Compliant Solutions algorithm, and give an example that shows that it renders the problems demonstrated in Example 5.2.3 obsolete. We then prove, as we did in the previous section, that the Compliant Solutions algorithm yields either an empty set or a compliant solution (according to our new formulation of $A_C(s)$). Next we present the Lesser Evil algorithm and prove that it picks those actions which minimize the number of violations incurred according to Definition 2.2.5. After a discussion on indirect normative conflicts and indirect permissions, we conclude our discussion on the second reasoner.

For a second reasoner model, we reform our definition of the set of compliant solutions $A_C(s)$ to better align with the more comprehensive definition of violation in Definition 2.2.5, which bases the notion of violation in the existence of some proposition (lit) that is obliged ($+\partial_O lit$), while the obligation is not fulfilled ($-\partial_C lit$).

$$A_C(s) = \{a \in A(s) \mid Th(\overline{s} = (s,a), \mathcal{N}) \nvdash +\partial_O lit, -\partial_C lit\} \tag{5.9}$$

In the expression above, $Th(\overline{s}, \mathcal{N})$ is obtained by adding a (or rather, the proposition asserting that a is performed) as a fact to $Th(s, \mathcal{N})$.

Here, the set of compliant actions in state s, $A_C(s)$, is the set of actions for which we cannot prove both $+\partial_O lit$ and $-\partial_C lit$ for some literal lit in $Th(\overline{s}, \mathcal{N})$ (that is, the actions that do not result in a scenario where we can prove $\mathbf{O}(lit)$ but not lit). In other words, for each action we consider whether the fact that that action was performed will result in a violation of the normative system as laid out in Definition 2.2.5. This contrasts with the first reasoner model in that Expression 5.7 makes it so that in order to exclude an action a from $A_C(s)$, we must be able to prove directly that a is forbidden (i.e., $+\partial_O \neg a$); Expression 5.9 does not require that we prove an action is itself forbidden, only that it will result in a violation if taken.

Note that, as was established in Proposition 2.3.3, $Th(\overline{s}, \mathcal{N}) \vdash +\partial_C \neg lit$ implies $Th(\overline{s}, \mathcal{N}) \vdash -\partial_C lit$, so the above definition will also exclude actions for which $Th(\overline{s}, \mathcal{N}) \vdash +\partial_O lit, +\partial_C \neg lit$ (that is, we can prove $\mathbf{O}(lit)$ and $\neg lit$) as well.

Constitutive Norms over Actions. For constitutive norms over actions of the form $\mathbf{C}(a_1, a_2 | c)$, we can simply use Expression 5.3 as their translation into DDL:

$$c, a_1 \rightarrow_C a_2$$

This is one way in which the second reasoner uses a more natural mode of reasoning, greatly simplifying how we translate constitutive norms over action; we will demonstrate the feasibility of this translation after we introduce Algorithm 5.3

The Compliant Solutions Algorithm. The second reasoner model provides a less efficient, but more comprehensive approach to discerning which possible actions are compliant. It utilizes the concept of violation outlined in Definition 2.2.5 and is predicated on the question, "if it is a fact that the agent takes action a, does it result in a violation of $\mathcal{N}$?" What we are doing, then, is evaluating all the possible scenarios that arise from the agent following any course of action. Algorithm 5.3 illustrates in pseudocode the reasoning behind this approach.

Algorithm 5.3: CompliantSolution2

    input : $Th(s, \mathcal{N}), A(s)$
    output: $conclusions, legalActions$

1 begin
2 $legalActions \leftarrow A(s)$;
3 $reasoner \leftarrow SPINdle.Reasoner$;
4 $conclusions \leftarrow \varnothing$;
5   **for** $a \in A(s)$ **do**
6 $Th(\overline{s}, \mathcal{N}) \leftarrow Th(s, \mathcal{N})$;
7 $Th(\overline{s}, \mathcal{N}).addFact(a)$;
8 $concl \leftarrow reasoner.generateConclusions(Th(\overline{s}, \mathcal{N}))$;
9 $conclusions.add(\langle a, concl \rangle)$;
10   **for** $lit \in Th(\overline{s}, \mathcal{N}).literals$ **do**
11   **if** $concl.has(+\partial_O lit) \wedge concl.has(-\partial_C lit)$ **then**
12 $legalActions.remove(a)$
13   **end**
14   **end**
15   **end**
16   **return** $conclusions, legalActions$
17 end

In Algorithm 5.3, for each possible action we amend $Th(s, \mathcal{N})$, adding as a literal a given action label, and then run the reasoning engine, storing the result of the computation in a map *conclusions* with the given action as the key. We then check, for each literal *lit* in $Th(\overline{s}, \mathcal{N})$, whether we can prove $\mathbf{O}(lit)$ but not *lit*; if so, then the given action results in a violation, and we exclude it from our list of compliant actions. In essence, the intuition behind this approach is Definition 2.2.5, in which we assert that a violation is something that is obligatory but not the case.

The way in which Algorithm 5.3 detects violations allows us more flexibility in which normative systems we can accommodate; below, we demonstrate how this reasoner model takes care of the problems introduced in Example 5.2.3.

Example 5.2.5. *Recall the second scenario in Example 5.2.3, where we have a regulative norm obliging b, translated as $r_1 : \Rightarrow_O b$, and two constitutive norms which we will translate to $r_2 : a \rightarrow_C b$ and $r_3 : a' \rightarrow_C b$. When we we cycle through actions to check for*

violations (lines 5-15 of Algorithm 5.3) we will each time be able to derive $+\partial_O b$, and when we add a as a fact, we derive from r_2 $+\partial_C b$. According to Proposition 2.3.1 we cannot have $-\partial_C b$, so a is not removed from $A_C(s)$. The same goes for a'. If we suppose there is a third action $a'' \in A(s)$, we will not be able to derive $+\partial_C b$, and will derive $-\partial_C b$ instead, so a'' will be excluded from $A_C(s)$.

We can get the same guarantees from Algorithm 5.3 that we did from Algorithm 5.1.

Theorem 5.2.2. *Algorithm 5.3 yields either an empty set or a compliant solution.*

Proof. For conditions (b) and (c) of Definition 5.2.1, recall that we have again identified a set of compliant solutions as $A_C(s) \subseteq A(s)$, so condition (b) is automatically fulfilled. Algorithm 5.3 removes every action a from $A(s)$ such that $Th(\overline{s},\mathcal{N}) \vdash +\partial_O lit, -\partial_C lit$ for some literal in the theory. According to Expression 5.9, in this model $A_C(s) = A(s) \setminus \{a \in A(s)|Th(\overline{s},\mathcal{N}) \vdash +\partial_O lit, -\partial_C lit\}$, so the algorithm yields only (and all) members of $A_C(s)$, fulfilling condition (c).

Again, since Algorithm 5.3 returns a set of single-action solutions (as RL agents can only select a single action to perform in a given state), these solutions are trivially consistent, fulfilling condition (d) of Definition 5.2.1. $\square$

As has already been mentioned, however, this reasoner model's conformance to our established definition of a violation comes with a computational cost.

Corollary 5.2.3. *Algorithm 5.3 can be completed in polynomial time with respect to* $|Th(s,\mathcal{N})| + 1$.

Proof. We refer back again to Theorem 2.3.1; $reasoner.generateConclusions(Th(\overline{s},\mathcal{N}))$ can be completed in linear time with respect to the size of the theory, which is the same as $|Th(s,\mathcal{N})| + 1$. Next, the checking over every literal in $Th(\overline{s},\mathcal{N})$ in lines 10-14 can also be completed in linear time. This is done $|A(s)|$ times, and since every action in $A(s)$ is a literal occurring in $Th(\overline{s},\mathcal{N})$, $|A(s)| < |Th(\overline{s},\mathcal{N})|$, and Algorithm 5.3 can be completed in polynomial time. $\square$

The Lesser Evil Algorithm. In the case that Algorithm 5.3 returns an empty set, we once more have an algorithm for selecting "lesser evil" actions, given below. Once again, we note that these actions compose the set $A_{NC}(s)$ associated with the $A_C(s)$ constructed by Algorithm 5.3; since this algorithm is run whenever $A_C(s) = \varnothing$, this procedure produces $A_{\mathcal{N}}$ as defined in Definition 5.1.1.

Algorithm 5.4 describes how we select minimally non-compliant actions in this framework; note that we need only input the *conclusions* map output by Algorithm 5.3, and it is notable that we do not need to run the reasoning engine again. Degrees of non-compliance are determined by the size of $viol(\overline{s},\mathcal{N})$.

Algorithm 5.4: LesserEvil2

input : *conclusions*
output : *bestActions*

1 **begin**
2   $scores \leftarrow \varnothing$;
3   **for** $act \in conclusions.getActions()$ **do**
4    $concl \leftarrow conclusions.getConcl(action = act)$;
5    $score \leftarrow 0$;
6   **for** $lit \in Th(\overline{s}, \mathcal{N}).literals$ **do**
7   **if** $concl.has(+\partial_O lit) \wedge \neg concl.has(+\partial_C lit)$ **then**
8      $score + +$;
9   **end**
10   **end**
11   **end**
12   $min \leftarrow min(scores)$;
13   $bestActions \leftarrow scores.getActions(score = min)$;
14   **return** $bestActions$
15 **end**

Theorem 5.2.3. *The actions in the set bestActions yielded by Algorithm 5.4 result in fewer violations (in the sense of Definition 2.2.5) of $\mathcal{N}$ than any other actions in $A(s)$.*

Proof. It is clear that the *score* computed in lines 7-11 of Algorithm 5.4 for each action is $|viol(\overline{s}, \mathcal{N})|$ (as it counts each time a violation according to Definition 2.2.5 can be derived from $Th(\overline{s}, \mathcal{N})$). Since we take only actions with a *score* equal to $min(scores)$, Algorithm 5.4 yields only actions that result in fewer violations than all other actions in $A(s)$. $\square$

We could not prove this generally for Algorithm 5.2, so this is another facet of the second reasoner that makes it a desirable alternative.

Since SPINdle has already been run for each action in Algorithm 5.3, we need only reference the map from action labels to conclusions that has already been generated. Thus, Algorithm 5.4 does not need to utilize SPINdle at all, and does no more than check elements in the map *conclusions* in lines 6-10; this is essentially a modification of lines 10-14 of Algorithm 5.3, where instead of removing actions we are updating the integer variable *score*.

Corollary 5.2.4. *Algorithm 5.4 can be completed in polynomial time with respect to $|Th(s, \mathcal{N})| + 1$.*

Proof sketch: We already established in the proof of 5.2.3 that lines 10-14 of Algorithm 5.3 can be completed in linear time; thus, so can lines 6-10 of Algorithm 5.4. Since this

is done $|A(s)|$ times, this algorithm can be completed in polynomial time with respect to $|Th(s, \mathcal{N})| + 1$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Indirect Normative Conflict.   It is not quite as simple to resolve indirect normative conflict in this framework, but it can still be done. Suppose we have two rules $r_1 : \Rrightarrow_O a_1$ and $r_2 : \Rrightarrow_O a_2$, where $a_1 \wedge a_2$ is impossible. Then if r_1 is of higher priority than r_2, we can add a constitutive rule to the normative system, $c_1 : a_1 \to_C a_1'$, along with a regulative rule $r_3 : \Rrightarrow_O a_1'$. Then, if action a_2 is taken, both a_1 and a_1' will be in $viol(\overline{s}, \mathcal{N})$, whereas if a_1 is taken, only a_2 will be in $viol(\overline{s}, \mathcal{N})$, resulting in Algorithm 5.4 choosing a_1 over a_2.

Indirect Permissions.   In the second reasoner model, we cannot use regular permissions to model indirect permissions, but there is nonetheless an easy solution.

Consider the scenario mentioned in the discussion on indirect permissions for the first reasoner; formalizing the scenario following the format discussed in Section 5.2.1, it would seem that we should have the rules:

$$red \Rightarrow_O \neg proceed$$

$$safe \Rightarrow_P right$$

$$\mathbf{P}(right) \rightsquigarrow_O right$$

$$right \to_C proceed$$

However, if we assume as facts red and $safe$, we will still be able to derive $\mathbf{O}(\neg proceed)$, meaning that if we do $right$, which counts as $proceed$, we will still be in violation, even though this action should be permitted. This is because between the fact that we perform $right$ and the rule $right \to_C proceed$, we can derive $+\Delta_C proceed$.

We can get around this problem by adjusting Algorithm 5.3 slightly. Instead of for each $a \in A(s)$ adding a as a fact, we can add $\Rightarrow_C a$. As a result, we can only derive $+\partial_C a$, and then if we want to permit a under condition c, we can alter the back-end translator and simply add $c \rightsquigarrow_C \neg a$ to the theory.

So, in the above scenario, we end up with:

$$red \Rightarrow_O \neg proceed$$

$$safe \Rightarrow_P right$$

$$\mathbf{P}(right) \rightsquigarrow_C \neg right$$

$$right \to_C proceed$$

Assessment of the Second Reasoner Model. Summarily, this framework is in general more computationally demanding than the first, but is extremely versatile. It does not require that we can prove explicitly the prohibition or obligation of particular actions, and has more general guarantees attached to it, in particular regarding the recognition of violations in the sense of Definition 2.2.5. Algorithm 5.3 was designed specifically to detect and react to violations conceived of in Definition 2.2.5, and we were able to show in Theorem 5.2.3 that Algorithm 5.4 will return the actions available to the agent that will result in the least such violations.

Additionally, this model allows for a more natural translation of constitutive norms over actions, relieving some of the burden of the preprocessing required in the first model, which can improve both ease of use and transparency.

5.2.3 Online Compliance Checking (OCC)

The supervisor was originally employed for online compliance checking in [NBCG21]; this entails, with each state transition, translating the agent's current state and the norms it is subject to into a theory of some logic for normative reasoning, and feeding it into a theorem prover, the output of which is parsed into a set of compliant solutions (or lesser evil solutions, if no compliant solutions exist), and sent to the agent. This effectively filters out non-compliant actions from the set of possible actions from which the agent can choose an optimal action. The process of OCC is depicted in Fig. 5.3.
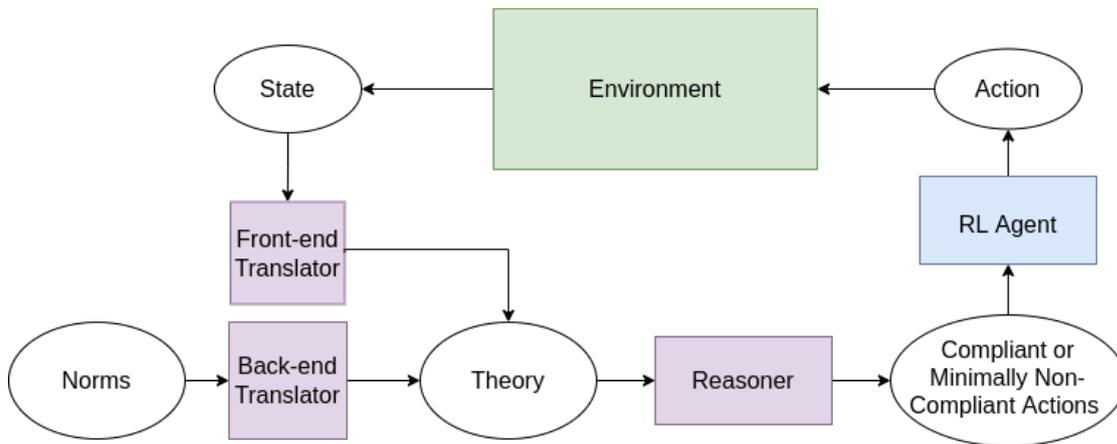
Figure 5.3: Purple boxes are the components of the normative supervisor.

In this technique, the normative supervisor simply sits on top of the RL agent, filtering out normatively sub-ideal actions in a manner similar to shields in pre-shielding [ABE+18]. The supervisor is completely decoupled from the agent, though, and does not interfere with the agent unless it observes the possibility an action that could potentially lead directly to a violation. Thus, Algorithms 5.1 (and possibly 5.2), or 5.3 (and possibly 5.4) must run at every time step, incurring significant cost over the course of operation.

Finally, it is worth noting that the normative supervisor is configured to output a *violation report* each time a violation is detected during OCC. This violation report contains a copy of the information that was sent to the supervisor from the agent (the current state) and the data returned to the agent (the lesser-evil solution). Because of this, whenever a violation occurs, we can reproduce it by individually submitting the violating state to the normative supervisor and observing the output – the resulting theory and conclusions derivable – facilitating *normative troubleshooting*.

5.3   Experiments and Evaluation

Below, we will walk through a wide variety of experiments performed through the three case studies introduced in Section 3.2. In the first, we see the normative supervisor, utilizing the second reasoner model (a necessary measure, as the first reasoner model cannot accommodate one of the normative systems employed), successfully applied to a Q-learning agent in a simple deterministic environment that can be modelled as a labelled MDP. In the second environment, the supervisor, utilizing the first reasoner model, is given the chance to operate in a simple but non-deterministic environment. In the final case study, the supervisor, once again utilizing the first reasoner model, is attached to a Q-learning agent with linear function approximation in a larger non-deterministic environment; here, a greater number and broader range of experiments is expounded upon. Though we utilize the first reasoner model in the last two case studies, it should be noted that it is possible to use the second reasoner model to implement all the normative systems we consider; however, since each set of experiments has 1000 games, the first reasoner model's efficiency is a useful asset.

5.3.1   Preliminary Experiments: the Merchant

Recall from Chapter 3 that the merchant is a resource-collecting game in an (optionally) deterministic environment. In this environment the agent, a merchant, must make its way from its home through a forest in which resources (ore from rocks, wood from trees) can be extracted. As in Chapter 4, we will consider games restrained by two different normative systems: the "Environmentally-Friendly Merchant" and the "Pacifist Merchant", which we saw back in Section 4.3.3. These normative systems are implemented for the second reasoner model because the computational cost should not be a problem in this small environment, and the second reasoner will allow us more natural translations of norms and – more importantly – allow us to oblige the agent to *negotiate* in the case of the pacifist merchant (this is not possible with the first reasoner model, due to the issues with constitutive norms over actions and obligations as demonstrated in Example 5.2.3).

Note that for the environmentally-friendly merchant, adjustments to the back-end translation have been made to accommodate indirect permissions, as discussed above.

The Environmentally-Friendly Merchant

In this normative system, it is mandated that the merchant acts in a way that is environmentally friendly, where the facts of what this entails are encoded into constitutive norms; the formalization in DDL is given below. Note also that we must add a rule asserting that if the agent does nothing, it is being environmentally friendly (that is, we can assume the agent is being environmentally friendly so long as it is not doing anything specifically designated as environmentally unfriendly).

1. $O(environment) : \Rightarrow_O environment$

2. $default : \Rightarrow_C environment$

3. $C(deforest, -environment) : deforest \rightarrow_C \neg environment$

4. $C(extract, deforest) : at\_tree, extract \rightarrow_C deforest$

5. $C(pickup, deforest) : at\_wood, pickup \rightarrow_C deforest$

6. $indirectP(pickup) : \neg has\_wood, at\_wood \Rightarrow_P pickup$

    - Automatically generated by translator: $genDefeat(pickup) : \mathbf{P}(pickup) \rightsquigarrow_C \neg pickup$

7. $indirectP(extract) : \mathbf{P}(pickup) \Rightarrow_P extract$

    - Automatically generated by translator: $genDefeat(extract) : \mathbf{P}(extract) \rightsquigarrow_C \neg extract$

The path generated by running the trained merchant along with the normative supervisor (configured for this normative system) is given in Figure 5.4.
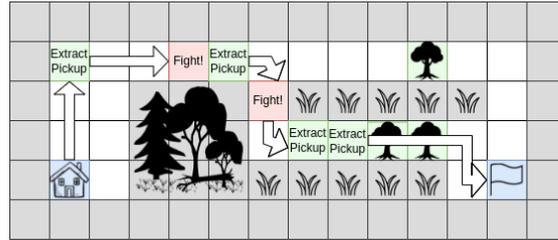
Figure 5.4: The environmentally friendly merchant's path.

As we can see, the merchant acts as expected, extracting and picking up the first piece of wood it comes across, while extracting and collecting the rest of the ore and ignoring all other trees on the map.

This normative system demonstrates a number of things. It demonstrates a form of strong permission, with a special characteristic; it is the indirect permission we described

in Section 5.2.2. The implementation of the environmentally-friendly merchant is a perfect demonstration of this form of strong permission in action, showing how DDL and the normative supervisor accommodate a wide variety of defeasible mechanisms for normative reasoning, an important component of normative resilience. It is also notable how the structure of constitutive norms – easily interpretable from their DDL translation – provides a completely transparent account of the obligation to be environmentally friendly and what exactly that means.

The Pacifist Merchant

In this normative system, the merchant is forbidden from entering dangerous areas (where it might be attacked); this is impossible, in the agent's environment, so the agent is bound to violate this norm. A second norm – a sub-ideal contrary-to-duty obligation – is in play, though: if the agent is in danger, it must negotiate (that is, give up its inventory).

1. $F(danger): \Rightarrow_O \neg at\_danger$

   - Since the normative supervisor restricts *actions*, we need to associate the state of being in danger to an action. Namely, *entering danger*: $F(enter\_danger): \mathbf{O}(\neg danger) \Rightarrow_O \neg enter\_danger$, with which we generate a default rule $default: \Rightarrow_C \neg enter\_danger$, along with:

     a) $C(North, enter\_danger): North\_danger, North \rightarrow_C enter\_danger$
     b) $C(South, enter\_danger): South\_danger, South \rightarrow_C enter\_danger$
     c) $C(East, enter\_danger): East\_danger, East \rightarrow_C enter\_danger$
     d) $C(West, enter\_danger): West\_danger, West \rightarrow_C enter\_danger$

2. $CTD: at\_danger, attacked \Rightarrow_O negotiate$

3. $C(unload, negotiate): attacked, unload \rightarrow_C negotiate$

The path generated by allowing the trained merchant to play the game under these constraints are shown in figure 5.5.
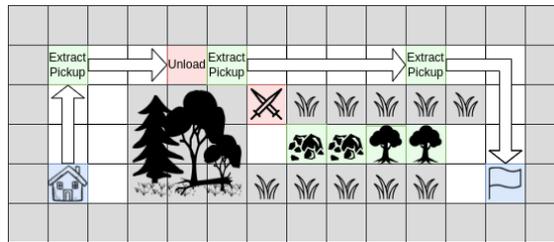
Figure 5.5: The pacifist merchant's path.

As we can see, the merchant acts how we would like: when given no choice but to pass through a dangerous area, it does so, and unloads its inventory (negotiates) when it is

attacked. Then, when given the choice between entering a dangerous area and repeating this strategy, and going the safe route, it chooses the safe route.

This normative system demonstrates the successful use of Algorithm 5.4 to deal with sub-ideal CTD obligations (as outlined in Section 2.2.2); this is precisely the element of normative resilience that was neglected by our compliance specifications approach from Chapter 4. While in Chapter 4, we failed to elicit behaviour in line with the pacifist CTD obligation (both in the full compliance and weak compliance case), this normative system was not a problem for the normative supervisor.

5.3.2   Introducing Stochasticity: Miniature Pac-Man

We now take a step forward and briefly explore some results gleaned from a simple non-deterministic environment – the miniature Pac-Man game (with the simplified maze environment) introduced in Section 3.2. We will examine two different normative systems – a *vegan* normative system, and a *benevolent* normative system – which, though formulated differently, should elicit the same behaviour.

This will be a demonstration of the first reasoner model in action, exhibiting the kind of pre-processing that needs to be applied to the norm base in order for the normative supervisor to function appropriately; this pre-processing and, more generally, the process of constructing a norm base from scratch will be shown in detail when we lay out the norm base for vegan Pac-Man. Again, due to the number of game states and the amount of games played, the more efficient reasoner model will suit our purposes well.

It is notable that this case study will serve to show one of the weaknesses of online compliance checking; the agent will be somewhat crippled by the normative supervisor, and less effective at accomplishing its primary goal. This is likely due to the small environment we use here, where there are few safe places in the environment, and thus the agent has more chances at every time step to take the wrong action and lose the game. We will see, once we start working with a larger environment, this disability disappears entirely.

Vegan Pac-Man

In the miniature Pac-Man game described in Section 3.2 (see page 42), Pac-Man shares its maze with a ghost that can eat Pac-Man, or be eaten by Pac-Man if Pac-Man first eats a power pellet. When Pac-Man learns to play this game, it will learn to eat the ghost whenever possible, because this wins more points. However, in the first normative system we look at, we demand that Pac-Man be *vegan*: "do not eat ghosts".

If this normative system is to constrain our agent's actions, it needs to reference concepts that correspond to the information processed by the agent, which is limited to the locations of game entities and the actions that Pac-Man can perform, which are *North*, *South*, *East*, *West*, and *Stop*. The question then becomes one of how we get from the

comparably abstract norm above to these lower-level state and action descriptions. We will need to fill in the gaps, exercising our knowledge of the game mechanics to do so.

We start at the top, at the abstract regulative norm $\mathbf{F}(eat_{ghost}|\top)$; the only way eat_{ghost} can be done is if (a) the ghost is in a 'scared' state, and (b) Pac-Man and the ghost move into the same cell. We can express these concepts as predicates over game objects, specifically as (a) $scared_{ghost}$ and (b) $inRange_{pacman,ghost}$. Pac-Man does not know which direction the ghost will move in, but we will assume a "cautious" model of action where Pac-Man should not perform any action that *could* constitute eating a ghost; that is, if Pac-Man takes an action that could reasonably lead to it violating a norm, we will consider that norm violated. Since Pac-Man's next action determines what is in range, we in fact need five entities to express $inRange_{pacman,ghost}$, one corresponding to each action. These conceptual entities are used to construct a constitutive norm, or a kind of strategy, regarding the action of eating. For example:

$$\mathbf{C}(North, eat_{ghost}|scared_{ghost} \wedge inNorthRange_{pacman,ghost})$$

To define $inNorthRange_{pacman,ghost}$, we note that we have access to the positions of Pac-Man and the ghosts, so we can create another set of constitutive norms which are applicable in the context $pacman_{i,j}$:

$$\mathbf{C}(ghost_{k,l}, inNorthRange_{pacman,ghost}|pacman_{i,j})$$

where (k,l) has a Manhattan distance of 1 from $(i, j+1)$.

In this case, the set F_s of facts contains literals representing the locations of Pac-Man and the ghosts. It may also contain other facts about the game; e.g., if there is a scared ghost, both its location and $scared_{ghost}$ will be included in F_s of $Th(s,\mathcal{N}) = \langle F_s, R_s^O, R_s^C, > \rangle$.

How does the back-end translator translate these norms? The original prohibition is simple:

$$vegan : \Rightarrow_O \neg eat_{ghost} \in R_s^O$$

The constitutive norms over states are similarly simple as well, and are generated as was discussed in Section 5.2.1; whenever $pacman_{2,1}$ is true the rule(s) defining $inNorthRange_{pacman,ghost}$ will include, e.g.:

$$pacman_{2,1}, ghost_{1,2} \rightarrow_C inNorthRange_{pacman,ghost} \in R_s^C$$

The translation of constitutive norms over actions will generate the following DDL rules:

$$scared_{ghost}, inNorthRange_{pacman,ghost}, \mathbf{O}(North) \Rightarrow_O eat_{ghost} \in R_s^O$$

and

$$scared_{ghost}, inNorthRange_{pacman,ghost}, \mathbf{O}(\neg eat_{ghost}) \Rightarrow_O \neg North \in R_s^O$$

and

$$scared_{ghost}, inNorthRange_{pacman,ghost}, \mathbf{O}(North) \rightarrow_C eat_{ghost} \in R_s^C$$

We can be assured that this formalization yields a consistent theory by Definition 2.3.4.

Proposition 5.3.1. *The defeasible theory $Th(s, \mathcal{N})$ is consistent and $\mathbf{O}$-consistent.*

Proof. Since F_s contains only propositions asserting the locations of Pac-Man and the two ghosts, as well as $scared_{ghost}$ if a ghost is scared, there can be no pairs of complementary literals. There are no rules in R_s^C with conflicting consequents, so the superiority relation is empty, and trivially acyclic. Moreover, R_s^O likewise does not contain any rules with complementary consequents, and the superiority relation is again trivially acyclic. So $Th(s, \mathcal{N})$ is both consistent and O-consistent. $\qquad\qquad\square$

Now that we have $Th(s, \mathcal{N})$, we can play the miniature Pac-Man game while running the normative supervisor with the *vegan* norm base.

% Games Won	Avg Game Score	Avg Ghosts Eaten
46.6%	25.22	0.0

Table 5.1: The result of playing 1000 games while running the normative supervisor.

Note that no ghosts are eaten, but the game performance suffers significantly; recall that the average score in the baseline test was 441.71, and 68.5% of games were won (see Table 3.2 on page 43). In the miniature Pac-Man environment, both average score and percentage of games won were greatly reduced with the addition of online compliance checking. Clearly, the normative supervisor is effective at producing compliant behaviour even in stochastic environments; however, it has the potential to greatly hamper the agent's effectiveness with respect to its primary goal.

Benevolent Pac-Man

We move on to a normative system built around the imposition of the "duty" of "benevolence" onto Pac-Man. This duty is represented by the norm $\mathbf{O}(benevolent|\top)$, which is translated to

$$O(benevolent) : \Rightarrow_O benevolent$$

which means that Pac-Man should be benevolent at all times. This introduces the question of what it means for Pac-Man to be benevolent. We do not have an explicit definition, but we have a description of non-benevolent behaviour: $\mathbf{C}(eat_{person}, \neg benevolent|\top)$, which is an constitutive norm over actions translated to

$$C(person)1 : \mathbf{O}(eat_{person}) \Rightarrow_O \neg benevolent \ \&$$
$$C(person)2 : \mathbf{O}(benevolent) \Rightarrow_O \neg eat_{person} \ \&$$
$$C(person)3 : \mathbf{O}(eat_{person}) \rightarrow_C \neg benevolent$$

where eat_{person} is a description of the act of eating an entity designated as a person. That is, we know that Pac-Man is not being benevolent if it eats a person. Additional constitutive norms could expand our definition of benevolence.

We next add a second constitutive norm, $C(eat_{ghost}, eat_{person})$, translated as

$$C(ghost, person)1: \ \mathbf{O}(eat_{ghost}) \Rightarrow_O eat_{person} \ \&$$
$$C(ghost, person)2: \ \mathbf{O}(\neg eat_{person}) \Rightarrow_O \neg eat_{ghost} \ \&$$
$$C(ghost, person)3: \ \mathbf{O}(eat_{ghost}) \rightarrow_C eat_{person}$$

asserting that eating a ghost counts as eating a person.

We can remove *vegan* from R_s^O of $Th(s, \mathcal{N})$ and add the above seven rules, creating a new normative system that provides a direct link between the notion of benevolence and the behaviour that involves eating a ghost.

This normative system produces behaviour identical to the one in the subsection above – that is, it manifests as a prohibition from eating the ghost – the difference lies in how it is formalized, using a more abstract set of concepts to govern behaviour. This normative system illustrates how the normative supervisor can provide a clear documentation of the reasoning that occurs when the agent acts within its environment; had there been a violation, the normative supervisor was configured to output a violation report, with which we could have reconstructed the state recorded in the report and viewed directly the theory that was being reasoned about when the agent made this error. There we would have had a transparent picture of the facts that hold in that state and which rules will be triggered, illustrating the trespassing of the obligation for benevolence and providing an accountability link between what is true in the state and the obligation for benevolence. We will get to experiment with this directly in the ensuing section, where we walk through the main experiments for the normative supervisor.

5.3.3   Main Experiments: Regular Pac-Man

The experiments we can do in a small non-deterministic environment like the miniature Pac-Man game are limited; we get only a rather bland picture of online compliance checking with the normative supervisor, where we elicit simple normative behaviours successfully, albeit at a cost. We are able to explore a broader variety of normative systems and encounter more colourful predicaments once we move to the regular Pac-Man game, as played out in [NBM+19, NBCG21, NBCG22]. We will expand our investigation significantly here, and attempt to cover many of the demands of normative resilience with a series of experiments geared toward demonstrating the capabilities of the normative supervisor[3]. We start by revisiting the vegan Pac-Man normative system we saw in the previous subsection, and explore normative deadlock a bit more; first, we will analyze the nature of the violations that occurred over these games, before moving on to a variation of the vegan normative system where we show indirect conflict resolution. After, we demonstrate the addition of a permission to get the "vegetarian Pac-Man" normative system, where we encounter some unforeseen complications, for which we will introduce some new rules. Afterwards, we encounter direct normative conflict in the "hungry

---

[3]Most of these experiments are taken from [NBCG22]

vegetarian" normative system. Then, we will demonstrate how the rules introduced for vegetarian Pac-Man can be used to show that the agent will only select actions in $A_{\mathcal{N}}(s)$, even at the expense of losing every game it plays. Finally we examine a normative system with a CTD obligation, the "passive vegan" system.

Note that all of these normative systems utilize the same constitutive norms as were laid out in the last subsection, and we will not review them below.

Vegan Pac-Man

As a first step, we revisit "vegan" Pac-Man. Recall that in the regular Pac-Man game, there are two ghosts (one blue and one orange) that we need to account for. So a first step will be to add some constitutive norms $\mathbf{C}(eat_{blueGhost}, eat_{ghost})$ and $\mathbf{C}(eat_{orangeGhost}, eat_{ghost})$ to the vegan normative system. These are translated into DDL, same as the constitutive norms over actions in the previous section, and then added to $Th(s, \mathcal{N})$.

With these additional norms in place, we ran three sets of tests on Vegan Pac-Man, on a random agent, one with the *safe* policy, and one with the *hungry* policy (see Chapter 3 for the details of how these agents were trained).

Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
random	0	-449.13 [-166]	0 / 0.003
safe	91.4	1216.67 [1547]	0.005 / 0.015
hungry	90.7	1209.86 [1708]	0.023 / 0.02

Table 5.2: Vegan Pac-Man with three different agents.

The results of the tests with RL agents are comparable to [NBM+19], where Pac-Man was trained to behave in compliance with the *vegan* norm; in that paper the authors ran 100 games, obtaining an average of 0.03 ghosts eaten per game, while our approach averaged at 0.02 or 0.043 depending on the policy; similarly, their score averaged at 1268.5, while ours averaged 1216.67 and 1209.86. With respect to our baseline tests, the performance of Pac-Man – with respect to % games won and score – did not suffer; there was of course a decrease in score for the *hungry* policy Pac-Man, since the up to 800 points it could win by eating ghosts are no long available to it. The score for the *safe* policy with and without the normative supervisor did not meaningfully change, and % of games won actually increased by nearly an entire percentage point.

Due to the layout of the maze, it is possible for Pac-Man to encounter situations where it is not possible to refrain from eating a ghost; e.g., Pac-Man can be trapped in between two scared ghosts, resulting in normative deadlock. If we examine the violation reports generated by the infractions shown in the above data, we can confirm that every case of eating a ghost was due to such a case of normative deadlock. We will explore these scenarios in the sub-section below.

Remark. *Due to the low complexity of the logic used and the modest size of $Th(s, \mathcal{N})$ theory – which rarely exceeded 50 rules – SPINdle took on average 1.1 ms (max 97 ms) in generating conclusions during the Vegan Pac-Man tests.*

The evaluation of non-compliant solutions, in the rare cases where it was required, took 45.6 ms on average (min 15 ms, max 114 ms). For a detailed analysis on the performance of SPINdle, see [LG09].

Analysing Violations.   Inherent to Pac-Man's environment is the possibility of encountering a state where no compliant action is possible; if Pac-Man encounters such a state, it is forced to violate the normative system. As we have noted, when the normative supervisor identifies these situations – that is, Algorithm 5.1 returns an empty solution set – we have configured it to store a description of them. Included in this description is a list of possible actions and the positions of all agents in the game; from this information we can reconstruct the circumstances in which Pac-Man took a non-compliant action. For vegan Pac-Man in particular, an examination of these records makes it clear that the vast majority of violations took the form described in Fig. 5.6(a) below, where every
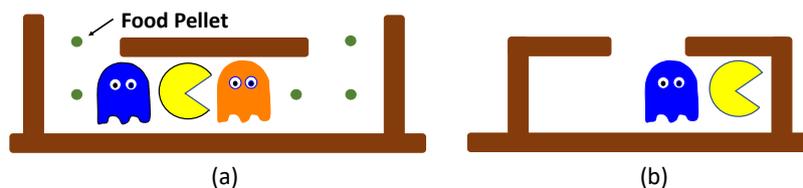
Figure 5.6: Pac-Man trapped between two ghosts (a) or in a corner (b).

direction Pac-Man is able to move in is in the trajectory of a nearby scared ghost. The only exception we found is depicted in Fig. 5.6(b); this situation was rare, only occurring once in two thousand games. We can, in fact, prove that these two types of scenario are the only cases where Pac-Man will be forced to violate *vegan*.

Proposition 5.3.2. *The cases depicted in Fig. 5.6 are the only cases in the vegan Pac-Man game where no compliant solution is possible.*

Proof. The absence of a compliant solution means that for every $a \in A(s)$, $Th(s, \mathcal{N}) \vdash +\partial_O \neg a$; that is, every action is forbidden.

Note that Pac-Man always has the action *Stop* available to it, but there are no game states in which its set of possible actions is $[Stop]$ – this would imply that Pac-Man is closed in on all sides by walls. However, we can have the set of possible actions $[Stop, Dir]$ where $Dir \in \{North, South, East, West\}$. This would imply that there are walls on all sides of Pac-Man, aside from the cell it can move to in taking action Dir. If a ghost is in range of Pac-Man in direction Dir, we can derive $+\partial_O \neg Dir$, and if the ghost is directly beside Pac-Man, we can derive $+\partial_O \neg Stop$; this is the exact scenario depicted in Fig. 5.6(b).

A second possibility is that the set of possible actions is [$Stop$, $Dir1$, $Dir2$]; $Dir1$ and $Dir2$ can be any pair of directions. This can describe corners in the maze (of which there are 16), or the the "tunnel"-like portions on of the maze. In all of these cases, it is possible for one ghost to occupy one of the two spaces Pac-Man is free to move into, which is described in Fig. 5.6(a). In this case, we would be able to derive $+\partial_O \neg Dir1$ and $+\partial_O \neg Dir2$.

For the remaining two possibilities, where Pac-Man has a list of possible actions of the form [$Stop$, $Dir1$, $Dir2$, $Dir3$] or [$Stop$, $Dir1$, $Dir2$, $Dir3$, $Dir4$], at least one direction in which it is possible to move will not be blocked by a ghost, since there are only 2 ghosts. As Pac-Man will be permitted to move in this direction, we cannot have a case where no compliance is possible. □

The violation reports allow us to reproduce violations; we can feed the state data into the normative supervisor individually and inspect how $Th(s, \mathcal{N})$ is translated and view the conclusions generated by inputting it to SPINdle. This data can tell us both where the violation originated and when and how similar violations can occur. This short investigation into what violations of the vegan normative system occurred is demonstrative of several factors. It is made evident how the violation reports provide a mechanism both for accountability and transparency of decisions made by the agent, and for aiding in our construction of guarantees regarding the limitations of the agent.

Preferences over Violations: Unfair Vegan Pac-Man

In the above vegan Pac-Man experiments, we saw several cases of Algorithm 5.2 being used. In all of these cases, each possible action will have been assigned the same score; it is equally errant to eat a blue ghost and an orange ghost. But what if we put more weight on the life of the blue ghost, so that given the choice our agent should choose to eat the orange ghost instead of the blue ghost?

As we noted, the first reasoner model provides a very straightforward mechanism for achieving this. All that must be done is to add an additional norm forbidding the eating of the blue ghost:

$$preference : \Rightarrow_O \neg eat_{blueGhost}$$

With this additional rule, when Algorithm 5.2 is triggered in the presence of normative deadlock, it will return the action that leads to the lesser infraction. Below are the results of 1000 games under the constraint of this "Unfair Vegan" normative system.

Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
hungry	90.6	1207.77 [1537]	0.002 / 0.034

Table 5.3: Vegan Pac-Man where it is preferred that orange ghosts are eaten.

We can see that for the most part, Pac-Man only consumed orange ghosts; there were two exceptions, however. An examination of the resulting violation reports shows that these infractions occurred in the situation described in Figure 5.6(b).

This normative system allows us to truly take advantage of the Lesser Evil algorithm when the agent runs into normative deadlock; with the addition of the rule $preference$, we can express a clear prioritization of keeping the blue ghost uneaten. This is not the same as indirect normative conflict resolution, strictly speaking; if this were a true indirect conflict resolution scenario, eating the orange ghost would not count as a violation. However, in such a normative system, a nominal indication of compliance would not change the agent's behaviour, and we argue that these experiments provide effective evidence for normative resilience on the norm conflict front.

Revising the Norm Base: Vegetarian Pac-Man

We now turn to a more permissive normative system – one where only eating the blue ghost is forbidden. There are two obvious ways to amend the vegan normative system in order to elicit this behaviour.

For the first, we can *contract* the norm base by removing $C(orange, ghost)1$, $C(orange, ghost)2$, and $C(orange, ghost)3$, so that only blue ghosts count as a ghost – which Pac-Man is prohibited from eating. We have another option, though; we can *expand* the normative system and add the permissive norm $\mathbf{P}(eat_{orangeGhost}|\top)$. Then the back-end translator generates two rules and adds them to R_s^O in $Th(s, \mathcal{N})$:

$$P(orange) : \Rightarrow_P eat_{orangeGhost} \ \& \ default(orange) : \mathbf{P}(eat_{orangeGhost}) \rightsquigarrow_O eat_{orangeGhost}$$

These two normative systems produce identical behaviour; indeed, we can see them as basically the same, since the universal permission applied to $eat_{orangeGhost}$ in the second effectively counteracts $C(orange, ghost)2$ whenever it is triggered.

In the experiments discussed below, note that we have run the normative supervisor with both normative systems, achieving the same results; nevertheless, we will walk through the results only once.

To start, we will look at some preliminary results – based on 100 games and trained over 100 episodes – which are given below.

Norm Base	Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
vegetarian (100 games)	*hungry*	94	1413.8 [1742]	0.01 / 0.79

Table 5.4: Preliminary tests with vegetarian Pac-Man, trained for 100 episodes.

For Vegetarian Pac-Man, the scenario in Fig. 5.6(a) cannot occur, but one blue ghost was still eaten. Proposition 5.3.2 tells us that there is exactly one other case where Pac-Man can be forced into non-compliance, but our investigation of the records capturing the

above test (or rather, the lack of records for this incident) yielded a second way in which Vegetarian Pac-Man can end up eating a blue ghost – without directly or knowingly violating its norm base. That is, Pac-Man (see Fig. 5.7) is put in a situation where there *is* at least one available action that will not result in it eating the blue ghost, but taking the action that results in eating the ghost does not, strictly speaking, violate any norms in Pac-Man's norm base.
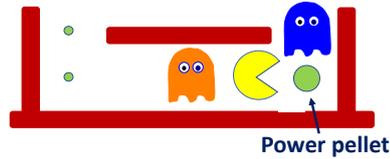
Figure 5.7: Pac-Man about to enter the same space as the blue ghost.

When both Pac-Man and the blue ghost move into the power pellet's cell at the same time, Pac-Man ends up eating it. This is a quirk in the game implementation; when generating the next state, the game changes the ghost's state to 'scared' immediately after Pac-Man moves, but before it is determined whether Pac-Man eats the ghost or the ghost eats Pac-Man. This is roughly analogous to an agent committing to an action it believes is compliant, which nevertheless ends up having morally or legally negative consequences because the agent's circumstances changed after they had already committed to the action. This curious edge case will make a reappearance in the next chapter, in Section 6.1.2.

For Vegetarian Pac-Man, the cases depicted in Fig. 5.6(b) and Fig. 5.7 represent the scenarios where it ends up eating blue ghosts; we can encounter non-compliance, albeit very infrequently. If we want to avoid such cases, however, we can again *expand* our norm base to prevent Pac-Man from entering such "dangerous" situations. To do that we introduce a number of constitutive rules defining the concept of "danger" for each coordinate (x, y) within range of a hazardous area, for example:

$$danger_{pacman} : pacman_{x,y} \rightarrow_C inNorthRange_{pacman,danger}$$

We will also need to indicate whether there is a ghost within range of this "danger zone", as avoiding it is only necessary if there is a ghost nearby:

$$danger_{ghost} : ghost_{x,y} \rightarrow_C inRange_{ghost,danger}$$

The regulative norm we impose on Pac-Man prevents it from "entering danger":

$$avoid : \Rightarrow_O \neg enter_{pacman,danger}$$

What does it mean to "enter danger"? As we did previously, we can define strategies (constitutive norms) that describe what constitutes entering danger. Namely, the following constitutive norm holds:

$$\mathbf{C}(North, enter_{pacman,danger}|inNorthRange_{pacman,danger}, inRange_{ghost,danger})$$

. So we can define the appropriate rules, including:

$$C(danger): inNorthRange_{pacman,danger}, inRange_{ghost,danger}, \mathbf{O}(\neg enter_{pacman,danger})$$
$$\Rightarrow_O \neg North$$

We designate the version of Pac-Man subjected to these additional norms with the tag "avoid". A summary of the performance of vegetarian Pac-Man during preliminary tests with this addition is given below; notice that the additional norms led to full compliance.

Norm Base	Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
vegetarian – avoid (100 games)	*hungry*	87	1336.2 [1747]	0.00 / 0.88

Table 5.5: Vegetarian Pac-Man with the *avoid* rule, once again trained with 100 episodes.

Remark. *[GR08b] discusses situations where the combination of the (ethical) norms and a particular factual situation results in breaches even for agents designed to comply with the norms at the cost of giving up their goals. There it is proved that sometimes the only way to comply with the norms is to prevent a situation from happening, and the agents have to modify their plans accordingly. The solution adopted currently is to introduce new norms that render a position in the game that unavoidably results in a violation, forbidden. This is not a comprehensive solution; we might not always be able to design coherent or feasible rules that can avoid such dangerous situations; for this reason, an additional mechanism for planning ahead to avoid violations would be desirable.*

When we move on to running batches of 1000 games, we find that neither the scenarios in Fig. 5.6 nor the one in Fig. 5.7 occur, and we have full compliance even without implementing the *avoid* rules:

Norm Base	Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
vegetarian	*random*	0	-446.96 [-111]	0.00 / 0.008
vegetarian	*hungry*	89.3	1343.8 [1750]	0.00 / 0.79
vegetarian – avoid	*hungry*	90.1	1361.06 [1751]	0.00 / 0.81

Table 5.6: Vegetarian Pac-Man with 250 episodes of training, over 1000 games.

These relatively identical results are likely due to Pac-Man's additional training for these tests; for example, the region of the maze depicted in Fig. 5.6(b) contains no food pellets, and it is not necessary to enter it in order to win the game. This suggests that running preliminary tests with a more poorly performing agent can prove a useful tool for better understanding how the normative system, the agent, and its environment interact, an interesting insight.

Nevertheless, the additional rule *avoid* does not hamper performance and supplies an additional degree of security, and in certain applications of ethical or legal AI, we might

be interested in having a guarantee that a violation scenario, however rare, will not be allowed to occur.

These experiments have demonstrated how the normative supervisor accommodates mutating normative systems, allowing us to add or subtract norms as needed, providing evidence of normative resilience. It also demonstrates the success of this framework at dealing with strong permission, which is an element of the normative conflict dimension of normative resilience.

Direct Normative Conflict: Hungry Vegetarian Pac-Man

Now suppose that Pac-Man is not just *permitted* to eat orange ghosts; perhaps eating orange ghosts is encouraged. Namely, we add the rule:

$$conflict: \ inRange_{pacman,orangeGhost} \Rightarrow_O eat_{orangeGhost}$$

That is, when Pac-Man is within range of an orange ghost, it is obligated to eat that orange ghost. This introduces a problem: $Th(s,\mathcal{N})$ is no longer O-consistent. Thus, we must add a superiority relation to the theory:

$$conflict > C(orange,ghost)2$$

meaning that the rule *conflict* takes precedence over the rule $C(orange,ghost)2$.

When we run the resulting normative system, we get the following numbers:

Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
hungry	89.3	1343.50 [1750]	0.00 / 0.034

Table 5.7: Vegetarian Pac-Man, when it is obligatory that if Pac-Man is near an orange ghost, it eats it.

This expansion of the vegan normative system allows us to demonstrate how the normative supervisor can handle direct normative conflict, using DDL's superiority relation. Clearly, this approach demonstrates this element of normative resilience without much difficulty.

Revising the Norm Base: Cautious and Over-Cautious Pac-Man

Returning to the vegan normative system, we can use the *avoid* rule to help Pac-Man avoid eating ghosts altogether: by not eating the power pellet which makes the ghosts scared in the first place. We call this normative system 'cautious'; meanwhile, we call the normative system that over-corrects for eating pellets by forbidding Pac-Man from entering entire *regions* of the maze 'over-cautious'.

For the 'cautious' norm base, we will have the *avoid* rule:

$$avoid: \ \Rightarrow_O \neg enter_{pacman,danger}$$

along with the constitutive norms:

$$danger_{pacman} : pacman_{x,y} \rightarrow_C inNorthRange_{pacman,danger}$$

where (x, y) is one step away from the area defined as dangerous, and the appropriate rules for constitutive norms over actions, including:

$$inNorthRange_{pacman,danger}, \mathbf{O}(\neg enter_{pacman,danger}) \Rightarrow_O \neg North$$

for all actions $\in A(s)$.

We designed two different normative systems; in one (denoted as *cautious* in the below table of results), we define the cells holding the power pellets as dangerous. In the other, we take a deliberately over-cautious approach, and define entering a dangerous area as entering an entire region of the maze; in particular, we define as dangerous certain 3×4 regions of the maze in which Pac-Man can find the power pellets. However, this makes the food pellets in these regions also inaccessible, and as a result, Pac-Man cannot win the game if it obeys *avoid*. The results of testing the vegan norm base with these *cautious* and *over-cautious* modifications of the *avoid* rule are given below.

Norm Base	Policy	% Games Won	Game Score (Avg [Max])	Avg ghosts eaten (Blue / Orange)
vegan – cautious	*hungry*	13.9%	28.36 [1340]	0.00 / 0.00
vegan – over-cautious	*hungry*	0%	-174.95 [176]	0.00 / 0.00

Table 5.8: Cautious and over-cautious Pac-Man.

As we might expect from the over-cautious norm base, Pac-Man is compelled to obey the norm base at the expense of winning the game, and loses every game it plays. We proved in Section 5.2.2 that Algorithm 5.1 returns either compliant solutions or the empty set (which returns in the presence of normative deadlock); we further proved that normative deadlock can only occur in the situations described in Figure 5.6, which cannot occur with either of the above normative systems (the ghosts are never scared, because Pac-Man never eats the power pellets). The fact that the normative supervisor will not allow an agent non-compliant actions (unless they cannot be avoided) even at the expense of failing to fulfil its primary function will be desirable quality in some cases, but this might not always be the case.

In the case of the cautious normative system, Pac-Man can, in theory, win the game (the power pellets – which are not technically food pellets – do not need to be eaten to win the game), but fails to do so in most cases. There are two reasons for this: the first is that this creates eight additional places in the maze where a scenario like that depicted in Fig. 5.6(b) can occur. This leads to additional opportunities for Pac-Man to become trapped by the ghosts, and since complying with the normative system takes precedence over the policy, Pac-Man will choose not to eat the power pellet even if it will lose the game as a result. The second reason why Pac-Man's success rate is so low is

due to a shortcoming of our decoupled approach; Pac-Man's policy does not take into account the fact that Pac-Man cannot continue moving in that direction. In this case, the supervisor filters out the best choice available, and Pac-Man has to go with the second best action – which, in some cases, might be *Stop*. In this case, Pac-Man will remain stuck in place until conditions in the game change enough for the policy to start recommending another action, and this may never get the chance to happen, if Pac-Man is trapped and eaten first. This particular scenario represents a good argument for combining the normative supervisor with additional methods for merging goal-directed behaviour and norm-compliant behaviour, for example by learning both simultaneously.

Contrary-To-Duty Obligations: Passive Vegan Pac-Man

We now turn to contrary-to-duty obligations. We will address a line of reasoning similar to Forrester's paradox [For84] (i.e., the gentle murder paradox; see page 28) in this section; that is, we modify the norm base of Vegan Pac-Man by adding the following rule (we will call this Passive Vegan Pac-Man): "If Pac-Man *does* eat a ghost, it must do so while standing still (taking the action *Stop*)."

The intricacies of this dynamic are not easy to demonstrate with the kinds of test results with which we have hitherto summarized Pac-Man's behaviour. Thus, we will here walk through a kind of case-study-within-a-case-study, illustrating how the introduction of this rule (as a sub-ideal CTD obligation) changes the results output by Algorithm 5.2 in a non-compliant scenario. In particular, we take a real example of a non-compliant scenario from a violation report from one of the Vegan Pac-Man tests (depicted in Fig. 5.8). In

```
{
"request": "FILTER",
"possible": ["West", "Stop", "East"],
"reasoner": "DDPL",
"game": {"blue_eaten": 0,
         "orange_eaten": 0,
         "score": 650,
         "layout": [{"position": {"y": 1, "x": 2}, "type": "p"},
                    {"position": {"y": 2.0, "x": 1.0}, "type": "sc_b"},
                    {"position": {"y": 1.0, "x": 3.0}, "type": "sc_o"}],
         "dimension": {"y": 11, "x": 20}},
"norms": "vegan",
"id": 277
}
```

Figure 5.8: The state that was passed from the agent to the supervisor when a violation of the vegan norm base was incurred. Note that 'p', 'sc_b', and 'sc_o' designate the corresponding positions as the location of Pac-Man, the scared blue ghost, and the scared orange ghost respectively.

this scenario, there is one ghost directly to the east of Pac-Man, and one just around the corner to the northwest. The output of Algorithm 5.2 for the scenario outlined in Fig. 5.8, under the vegan norm base, is shown in Fig. 5.9:

In the above result, Pac-Man is not compelled to take any action in particular; each move it can make will be equally bad at this point.

113

```
No compliant actions. Locating maximally compliant action...
{"response":"RECOMMENDATION","compliant":false,"actions":["Stop","West","East"]}
```

Figure 5.9: Result passed back to agent according to the Vegan norm base.

However, we now formalize the above stated contrary-to-duty obligation, introducing an additional norm to the norm base:

$$ctd: \ eat_{pacman,ghost} \Rightarrow_O Stop$$

After the addition of the rule *ctd*, the output of Algorithm 5.2 changes, as shown in Fig. 5.10. Now, the only action recommended is *Stop* – in compliance with the contrary to

```
No compliant actions. Locating maximally compliant action...
{"response":"RECOMMENDATION","compliant":false,"actions":["Stop"]}
```

Figure 5.10: Result passed back to agent after the addition of the rule *ctd*.

duty obligation *ctd*. This change occurs because now, in addition to violating *vegan*, the choices of *West* and *East* violate *ctd* as well. Thus, while the scenario is still non-compliant (and therefore a violation report would be generated), the agent receives an instruction in line with its contrary-to-duty obligation.

This case demonstrates how the first reasoner model can accommodate norm violation, as manifested in normative deadlock and contrary-to-duty obligations, confirming that an important facet of normative resilience is managed by this framework.

5.4 Final Remarks

In the preceding sections, we have reviewed the problem at hand, introduced the proposed approach (normative interference), explained the normative supervisor and its function for online compliance checking, and evaluated the normative supervisor's performance in a number of case studies.

Normative interference involves accessing an agent's internal processes and modifying the output of one of the modules that compose these processes, resulting in the modification of the agent's behaviour in a way that conforms to a normative system. In this chapter, we looked at online compliance checking, a naive form of normative interference that filters non-compliant actions out of the agent's arsenal before it selects an action, at time of operation. The normative supervisor in this way can be almost completely decoupled from the agent while still limiting its behaviour.

This approach is straightforward and easy to parse – but does it *work*? With respect to transparency, the fact that the supervisor can generate $Th(s, \mathcal{N})$ for any state and normative system (see the example in Figure 5.11) goes a long way for promoting transparency and facilitating accountability. Within $Th(s, \mathcal{N})$, the set of constitutive norms

```
Fact (29):
    [East_danger]        >> -East_danger(X)
    [East_ore] >> -East_ore(X)
    [East_rock] >> -East_rock(X)
    [East_tree] >> -East_tree(X)
    [East_wood] >> -East_wood(X)
    [North_danger]       >> -North_danger(X)
    [North_ore] >> -North_ore(X)
    [North_rock]         >> -North_rock(X)
    [North_tree]         >> -North_tree(X)
    [North_wood]         >> -North_wood(X)
    [South_danger]       >> -South_danger(X)
    [South_ore] >> -South_ore(X)
    [South_rock]         >> -South_rock(X)
    [South_tree]         >> -South_tree(X)
    [South_wood]         >> -South_wood(X)
    [West_danger]        >> -West_danger(X)
    [West_ore]  >> -West_ore(X)
    [West_rock] >> -West_rock(X)
    [West_tree] >> -West_tree(X)
    [West_wood] >> -West_wood(X)
    [at_danger] >> at_danger(X)
    [at_ore]    >> -at_ore(X)
    [at_rock]   >> -at_rock(X)
    [at_tree]   >> -at_tree(X)
    [at_wood]   >> -at_wood(X)
    [attacked]  >> -attacked(X)
    [from_West] >> from_West(X)
    [has_ore]   >> has_ore(X)
    [has_wood]  >> -has_wood(X)
Strict rule (11):
    [C(unload,negotiate)]        Unload(X) -> negotiate(X)
    [move(East)]         East(X),East_danger(X) -> enter_danger(X)
    [move(North)]        North(X),North_danger(X) -> enter_danger(X)
    [move(South)]        South(X),South_danger(X) -> enter_danger(X)
    [move(West)]         West(X),West_danger(X) -> enter_danger(X)
    [non-concurrence(East,Fight)]        East(X) -> -Fight(X)
    [non-concurrence(East,Unload)]       East(X) -> -Unload(X)
    [non-concurrence(Fight,East)]        Fight(X) -> -East(X)
    [non-concurrence(Fight,Unload)]      Fight(X) -> -Unload(X)
    [non-concurrence(Unload,East)]       Unload(X) -> -East(X)
    [non-concurrence(Unload,Fight)]      Unload(X) -> -Fight(X)
Defeasible rule (3):
    [F(danger)[O]]       => -at_danger(X)
    [O(negotiate)[O]]    at_danger(X) => negotiate(X)
    [enter(danger[O]]    [O]-at_danger(X) => -enter_danger(X)
No compliant actions. Locating maximally compliant action...
{"response":"RECOMMENDATION","compliant":false,"actions":["Unload"]}
```

Figure 5.11: $Th(s,\mathcal{N})$ and the output solution for the pacifist merchant norm base in a given state.

has the potential to be quite illuminating; we saw that, especially in the environmentally-friendly merchant and benevolent Pac-Man normative systems, constitutive norms bore out the link between the agent's individual actions and the more abstract qualities of environment-friendliness and benevolence. Furthermore, the automatically generated violation reports proved an essential tool for tracking down and analyzing violations that occurred. These factors went a long way towards promoting accountability and transparency in this framework.

Through the extensive battery of tests used, normative resilience was also demonstrated. Clearly, the normative supervisor framework is agnostic with respect to normative system implemented, and expanding and contracting the normative system is no challenge, especially due to the defeasibility of the logic used. On the subject of defeasibility, we saw several different implementations of strong permission, and saw a clear example of direct normative conflict. Indirect normative conflict was somewhat neglected, however.

The behaviour induced by resolving indirect normative conflict was captured by the mechanism for resolving normative deadlock – the Lesser Evil algorithm(s). In the framework of the first reasoner model, courses of actions that involve more violations of individual norms are considered worse; only actions that result in the lowest possible number of violations are passed onto the agent. This method allows us to simply add an additional norm(s) prohibiting a 'worse' behaviour, simulating the result of resolving an indirect normative conflict. Moreover, this allows us to implement contrary-to-duty reasoning; if, in the presence of a triggering violation, the contrary-to-duty obligation is not obeyed, an additional norm will be violated, thus causing the course of action obeying the contrary-to-duty will be prioritized by the Lesser Evil algorithm.

Despite this approach's success on the fronts of transparency and normative resilience, it is far from perfect. It is notable that in the miniature Pac-Man environment, the agent's performance was badly impacted, while it maintained a record of perfect compliance. This was remedied in the regular Pac-Man game; however, in these tests, we saw infrequent but not insignificant violations, which we saw were always a result of normative deadlock. These two problems are linked; because the normative supervisor is completely decoupled from the agent, the agent learns a plan for its (norm-agnostic) behaviour completely independently. The supervised RL agent is not a truly hybrid agent, because though it is a hybrid *framework*, it is not a cohesive agent which is able to incorporate different and conflicting goals into the same plan.

To clarify the issue here, consider this example: a self-driving car has planned a route to a desired location, and the normative supervisor is attached to ensure the car does not break any local regulations. The self-driving car eventually comes to a private road, which it is prohibited from passing through; the normative supervisor forces the car to turn around and reroute. If the applicable norms had been incorporated into the agent's plan from the beginning, it probably could have reached its destination much more efficiently. Alternatively, consider the case where the road leading to the private road is one-way; then, upon coming to the private road, the supervisor must decide whether to proceed or illegally reverse, violating a regulation either way.

116

RL has proven successful for planning tasks, but plans are created through the propagation of delayed rewards through the Q-function during the learning process; at time of operation, all the agent has access to is the learned Q-function. Clearly, the only way to incorporate normative information into a regular RL agent is to involve the normative supervisor in the training process, which we will explore in the next chapter.

A final issue we encounter is that, even though the introduced computational overhead was not prohibitive, it was still substantive, and will be a more prominent factor for larger environments or normative systems. The agent queries the normative supervisor each time it takes an action, and in agents with more time-sensitive functions, this may be problematic. We were able to alleviate the cost of these queries somewhat by choosing a very computationally feasible logic and dynamically translating norms and adding them to the theory as they became relevant (cutting down on the size of the theory and therefore computation time). Nonetheless, the scalability to applications that must process large amounts of data extremely quickly will be limited. This approach certainly prioritizes transparency and oversight over time-efficiency, and will likely remain most effective for applications that do the same.

CHAPTER 6

Learning Compliance

In the ensuing chapter we discuss the results of utilizing the normative supervisor to interfere with an RL agent's training, incorporating normative reasoning into the training phase. First, in a presentation of unpublished work, we attempt to restrict the MDP the agent learns in; there are serious issues with this simple approach, and we will witness some concerning experimental outcomes in the process of exploring an implementation. After, we will introduce norm-guided reinforcement learning (NGRL) [Neu22], a more sophisticated and effective technique, which we show can greatly improve the performance of an agent on which we use online compliance checking. However, NGRL on its own has a number of pertinent limitations (some of which were demonstrated in [Neu22], and some of which we will demonstrate with new experiments), which we will attempt to overcome in the last section (Section 6.3), which contains only unpublished work.

6.1 Restricting Exploration

In the previous chapter, we saw that online compliance checking (OCC), while attractive on several counts – the framework is quite transparent, and can accommodate a significant degree of normative resilience – has some notable failures. The decoupling of the agent and the normative supervisor resulted in a marked disability when it came to planning, both to achieve the agent's primary goal and preemptively avoid violations.

This conundrum is unsurprising. When we run OCC, we are essentially training an agent in an MDP $\mathcal{M}$ and then sending it to operate in another MDP $\mathcal{M}'$ where non-compliant transitions are barred (that is, in all cases where it is possible to do so). The agent has not learned to maximize its expected return in this new MDP, so it is no strange thing that it does not function nearly as well within it.

In the concluding remarks of the previous chapter, we noted that the reasonable next step would be to shift our normative interference to the training phase, specifically by

CHAPTER 6

Learning Compliance

In the ensuing chapter we discuss the results of utilizing the normative supervisor to interfere with an RL agent's training, incorporating normative reasoning into the training phase. First, in a presentation of unpublished work, we attempt to restrict the MDP the agent learns in; there are serious issues with this simple approach, and we will witness some concerning experimental outcomes in the process of exploring an implementation. After, we will introduce norm-guided reinforcement learning (NGRL) [Neu22], a more sophisticated and effective technique, which we show can greatly improve the performance of an agent on which we use online compliance checking. However, NGRL on its own has a number of pertinent limitations (some of which were demonstrated in [Neu22], and some of which we will demonstrate with new experiments), which we will attempt to overcome in the last section (Section 6.3), which contains only unpublished work.

6.1 Restricting Exploration

In the previous chapter, we saw that online compliance checking (OCC), while attractive on several counts – the framework is quite transparent, and can accommodate a significant degree of normative resilience – has some notable failures. The decoupling of the agent and the normative supervisor resulted in a marked disability when it came to planning, both to achieve the agent's primary goal and preemptively avoid violations.

This conundrum is unsurprising. When we run OCC, we are essentially training an agent in an MDP $\mathcal{M}$ and then sending it to operate in another MDP $\mathcal{M}'$ where non-compliant transitions are barred (that is, in all cases where it is possible to do so). The agent has not learned to maximize its expected return in this new MDP, so it is no strange thing that it does not function nearly as well within it.

In the concluding remarks of the previous chapter, we noted that the reasonable next step would be to shift our normative interference to the training phase, specifically by

restricting the transitions available to the agent at training time. In practise, this proved to be a problematic approach; nevertheless, in the unpublished work below, we briefly walk through it and some experiments employing it after offering a basic explanation of the technique. This provides some interesting insights into the potential pitfalls of trying to train norm-compliant agents.

6.1.1   Sheltered MDPs

One way to interfere with an RL agent is to change the way it experiences its environment; that is, we will limit its choices by teaching it an optimal policy within a modified MDP.

Definition 6.1.1. *The sheltered MDP of an MDP $\mathcal{M} = \langle S, A, P, R \rangle$ is the MDP $\mathcal{M}_{sh} = \langle S, A, P_{sh}, R \rangle$, where*

$$P_{sh}(s, a, s') = \begin{cases} P(s, a, s') & a \in A_{\mathcal{N}}(s) \\ 0 & otherwise \end{cases}$$

Within a sheltered MDP, the agent will learn a completely new Q-function. Practically, this will be achieved by running the normative supervisor during the training phase in a manner similar to OCC, removing non-compliant actions and having the agent choose from $A_{\mathcal{N}}(s)$ instead of $A(s)$. Recall that in cases where the set of compliant actions $A_C(s)$ is empty, we have $A_{\mathcal{N}}(s) = A_{NC}(s)$, where $A_{NC}(s)$ is the set of "lesser evil" actions; this will prove to be a severe issue, when we actually implement this technique in an environment where normative deadlock is possible. We will explore the practical results of this in the following subsection.

6.1.2   Experimental Results

We will now walk through the results of training an agent within a sheltered MDP; first, we look at the simpler case with miniature Pac-Man, where some unexpected complications pop up, revealing a couple of potential pitfalls of this approach. Then, we move on to regular Pac-Man, where we see some predictable but disappointing results with respect to the efficacy of using a sheltered MDP. Our battery of experiments will be minimal – we will not explore many normative systems (and we will not utilize the Travelling Merchant at all) because the limited experiments we employ are sufficient to demonstrate that this technique is unsuitable for our purposes.

Miniature Pac-Man

In the below table, we have laid out the results of running the normative supervisor during training. As you can see in the below table, we restrict our attention to the vegan normative system. The table has an additional column labelled **OCC** – a "no" is given when we utilize the normative supervisor only during training; the agent learns within a sheltered MDP $\mathcal{M}_{sh}$ and then is released into an environment that can be described

Norm Base	OCC	% Games Won	Game Score (Avg)	Ghosts Eaten (Avg)
vegan	no	54.8%	223.44	0.478
vegan	yes	80.5%	417.41	0.138

Table 6.1: Results of using a sheltered MDP to train Pac-Man.

with $\mathcal{M}$ without online compliance checking. A "yes" indicates that online compliance checking is used after training, so that the agent is both trained in and operating in $\mathcal{M}_{sh}$.

Clearly, these results are anything but promising. In the first round of games (without online compliance checking), the agent fails both to achieve its goals and refrain from eating the blue ghost (though, the number of ghosts eaten does decrease significantly from what we saw in the baseline tests in Chapter 3, which yielded an average number of ghosts eaten of 0.851). In hindsight, the reason for this failure is obvious; during training, some state-action pairs (s, a) that are not accessible in $\mathcal{M}_{sh}$ (those (s, a) such that $a \notin A_{\mathcal{N}}(s)$) are never visited, and thus the value $Q(s, a)$ remains initialized at zero. When the agent is released into $\mathcal{M}$, it is given the opportunity to visit these state-action pairs; and if in a state s, $Q(s, a') \le 0$ for all $a' \in A(s)$, the agent will select from the actions with the highest Q-values, which will inevitably include any actions $a \notin A_{\mathcal{N}}(s)$ which were effectively excluded while training in $\mathcal{M}_{sh}$, since during training $Q(s, a)$ never changed from 0. The poor game performance can be attributed to the fact that since the agent is now operating in $\mathcal{M}$, the policy it learned is no longer optimal.

Even with this explanation, however, we are left to wonder why OCC suddenly proves ineffective in preventing violations when we train within a sheltered MDP. Suspiciously, no violation reports were generated, and upon re-examining these games, it became clear that the agent had in fact learned to instigate the edge case discussed Section 5.3.3, where Pac-Man enters an area with a power-pellet at the exact same time as a ghost enters, resulting in the ghost becoming scared and Pac-Man eating the ghost all in the same step[1]. This reveals a particularly dangerous facet of trying to teach compliance; if such edge cases exist where the agent can get away with violations undetected, and those violations increase the rewards the agent receives, it will learn to replicate these undetectable violations whenever it can in order to maximize the rewards it learns.

In the interest of checking how an agent learning within a sheltered MDP fared when truly no ghosts can be eaten, we added to the *vegan* norm base the *avoid* rule:

$$avoid : \Rightarrow_O \neg enter_{pacman, danger}$$

with the area designated *danger* corresponding to the cell where the power-pellet is; this essentially yields the cautious vegan normative system we saw in the previous chapter. The results are shown below.

---

[1] Recall, due to a quirk in the game implementation; when Pac-Man eats a power-pellet at the same time that it collides with a ghost, the game changes the ghost into a scared state before it decides whether or not Pac-Man eats the ghost or is eaten by the ghost.

Norm Base	OCC	% Games Won	Game Score (Avg)	Ghosts Eaten (Avg)
vegan − cautious	no	58.3%	226.16	0.302
vegan − cautious	yes	81.2%	400.60	0.0

Table 6.2: Results of using a sheltered MDP with the additional *avoid* rule.

Here, we get much different results. In the games without OCC, we once again get violations occurring, along with sub-optimal performance. This, again, is not surprising, because we are still learning within one MDP and operating in another, and the issue that arises from the possibility of negative Q-values still applies here. However, in the second set of tests, we see excellent results. Recall that for the vegan and benevolent Pac-Man games in the last chapter, Pac-Man ate no ghosts but won less than half the games played, achieving an average score of 25.22 (see page 103 for the precise results). Here, we once again have no violations, but 81.2% of the games played were won, and the average score was 400.60. This performance is on par with the performance of the baseline tests (see page 43); clearly, learning within a sheltered MDP has the potential to fix some of the performance problems noted in Chapter 4, if the proper precautions are taken.

Regular Pac-Man

For the regular Pac-Man game, we once again run the normative supervisor with the *vegan* normative system, and relay the results of tests both with and without OCC. We ran tests with both the safe and hungry policies.

Policy	OCC	% Games Won	Game Score (Avg)	Avg ghosts eaten (Blue / Orange)
safe	no	90.6%	1208.52	0.008 / 0.020
hungry	no	93.2%	1550.72	0.759 / 0.772
safe	yes	91.5%	1219.46	0.007 / 0.019
hungry	yes	90.7%	1210.45	0.016 / 0.026

Table 6.3: Results for tests using sheltered MDPs with the more complex Pac-Man game.

The number of ghosts eaten by the Pac-Man following the safe policy (without OCC) was noticeably decreased from the baseline (a little more than halved), and was not much more than the number of ghosts eaten when OCC alone was used. However, when we run the hungry policy with no OCC, learning within the sheltered MDP proves completely ineffective. This is unsurprising, given that violations are inevitable in this environment (due to the many opportunities for normative deadlock), and more likely for the hungry policy (which compels Pac-Man to seek out ghosts); the agent never actually learned to behave within an MDP in which no violations occur, because no such MDP exists in this case. Summarily, the presence of normative deadlock scenarios almost entirely

cancels out any positive effect learning within a sheltered MDP might have resulted in[2]. The other results, where OCC was used, were not meaningfully different from the results in Chapter 5, suggesting once again that learning in a sheltered MDP in this more normative-deadlock-prone environment has no real positive effect.

6.1.3 Evaluation

As a whole, learning within a sheltered MDP is a largely ineffective method for securing normative resilience – and normative compliance in general. We saw that it even under ideal circumstances, where no violations are allowed to occur during training, training with a sheltered MDP was not itself enough to ensure compliance during operation. We explained how this failure has two intertwined facets; on one hand, we are training the agent in one MDP and releasing it to operate in another. On the other, since the sheltered MDP excludes non-compliant transitions (and therefore violating state-action pairs), the respective Q-values remained initialized at zero; thus in cases where the compliant actions all correspond to negative Q-values, the non-compliant action will be selected.

Even when used in addition to OCC, training with a sheltered MDP had the potential to be just as detrimental as it could be beneficial; we saw how the technique encouraged the agent to exploit loopholes in the normative system, for example. It only has the potential to be effective in improving game performance with OCC if the implementation of the normative system is air-tight (not allowing any unintentional violations).

We furthermore saw that normative deadlock renders the technique effectively useless; in environments where normative deadlock can occur, violation is inevitable, and it is effectively impossible to train a compliant agent while these inevitable points of non-compliance are being rewarded (recall, Pac-Man collects points each time it eats a ghost).

These severe limitations strip any merits of the technique of usefulness, especially considering the cost of running the normative supervisor during training; if we are going to expend so much computational power, we will want more reliable results. Clearly, this point of normative interference was an insufficient measure for improving upon the performance of the normative supervisor; we will explore another form of normative interference in the upcoming section.

6.2 Norm-Guided Reinforcement Learning

Our first attempt at normative interference during the training phase was unsuccessful. Due to theoretical assumptions that clashed with the reality we were attempting to cope with, the results we saw fell short of most of our hopes. We will move on to a new

---

[2]As noted above, there *was* an improvement for the safe policy without OCC. This may be because the agent is already learning to avoid ghosts, and the normative supervisor's prevention of eating ghosts during training did not allow the associated Q-values to increase as much.

strategy, here: we will modify instead of the transitions available to the agent, the agent's reward signal. In particular, we will add a second reward signal.
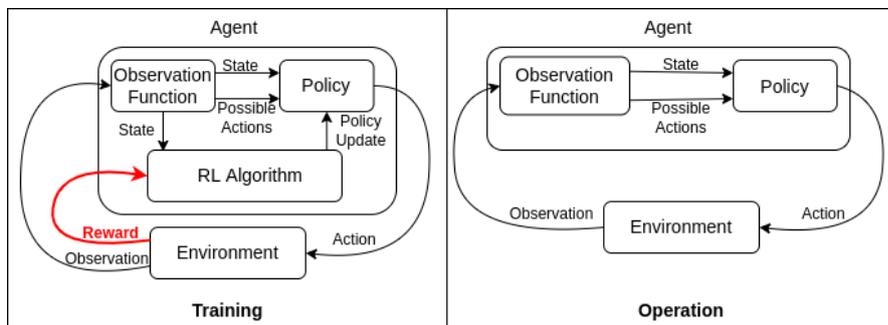

Figure 6.1: Our point of normative interference is highlighted in red.

Recall the objections we considered in Chapter 4 to "ethical utility functions"; while not much can be done about the lack of transparency, we *can* use the normative supervisor framework to remedy some of the other issues that were specified. We identified two essential challenges with reward specification: we must decide (1) *when* to assign rewards or punishments to the agent, and (2) what their *magnitude* should be. In this section, we address these challenges by allotting the first task to the normative supervisor, thereby automating it, and leaving the second to be rendered irrelevant by our methods. This work is taken from [Neu22].

We will be using multi-objective RL in order to balance the normative objective and the agent's primary objective, and we will use the same mechanisms we used to find "lesser evil" actions in the previous chapter (specifically, the mechanism used to evaluate individual actions) to decide whether a given action is compliant or not. Below, we will discuss how the mechanisms of the normative supervisor explained in Section 5.2.2 can be utilized within a MORL framework to learn compliant behaviour, in a process we will call norm-guided reinforcement learning (NGRL). NGRL is a customizable approach to implementing normatively compliant behaviour, that blends techniques and tools from both logic and reinforcement learning to overcome many of the difficulties discussed in Section 4.1.1, when we discussed state-of-the-art approaches to ethical RL and their points of lacking.

The basic approach is this: given an agent with an objective x (and an associated reward function $R_x(s,a)$), we define a second reward function that assigns punishments when an agent violates a normative system $\mathcal{N}$. We call this second reward function a *non-compliance function*:

Definition 6.2.1 (Non-Compliance Function)**.** *A non-compliance function for the normative system $\mathcal{N}$ is a function of the form:*

$$R_{\mathcal{N},p}(s,a) = \begin{cases} p & Th(s,\mathcal{N}) \vdash +\partial_O \neg a \\ 0 & otherwise \end{cases}$$

or

$$R_{\mathcal{N},p}(s,a) = \begin{cases} p & Th(\overline{s}, \mathcal{N}) \vdash +\partial_O lit, -\partial_C lit \\ 0 & otherwise \end{cases}$$

where $p \in \mathbb{R}^-$.

Remark. *Two non-compliance functions are given, one corresponding to each reasoner model in the previous chapter. Note that since we are evaluating a single action in both cases, both functions can be computed in linear time with respect to the size of the theory. In practise, this can be accomplished by inputting $\{a\}$ into the CompliantSolutions algorithm and checking whether or not an empty set is output, or running Algorithm 5.2 for a single action or Algorithm 5.3 for a single action.*

In the above definition, p is called the *penalty*, and $Th(s, \mathcal{N})$ is a normative system translated together with a state s into a defeasible theory, for example by the translators used by the normative supervisor; $Th(\overline{s}, \mathcal{N})$ is the same theory, but with the additional fact asserting that action a from $\overline{s} = (s, a)$ is performed taken into account. The automated derivation of conclusions from $Th(s, \mathcal{N})$ or $Th(\overline{s}, \mathcal{N})$ solves the first challenge of reward specification, allowing us to dynamically determine the compliance of an action in a given state and thereby assign a punishment. As for the second challenge, we will discuss the magnitude of p in Section 6.2.3. Summarily, what this non-compliance function $R_{\mathcal{N},p}$ does is, for each state-action pair (s, a), assign a fixed punishment if and only if a results in a violation of $\mathcal{N}$ in state s.

Now, suppose we have an agent we want to learn objective x with the reward function R_x; it can learn to do so over the MDP

$$\mathcal{M} = \langle S, A, R_x, P \rangle$$

If we have a normative system $\mathcal{N}$ we wish to have the agent adhere to while it fulfills objective x, we can build an MOMDP we will call a *compliance MDP*:

Definition 6.2.2 (Compliance MDP)**.** *Suppose we have a single-objective MDP $\mathcal{M} = \langle S, A, R_x, P \rangle$. Then we can create an associated compliance MDP by introducing a non-compliance function $R_{\mathcal{N},p}(s, a)$ to form the MOMDP:*

$$\mathcal{M}_{\mathcal{N},p} = \langle S, A, (R_x, R_{\mathcal{N},p})^T, P \rangle$$

The goal of NGRL is to find an optimal maximally compliant policy for a compliance MDP, where maximal compliance is a term we substitute for the terminology *ethical policy* from [RSLSRA21][3]. We adapt their definitions below:

---

[3]Note that the presentation of ethical policies in [RSLSRA21] differs slightly from the presentation in [RSSLSRA22], which was conveyed in Section 4.1.

Definition 6.2.3 (Maximally Compliant Policy). *Let Π be the set of all policies over the compliance MDP $\mathcal{M}_{\mathcal{N},p}$. A policy $\pi^* \in \Pi$ is maximally compliant if and only if it is optimal with respect to the value function $V_{\mathcal{N},p}^\pi$ corresponding to $R_{\mathcal{N},p}$. In other words, π^* is maximally compliant for $\mathcal{M}_{\mathcal{N},p}$ iff*

$$V_{\mathcal{N},p}^{\pi^*}(s) = \max_{\pi \in \Pi} V_{\mathcal{N},p}^\pi(s) \tag{6.1}$$

for all s.

Then like in [RSLSRA21], we define within this set of maximally compliant policies those that are optimal with respect to the reward function R_x.

Definition 6.2.4 (Optimal Maximally Compliant Policy). *Let $\Pi_{\mathcal{N}}$ be the set of all maximally compliant policies for $\mathcal{M}_{\mathcal{N},p}$. Then $\pi^* \in \Pi_{\mathcal{N}}$ is optimal maximally compliant for $\mathcal{M}_{\mathcal{N},p}$ iff*

$$V_x^{\pi^*}(s) = \max_{\pi \in \Pi_{\mathcal{N}}} V_x^\pi(s) \tag{6.2}$$

for all s.

There are a plethora of MORL techniques for learning optimal solutions for MOMDPs; there will be multiple ways to learn optimal maximally compliant solutions for some compliance MDP $\mathcal{M}_{\mathcal{N},p}$. Below we will discuss two different MORL algorithms (both of which were introduced in 2.1.2) that we adapt to solve $\mathcal{M}_{\mathcal{N},p}$ for optimal maximally compliant policies. After delving into these methods we will prove some useful properties of NGRL, explore some experimental results, and discuss some notable limitations.

6.2.1   Linear Scalarization

Here, we will use an approach similar to [RSLSRA21]; however, we simplify their case slightly by having only one function associated with normative behaviour, $R_{\mathcal{N},p}$.

Given a weight vector $\vec{w} \in \mathbb{R}_n^+$, the approach we take is to learn $Q_{\mathcal{N},p}(s,a)$ and $Q_x(s,a)$ concurrently, and scalarize the vector of Q-functions via the inner product of the weight and Q-value vector. Recall, we want to maximize

$$V_{scalar}(s) = \vec{w} \cdot \vec{V}(s) \tag{6.3}$$

where $\vec{V}(s) = (V_x(s), V_{\mathcal{N},p}(s))^T$, by selecting actions through the scalarized Q-function be defined as

$$Q_{scalar}(s,a) = \vec{w} \cdot \vec{Q}(s,a) \tag{6.4}$$

where $\vec{Q}(s,a) = (Q_x(s,a), Q_{\mathcal{N},p}(s,a))^T$. In the case of the compliance MDP $\mathcal{M}_{\mathcal{N},p}$ defined above, we can assume without loss of generality that this weight vector is of the form $(1, w)^T$ for some $w \in \mathbb{R}^+$ [RSLSRA21].

How do we find an appropriate w? [RSLSRA21] describes a method using Convex Hull Value Iteration [BN08]. For the sake of brevity, we will not discuss this algorithm in detail, and only relay the results from [RSLSRA21]:

Proposition 6.2.1. *If at least one maximally compliant policy exists, there exists a value* $w \in \mathbb{R}^+$ *such that any policy maximizing* $V_{scalar}(s)$ *is optimal maximally compliant.*

This proposition follows from Theorem 1 of [RSLSRA21].

6.2.2 TLQ-Learning

We turn next to TLQ-Learning. In this case, implementing this framework is quite simple, as we have only two objectives: the objective represented by R_x (which we want to maximize) and the objective represented by $R_{\mathcal{N},p}$, which we would like to prioritize and constrain. Thus, we need to set a constant threshold $C_{\mathcal{N},p}$ for the objective represented by $R_{\mathcal{N},p}$, while the threshold for the objective represented by R_x is set to $C_x = +\infty$.

Recall from Section 2.1.2, when we select an action, instead of Q-values, we consider CQ-values defined as

$$CQ_i(s, a) = \min(Q_i(s, a), C_i)$$

which take the thresholds into account. The next step in this approach is to order actions based on their CQ-values. Again, since we only have two objectives, we were able to simplify the procedure given in [VDB$^+$11] to:

1. Create a set $norm(s) = \{a \in A(s) \mid CQ_{\mathcal{N},p}(s, a) = \max_{a' \in A(s)} CQ_{\mathcal{N},p}(s, a')\}$ of actions with the highest $CQ_{\mathcal{N},p}$-values for the current state s.

2. Return a subset $opt(s) = \{a \in norm(s) \mid CQ_x(s, a) = \max_{a' \in norm(s)} CQ_x(s, a')\}$ of the actions of $norm(s)$ with the highest CQ_x-values.

Our policy must select, for a state s, an action from $opt(s)$.

Summarily, we always take an action that is maximal for $CQ_{\mathcal{N},p}$, and within the actions that maximize $CQ_{\mathcal{N},p}$, we take the action with a value that is maximal for Q_x.

Proposition 6.2.2. *If an maximally compliant policy exists, when* $C_{\mathcal{N},p} = 0$, *any policy such that* $\pi(s) \in opt(s)$ *for all* s *designates an optimal maximally-compliant policy for* $\mathcal{M}_{\mathcal{N},p}$.

Proof sketch: Since the value of $R_{\mathcal{N},p}(s, a)$ is either strictly negative (when a violation takes place) or zero (when the action is compliant), $Q_{\mathcal{N},p}(s, a) \leq 0$ for all state-action pairs. Therefore, if we set the threshold $C_{\mathcal{N},p} = 0$, we get $CQ_{\mathcal{N},p}(s, a) = Q_{\mathcal{N},p}(s, a)$. This means that any policy π that only follows actions from $norm(s)$ fulfills Definition 6.2.3. Likewise, any policy π^* that only follows actions from $opt(s)$ will fulfill Definition 6.2.4 for an optimal maximally compliant policy. □

6.2.3   Magnitude of Penalties

In the above subsections, we have referenced $R_{\mathcal{N},p}(s,a)$ without specifying a penalty p. As promised, we now address the magnitude of p. In order to address the issue of what p should actually be, in practise, we will need the following lemma:

Lemma 6.2.1. *Let $\mathcal{M} = \langle S, A, R, P \rangle$ be an MDP, with two compliance MDPs $\mathcal{M}_{\mathcal{N},p} = \langle S, A, (R, R_{\mathcal{N},p})^T, P \rangle$ and $\mathcal{M}_{\mathcal{N},q} = \langle S, A, (R, R_{\mathcal{N},q})^T, P \rangle$. Then for any policy $\pi \in \Pi_{\mathcal{M}}$:*

$$V^\pi_{\mathcal{N},p}(s) = c \cdot V^\pi_{\mathcal{N},q}(s)$$

for some positive coefficient c, where $V^\pi_{\mathcal{N},p}(s)$ and $V^\pi_{\mathcal{N},q}(s)$ are the value functions associated with $R_{\mathcal{N},p}$ and $R_{\mathcal{N},q}$ respectively.

Proof. Let $\pi \in \Pi_{\mathcal{M}}$. Then by the linearity of conditional expectation:

$$V^\pi_{\mathcal{N},p}(s) = \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t R_{\mathcal{N},p}(s_{i+t}, \pi(s_{i+t}))|s_i = s\right]$$

$$= p \cdot \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t R_{\mathcal{N},-1}(s_{i+t}, \pi(s_{i+t}))|s_i = s\right]$$

$$= \frac{p}{q} \cdot \mathbb{E}\left[\sum_{t=0}^\infty \gamma^t R_{\mathcal{N},q}(s_{i+t}, \pi(s_{i+t}))|s_i = s\right]$$

$\square$

With this lemma it is a simple matter to prove the following useful feature of NGRL:

Theorem 6.2.2. *If a policy π is maximally compliant for the compliance MDP $\mathcal{M}_{\mathcal{N},p}$ for some constant $p \in \mathbb{R}^-$, it is maximally compliant for the compliance MDP $\mathcal{M}_{\mathcal{N},q}$ for any $q \in \mathbb{R}^-$.*

Proof. Let π^* be a maximally compliant policy for $\mathcal{M}_{\mathcal{N}} = \langle S, A, (R_x, R_{\mathcal{N},p})^T, P \rangle$. Then we know that

$$V^{\pi^*}_{\mathcal{N},p}(s) = \max_{\pi \in \Pi_{\mathcal{M}}} V^\pi_{\mathcal{N},p}(s)$$

However, for any arbitrary penalty $q \in \mathbb{R}^-$, there is a $c \in \mathbb{R}^+$ such that $V^\pi_{\mathcal{N},p}(s) = c \cdot V^\pi_{\mathcal{N},q}(s)$ for any $\pi \in \Pi_{\mathcal{M}}$. Therefore

$$V^{\pi^*}_{\mathcal{N},p}(s) = \max_{\pi \in \Pi_{\mathcal{M}}} V^\pi_{\mathcal{N},p}(s)$$

,

$$c \cdot V^{\pi^*}_{\mathcal{N},q}(s) = \max_{\pi \in \Pi_{\mathcal{M}}} c \cdot V^\pi_{\mathcal{N},q}(s)$$

$$V^{\pi^*}_{\mathcal{N},q}(s) = \max_{\pi \in \Pi_{\mathcal{M}}} V^\pi_{\mathcal{N},q}(s)$$

.

So if π^* is a maximally compliant policy for the compliance MDP $\mathcal{M}_{\mathcal{N},p} = \langle S, A, (R_x, R_{\mathcal{N},p})^T, P \rangle$ it is maximally compliant for the compliance MDP $\mathcal{M}_{\mathcal{N},q} = \langle S, A, (R_x, R_{\mathcal{N},q})^T, P \rangle$ as well. $\qquad\square$

Thus, the value of p is irrelevant, so for the remainder of this discussion we will only deal with the non-compliance function $R_{\mathcal{N}} := R_{\mathcal{N},-1}$. This solves the second essential challenge we identified for reward specification.

6.2.4 Experimental Results

In the following sections, we will explore NGRL through our case studies. In the section immediately below, we will reproduce the miniature Pac-Man experiments with regular NGRL (NGRL without any additional features like function approximation), and then in a subsection we will look at NGRL with function approximation. In the case of NGRL with function approximation, we will use the vegan normative system to get a performance baseline for the technique, but then we will proceed to the vegetarian normative system which will show us a pertinent weakness of NGRL with function approximation. We will not recycle the whole battery of experiments explored in Section 5.3.3, as we will have already shown that though the technique in principle can take advantage of the defeasibility of DDL (e.g., it can handle strong permission), there are problems that arise from attempting to implement more complex normative systems with the function approximation variant of NGRL. In the next section (Section 6.2.5), where we discuss the potential shortcomings of NGRL, we will look at the Travelling Merchant case study, which allows us to establish further limitations around the technique.

In this preliminary section utilizing regular NGRL, we resurrect the "benevolent Pac-Man" from Section 5.3.3 (we will not bother with Vegan Pac-Man since we know that the normative supervisor yields the exact same results when these two normative systems are used); we will stick with this normative system $\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, \mu \rangle$ throughout these experiments, only adding a permissive norm to contrast results a bit later.

We ran the tests summarized in Table 6.4 on the two NGRL agents we described above (the linear scalarization agent and the TLQ-learning agent). Again, the **OCC** column tells us if online compliance checking was used at operation time.

Agent	OCC	% Games Won	Avg Game Score	Avg Ghosts Eaten
Scalarized	no	82.0%	433.74	0.142
TLQL	no	82.0%	433.74	0.142
Scalarized	yes	86.3%	448.23	0.001
TLQL	yes	86.3%	448.23	0.001

Table 6.4: Results with the two different MORL agents, with and without OCC.

The results of Table 6.4 show that for NGRL (without OCC), the agents failed to comply with the normative system about as often as they failed to win the game (the agent ate

the ghost in 14.2 % of the games and and lost 18% of the games). It is notable that both NGRL methods, linear scalarization and TLQL, converged to the same policy, resulting in identical numbers. However, it is also worth noting that though the number of ghosts eaten was reduced significantly (in the linear scalarization and TLQL tests, the number of ghosts eaten is only one sixth of what it was for the Q-learning agent whose baseline statistics we saw in Chapter 3 on page 43), the behaviour of eating ghosts was not even close to being eliminated.

We received the best overall results when combining NGRL with OCC; it was in these tests that we saw the highest win rates and scores, and an elimination of ghosts consumed.[4] While the normative supervisor, in its original role as an online compliance checker, eliminates the possibility of unnecessarily violating the norm base, it does not teach the agent to learn an optimal policy *within* the bounds of compliant behaviour. We saw in the previous chapter how it hampered the agent's performance. NGRL allows the agent to incorporate information about which actions are compliant and which are not into its learning of optimal behaviour; as a result, the performance of the agent in accomplishing its norm-agnostic goals is not compromised to the same degree it might be when we employ only online compliance-checking. In fact, given the baseline results, where only 68.5% of games were won and the average score was 441.71, we can fairly say that the agent's performance is not hampered at all; quite the opposite.

As a final consideration, suppose we introduce a new regulative norm, a permissive norm:

$$\mathbf{P}(eat_{blueGhost})|\top) \in \mathcal{R} \tag{6.5}$$

This gives Pac-Man explicit permission to eat the blue ghost, overriding any prohibitions implied by the obligation for benevolence and the consttutive norms linking benevolence to eating ghosts. As we can see from the results in Table 6.5, Pac-Man will return to eating ghosts after being given explicit permission, exhibiting identical behaviour to the baseline Q-learning agent from Chapter 3.

Agent	OCC	% Games Won	Game Score (Avg)	Ghosts Eaten (Avg)
Scalarized	no	68.5%	441.71	0.851
TLQL	no	68.5%	441.71	0.851

Table 6.5: Results with Pac-Man permitted to eat the ghost.

NGRL with Function Approximation

In this subsection, we will go over the results of approximating Q_x and $Q_{\mathcal{N}}$ (separately) with linear functions, after which they are once again scalarized.

---

[4]An elimination with the exception of 1. We confirmed this to be the edge case described in the previous chapter and the section on sheltered MDPs above, where the agent consumes a ghost 'by accident', when it collides with the ghost right as it eats the power pellet.

Our first tests were within the same normative system employed above, prior to the addition of the permissive norm 6.5. The only difference is that for the normative system in an environment with two ghosts we must add a constitutive norm for the orange ghost:

$$\mathbf{C}(eat_{orangeGhost}, eat_{person}|\top) \in \mathcal{C} \tag{6.6}$$

The results in Table 6.6 were obtained from batches of 1000 tests, performed after Pac-Man had been trained over 300 games[5]. Here, recall, training entails adjusting the weight vector $\vec{\theta}$ to create a better approximation of the Q-function (we specifically use the feature extractor associated with the "hungry" policy). Note that the below results do not include function approximation with TLQ-learning; this technique strictly prioritizes the normative objective, and so long as we maintain the threshold of 0 for $Q_{\mathcal{N}}$, Pac-Man stayed perfectly still in its initial position, where there was no chance of it violating its normative system.

Agent	OCC	% Games Won	Game Score (Avg)	Avg ghosts eaten (Blue / Orange)
Scalarized	no	91.9%	1228.40	0.019 / 0.015
Scalarized	yes	90.4%	1204.47	0.0 / 0.021

Table 6.6: Results for tests on NGRL with function approximation, using the hungry policy.

From these results we can see that the NGRL approach is more effective than the Q-learning agent with OCC (that is, with the hungry policy – see page 105); this is presumably because of a reduction of the scenarios where Pac-Man becomes trapped between two scared ghosts, and is forced to eat one of them, which were proven to be one of the only two possible violation cases. Using NGRL, Pac-Man will learn to avoid some of these situations, while the Q-learning agent controlled with OCC alone must be reactive in its behaviour. However, we can also see that once again the combined use of NGRL and supervision is the most effective, suggesting that these two approaches should be used complementarily.

However, problems arise when we adopt the permissive norm 6.5 introduced above. The agent does not take the permission into account, and proceeds to learn as though we had not added this norm – this is because of the feature extractor used for the Q-function approximation, which was the same as the one used in previous chapters (for the "hungry" policy). This feature extractor uses as features: (1) the distance to the closest food pellet to Pac-Man after taking action a in state s, (2) whether taking action a in state s results in Pac-Man eating a food pellet, (3) the number of ghosts one step away, and (4) the number of scared ghosts one step away.

The original extractor (labelled "hungry" in Table 6.7) does not differentiate between blue and orange ghosts, and instead uses as a feature the number of ghosts that are

---

[5]In the case of NGRL with function approximation, an additional 50 episodes of training gave slightly better results.

Agent	OCC?	Feature Extractor	% Games Won	Game Score (Avg)	Avg ghosts eaten (Blue / Orange)
linear scalarization	no	hungry	91.7%	1226.41	0.018 / 0.017
linear scalarization	no	blue	94.4%	1423.87	0.813 / 0.020

Table 6.7: Results with the permission norm.

within one step of Pac-Man. As a result, Pac-Man cannot learn to stop eating only the blue ghosts – there is no way to distinguish states where a blue ghost is near Pac-Man from ones where an orange ghost is. However, we can split this into two features, one giving 1 if a blue ghost is nearby and 0 otherwise, and the other doing the same for the orange ghost. We called this the 'blue' extractor, and it led to the final results in Table 6.7.

Though Pac-Man certainly did display the behaviour of avoiding the consumption of orange ghosts, it still did eat 20 orange ghosts over 1000 games. Clearly, though NGRL can better avoid violations of some normative systems due to normative deadlock, than can a Q-learning agent under OCC, it at the same time is prone to unnecessary violations when used with function approximation – suggesting, again, that these two methods for constraining behaviour are best used together. That is, the agent can learn compliant behaviour, while still having the supervisor correct for any avoidable violations that are ignored by the agent's policy, and recording any violations that do occur for the sake of transparency. This is where we get our best results.

However, it may be possible to improve the agent's ability to avoid situations resulting in unavoidable violations as occurred with respect to $\mathcal{N}$, provided that the correct features are selected for the linear approximation of $Q_{\mathcal{N}}$. Unfortunately, doing so requires manual work involving comparing domain knowledge of the agent's environment and the normative systems, which could be an unwieldy or downright unfeasible task in more complex cases. Summarily, scaling NGRL through linear function approximation would require significant further investigation into the problem of feature engineering.

6.2.5   Limitations of NGRL

Aside from the above-demonstrated successes and failures, NGRL may fail to capture some desired facets of normative resilience in the form of normative conflicts and contrary-to-duty obligations; this is discussed below.

DDL facilitates non-monotonic reasoning, which makes it suitable for handling normative systems that may at first glance seem to be inconsistent. Take, for instance, a normative system $\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, \mu \rangle$, where $\mathcal{R} = \{\mathbf{O}(p|q), \mathbf{F}(p|q)\}$, with the obligation taking precedence over the prohibition. In DDL, we would represent the rules as $r_1 : q \Rightarrow_O p$ and $r_2 : q \Rightarrow_O \neg p$, where $r_1 > r_2$. If we assume as a fact q, we will be able to derive the conclusion: $+\partial_O p$. This dictates the behaviour we would desire from the system – the agent sees to it that p, or else is punished. However, r_2 (that is, $\mathbf{F}(p|q)$) is still violated by

this course of action, and we might want to assign the agent a (albeit lesser) punishment for this violation, incentivising the agent to avoid situations where these conflicting norms are triggered at all. In our framework, however, obeying the obligations of p is considered, for all intents and purposes, compliant.

In some cases, where the conflict occurs for reasons completely out of the agent's control, our framework is the more appropriate approach. To illustrate, consider that (1) you are forbidden from surpassing the speed limit, and (2) if your child is dying and you are driving them to the hospital, you ought to speed – then there is no reason to disincentivize you from speeding to save your dying child, even though you have violated norm (1). However, in cases where the agent's own actions have resulted in a scenario where it cannot obey all applicable norms, we might want to assign punishments to any action the agent takes, albeit of different magnitudes.

A second, more problematic scenario can be found in contrary-to-duty obligations. Below, consider a normative system $\mathcal{N} = \langle \mathcal{C}, \mathcal{R}, \mu \rangle$, where $\mathcal{R} = \{\mathbf{O}(b|\neg a), \mathbf{O}(a|\top)\}$. In DDL these norms can be translated as: $r_1 : \Rightarrow_O a$ and $r_2 : \neg a \Rightarrow_O b$. Now, suppose that the agent has two actions available to it in a given state s: b and c. Now, if the agent could take action a, it would not receive a penalty from the non-compliance function for violating r_1; however, that is not possible. If the agent takes action b, it will be punished, because r_1 along with the non-concurrence rules in $Th(s, \mathcal{N})$ will (for the first reasoner model) allow us to derive $+\partial_O \neg b$, signaling a violation of the above normative system which will trigger a penalty. If the agent takes action c, the result will be identical. However, choosing b obeys the CTD obligation in the above normative system, and so ideally, we would incentivize the agent to choose this action over c; if we were using OCC, the LesserEvil algorithm would take care of this case, but the non-compliance function does not allow for this nuance. The same problem, of course, occurs for the second reasoner model as well.

We can demonstrate these issues with the merchant environment. If we consider the case of the environmentally-friendly merchant, NGRL yields the correct behaviour (see Figure 5.4). When we use NGRL with the pacifist normative system, however, we see the behavioiur in Figure 6.2.

We can see that the agent takes a path identical to the path taken with online compliance checking (see Figure 5.5), except for the point at which it enters the dangerous area. The NGRL agent fights instead of negotiating; as we predicted above, the non-compliant action *fight* is punished just as much as the weakly compliant action *unload* (both actions in that state have identical $Q_{\mathcal{N}}$-values), and as a result, the contrary-to-duty obligation is not observed (since $fight$ is the more advantageous action for the agent).

Based on examples like those above, we might find it appropriate to assign graded penalties to violations of a normative system, depending on how much of it is violated; however, this re-introduces the problem of scaling the magnitude of rewards, and we are once again left to attempt to justify the weights we apply.

An additional shortcoming of NGRL is that it lacks transparency, when compared to
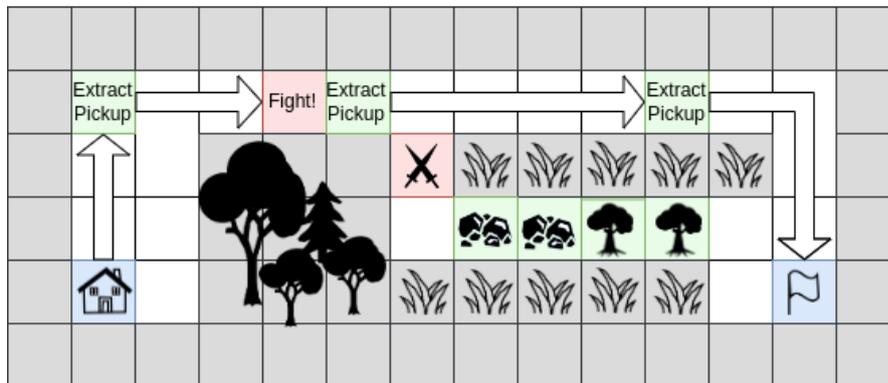
Figure 6.2: The pacifist merchant's path after NGRL.

OCC. Though we can in theory check the compliance of a given state-action pair with the normative supervisor (and this is indeed done during training), while the agent is actually operating in the environment, it does so by selecting actions based on its Q-function, which retains none of the information from the original normative system (besides how likely we are to incur violations from a given point). If we do encounter an issue with an (non-compliant) action a committed in a state s, one approach we can take is to evaluate $Th(\overline{s}, \mathcal{N})$ for $\overline{s} = (s, a)$ with the normative supervisor. If no violations occur, we can assume that the issue is with the normative system; either the system itself or the automated implementation is too permissive. Then, we must fix our implementation and retrain the agent from scratch. However, if we encounter violations, we are left to wonder blindly why the agent learned to select a non-compliant action.

This brings us to a final limitation: as stated above, if we wish to change the normative system governing the agent, we must retrain it from scratch. This will be a computationally expensive and potentially time-consuming task; NGRL does not facilitate seamless switching between normative systems in the way OCC does.

To remedy the above issues, we can, of course, do what we did in the last section – perform normative interference while training, and use OCC during operation. This solves the issues with norm violation and transparency, and can to some degree facilitate norm change; OCC can keep the agent from unnecessary violations while the agent is retraining with its new normative objective.

6.2.6   Evaluation

One of the most prominent avenues to teaching normative compliance to autonomous agents through reinforcement learning entails assigning penalties to behaviour that violates whatever normative system the agent is subject to. To do so, we must determine (1) when to assign a penalty, and (2) how steep that penalty should be. We have attempted to address both of these challenges with norm-guided reinforcement learning (NGRL), a framework for learning that utilizes the normative supervisor to evaluate the

agent's actions with respect to a normative system. In doing so, we relegate (1) to the assessments of the normative supervisor, and sidestep (2) with techniques that produce compliant policies, showing that they will do so regardless of the magnitude of punishment given. This approach provides us with a framework for identifying non-compliant actions with respect to a given normative system, and assigning punishments to a MORL agent simultaneously learning to achieve normative and norm-agnostic objectives. NGRL offers more versatility, with respect to the complexity of the normative systems that can be adhered to, than directly assigning rewards to specific events or with respect to simple constraints as is the case with most ethical RL techniques. It may be the case that there is no obvious or coherent way to summarize an entire normative system by selecting specific events and assigning punishments to them. By using NGRL, we expand what kinds of normatively compliant behaviour we can learn, and are allowed to specify them in a more natural way.

That being said, the technique falls short of several of the standards we originally put forth. Since evaluations through the normative supervisor account for strong permissions and conflicting norms (although, not indirect conflicts – recall that we resolved these with the Lesser Evil algorithms, which are only called when all actions are non-compliant; in NGRL a punishment will be delivered no matter what choice is made), some facets of normative resilience are accounted for. Normative conflict, with the subtle caveat addressed in Section 6.2.5 and the marked exception of indirect conflict, was not an issue. Norm violation, however, was. As we discussed in Section 6.2.5, CTD obligations, much like indirect normative conflict, are a problem; both the more and less compliant choices are punished equally, and the agent has no incentive to, for example, obey the contrary-to-duty obligation. One obvious way to fix this would be to assign graded punishments based on, for example, how many violations are incurred by a given action. But this would leave the issue of penalty magnitude open again. Finally, norm change is also a problem; while the normative supervisor itself can easily accommodate changing normative systems, the agent will have to be retrained in order for its behaviour to reflect these changes.

There are also issues on the front of transparency. As was discussed at the end of Section 6.2.5, there are ways we can leverage the transparency of the normative supervisor as a framework, but ultimately, the norms the agent is subject to are still being compressed into a value function. This may be something of an inevitability if an agent uses RL to learn compliance.

Of course, transparency ceases to be an issue when we utilize both OCC and NGRL – and that is where our best results were achieved, according to the experiments in Section 6.2.4. However, to do so is to require much computational expenditure both during and after training, and this might not be feasible in certain cases. So, while we can remedy the shortcomings of OCC with NGRL, and vice versa, the issue of the price of computation remains.

135

6.3 NGRL with Violation Counting

In a manner similar to compliance specifications, NGRL fails when dealing with the sub-ideal world. No mechanism is given to differentiate between "more compliant" and "less compliant" courses of action, and as a result, any scenario that results in a violation is treated equal to all others. This is a debilitating shortcoming when it comes to dealing with contrary-to-duty obligations and resolving normative deadlock; due to the manner in which we dealt with indirect normative conflict with the normative supervisor, this is also not possible. However, there is a quick fix to this issue, which will work at least in the case of NGRL without function approximation, when using the non-compliance function associated with the second reasoner model.

Let us first revisit the non-compliance function associated with the second reasoner model. In order to simplify how we express it, we will summarize the violation condition with a binary compliance function:

$$compl(\overline{s}) = \begin{cases} 1 & |viol(\overline{s}, \mathcal{N})| = 0 \\ 0 & otherwise \end{cases} \tag{6.7}$$

Then the non-compliance function can be rewritten as:

$$R_{\mathcal{N},p}(s,a) = p \cdot (1 - compl(s,a)) \tag{6.8}$$

where $p \in \mathbb{R}^-$ is the penalty; however, since the magnitude of the penalty does not matter (see Theorem 6.2.2), we assume $p = -1$ from here. This simplified definition will come in handy when we prove some additional results about NGRL momentarily.

We next introduce a *violation counting functions* to supplement non-compliance functions as introduced in the last section:

Definition 6.3.1 (Violation Counting Function)**.** *A violation counting function for a normative system $\mathcal{N}$ is a function of the form:*

$$VC_{\mathcal{N}}(s,a) = |viol(\overline{s}, \mathcal{N})|$$

For a state-action pair $\overline{s} = (s,a)$, we call $VC_{\mathcal{N}}(s,a)$ the pair's VC-value

In other words, a violation counting function is a function over state-action pairs to the natural numbers. This function outputs the number of violations associated with state-action pair $\overline{s} = (s,a)$ with respect to normative system $\mathcal{N}$.

If, during NGRL, we are computing some $R_{\mathcal{N}}(s,a)$, we can essentially get $VC_{\mathcal{N}}(s,a)$ for free; in the process of checking the compliance of some state-action pair $\overline{s}$ with the second reasoner model, we must call the theorem prover once. That is, we can easily extract $|viol(\overline{s}, \mathcal{N})|$ from the set *conclusions* output by Algorithm 5.3.

This approach allows us to capture the results of online compliance checking without resorting to running the normative supervisor at operation time; as we check the compliance of a given action a in a given state s during training to update $Q_{\mathcal{N}}(s,a)$, we can update $VC_{\mathcal{N}}(s,a)$ along the way. Then in each state s, the set of actions with minimal VC-values is precisely $A_{\mathcal{N}}(s)$:

Theorem 6.3.1. *For a normative system $\mathcal{N}$, the following equivalence holds:*

$$A_{\mathcal{N}}(s) = argmin_{a \in A(s)} VC_{\mathcal{N}}(s,a) \tag{6.9}$$

Proof. If the set $A_C(s)$ extracted from Algorithm 5.3 is non-empty, then the $a \in A_C(s)$ are those $a \in A(s)$ such that for $\overline{s} = (s,a)$, $|viol(\overline{s}, \mathcal{N})| = 0$. Then $argmin_{a \in A(s)} VC_{\mathcal{N}}(s,a)$ are precisely those actions such that $|viol(\overline{s}, \mathcal{N})| = 0$, and $A_C(s) = argmin_{a \in A(s)} VC_{\mathcal{N}}(s,a)$.

If $A_C(s)$ is empty, we must run Algorithm 5.4. Lines 7-11 compute the value *score* which is clearly $|viol(\overline{s}, \mathcal{N})|$ by Definition 2.2.5, and therefore the output of Algorithm 5.4, $A_{NC}(s)$, is $argmin_{a \in A(s)} VC_{\mathcal{N}}(s,a)$.

So according to Definition 5.1.1, $A_{\mathcal{N}}(s) = argmin_{a \in A(s)} VC_{\mathcal{N}}(s,a)$. $\qquad\square$

Using this violation counting function, we can revisit the TLQ-learning approach to NGRL we proposed above, and in addition to the functions CQ_x and $CQ_{\mathcal{N}}$, we add the function $VC_{\mathcal{N}}$ to our procedure:

1. Create a set $norm(s) = \{a \in A(s) \mid CQ_{\mathcal{N}}(s,a) = \max_{a' \in A(s)} CQ_{\mathcal{N}}(s,a')\}$ of actions with the highest $CQ_{\mathcal{N}}$-values for the current state s.

2. Create a set $viol(s) = \{a \in norm(s) \mid VC_{\mathcal{N}}(s,a) = \min_{a' \in norm(s)} VC_{\mathcal{N}}(s,a')\}$ of actions from $norm(s)$ with the lowest $VC_{\mathcal{N}}$-values for the current state s.

3. Return a subset $opt(s) = \{a \in viol(s) \mid CQ_x(s,a) = \max_{a' \in viol(s)} CQ_x(s,a')\}$ of the actions of $viol(s)$, with the highest CQ_x-values.

Then our policy must select, for a state s, an action from $opt(s)$.

Notice that we give priority to selecting actions maximizing $CQ_{\mathcal{N}}$-values, not for selecting actions which minimize $VC_{\mathcal{N}}$-values, even though Theorem 6.3.1 confirms that the actions which minimize $VC_{\mathcal{N}}$ are precisely those normatively optimal actions as defined in Definition 5.1.1. As a result, the actions produced by the above procedure might not be normatively optimal with respect to all actions in $A(s)$. In order to explain this design decision, we need to prove a couple of things.

Firstly, we introduce a new function over policies, an Expected Non-Compliance Count (ENCC):

$$ENCC(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t (1 - compl(\overline{s}_t)) | s_0 \in S\right] \tag{6.10}$$

$\bar{s}_0, \bar{s}_1, \ldots$ where $\bar{s}_t = (s_t, \pi(s_t))$ is a trace of state-action pairs generated by following policy π, and $\gamma \in [0, 1)$ is a discount factor; this function gives an expected count of violations occurring over the course of following π, prioritizing violations occurring in the immediate.

As it turns out, maximally compliant policies minimize the ENCC:

Lemma 6.3.2. *Let Π be the set of all policies over the compliance MDP $\mathcal{M}_{\mathcal{N}}$. Then if π^* is maximally-compliant:*

$$ENCC(\pi^*) = \min_{\pi \in \Pi} ENCC(\pi)$$

Proof. Suppose π^* is maximally compliant, then we know $V_{\mathcal{N}}^{\pi^*}(s) = \max_{\pi \in \Pi} V_{\mathcal{N}}^{\pi}(s)$. At each step $\bar{s}_t = (s_t, \pi(s_t))$ in a trace generated by a policy π, the reward awarded to the agent is equal to $-(1 - compl(\bar{s}_t))$; then according to Equation 2.1 we have the equivalence:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} -\gamma^t(1 - compl(s_{t+i}, \pi^*(s_{t+i})))|s_i = s\right]$$
$$= \max_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^{\infty} -\gamma^t(1 - compl(s_{t+i}, \pi(s_{t+i})))|s_i = s\right]$$

Then by the linearity of conditional expectation:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t(1 - compl(s_{t+i}, \pi^*(s_{t+i})))|s_i = s\right]$$
$$= -\max_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^{\infty} -\gamma^t(1 - compl(s_{t+i}, \pi(s_{t+i})))|s_i = s\right]$$
$$= \min_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t(1 - compl(s_{t+i}, \pi(s_{t+i})))|s_i = s\right]$$

So we get the equivalence:

$$ENCC(\pi^*) = \min_{\pi \in \Pi} ENCC(\pi)$$

□

Now recall that according to Proposition 6.2.2, the policy produced through the TLQ-learning algorithm is maximally compliant. Then we can see that:

Corollary 6.3.1. *The policy produced with NGRL with violation counting minimizes expected ENCC.*

Proof. Proposition 6.2.2 shows that the policy produced with TLQ-learning is maximally compliant, and Lemma 6.3.2 tells us that it has minimal expected ENCC. □

Corollary 6.3.1 shows why *Policy* prioritizes maximizing $Q_\mathcal{N}$ over choosing only actions from $A_\mathcal{N}(s)$. Though ideally we will always want to choose actions from $A_\mathcal{N}(s)$ (which is $argmin_{a \in A(s)} VC_\mathcal{N}(s,a)$ according to Theorem 6.3.1), policies that choose actions from $argmax_{a \in A(s)} Q_\mathcal{N}(s,a)$ minimize the expected count of violations (ENCC) that will occur over the whole trace generated by the policy. This has the effect of prioritizing the overall normative objective over immediate maximal compliance and can minimize violations in the long term. In other words, if we take only actions from $A_\mathcal{N}(s)$ we may end up choosing an action that is compliant in the immediate, but results in more violations later on. By using a policy that prioritizes minimizing the ENCC, we can choose actions which minimize violations in the long term, and by further minimizing $VC_\mathcal{N}$, we can then exclude the actions which result in the most violations in normative deadlock scenarios.

6.3.1 Experiments

We will consider now the pacifist merchant, to follow up on our demonstration in Section 6.2.5; however, we modify the case study slightly by using another map (in which we trained the agent with 3000 episodes[6]). We demonstrate the optimal behaviour in Figure 6.3.

Figure 6.3: Optimal behaviour for the merchant. (E) indicates that the agent extracted resources, (P) indicates the agent picked up resources, and (F) indicated that the agent fought.

Now, once we adopt the pacifist normative system, the agent should take the action *unload* when it is forced to pass through a dangerous area; when it is given the choice between entering a dangerous area and not, it should choose not to enter the dangerous area. This behaviour is depicted in Figure 6.4.

However, in Figure 6.5, we see the result of of running NGRL (without violation counting) in this new environment with the pacifist normative system imposed.

---

[6]This environment is somewhat easier to navigate than the original merchant environment as it is more spread out.

Figure 6.4: The correct behaviour for the pacifist merchant. (U) indicates that the agent unloads its inventory.


Figure 6.5: The pacifist merchant trained using NGRL.

As we can see, the agent takes the same route, but when it is forced to enter the dangerous area, it chooses *fight* instead of *unload*. This is because of the issue we have already discussed; since any course of action will incur some violation, all options are punished and therefore treated equally, even though fighting violates both the primary and contrary-to-duty obligations, and unloading only violates the primary obligation.

When we run NGRL with violation counting, however, we get exactly the behaviour in Figure 6.4. Though in the state s where at_danger is true, $Q_\mathcal{N}(s, fight) = Q_\mathcal{N}(s, unload)$, it is going to be the case that $VC_\mathcal{N}(s, fight) > VC_\mathcal{N}(s, unload)$, so the action $unload$ is taken.

Now, we will consider one more case, where we have changed the environment again; instead of having only two dangerous areas, we have four; in this environment, when the agent is given the choice to enter the second dangerous area, it can either do so (and that is the last violation that will happen), or it can choose not to, and end up forced to pass through two additional dangerous areas. In this environment, choosing to violate the normative system unnecessarily results in fewer violations later on, so the agent should choose that path.

We can see in Figure 6.6 that the agent we train with NGRL with violation counting does indeed follow this path, while observing the CTD obligation whenever a violation is incurred.

Figure 6.6: The pacifist agent voluntarily violating a norm in order to avoid future violations.

6.3.2 Evaluation

When we attempted to train the pacifist merchant with NGRL, we saw undesirable results (see Figures 6.2 and 6.5); when the agent was forced to enter the dangerous area, both unloading (which was mandated by the CTD obligation) and fighting had the same (negative) Q-value, according to $Q_\mathcal{N}$. Then, since fighting is the more advantageous choice according to Q_x, the agent chose to fight instead of negotiate.

When we run NGRL with violation counting on the pacifist merchant case study, however, we get the correct behaviour; that is, the behaviour depicted in Figure 6.4. The action *fight* is excluded in step 2 in the procedure on page 137, leaving the agent to choose *unload*. This is in general how normative deadlock will be handled; in the case where $Q_\mathcal{N}$ is negative and identical for two or more actions, $VC_\mathcal{N}$ will be used to judge them based on how many violations each action causes.

This method for augmenting NGRL incurs virtually no additional computational cost, because $R_\mathcal{N}(s, a)$ and $VC_\mathcal{N}(s, a)$ can both be computed from $viol(\overline{s}, \mathcal{N})$. Despite being cost-free, this technique allows us to cope with CTD obligationss, normative deadlock, and indirect normative conflict, making NGRL a much more normatively resilient technique than it is in the absence of violation counting. In addition, it can, for the most part, replace online compliance checking in filtering out non-compliant actions whenever possible; as we showed in the proof of Theorem 6.3.1, the set of actions with minimal VC-values are normatively optimal.

However, if we discard online compliance checking, we lose its function as an event recorder and the transparency that lies therein. We can compromise to some degree; should we still want violation reports to be generated, for example, we could configure the agent to perform online compliance checking whenever $VC_\mathcal{N}(s, a) > 0$. Moreover, we can learn something significant about the compliance of a state action pair $\overline{s} = (s, a)$ just from the content of $VC_\mathcal{N}(s, a)$: we can learn how many violations have been recorded for that state. Again, if that value is positive, we can examine that state-action pair more closely with the normative supervisor.

However, there are flaws in this technique we cannot circumvent. For instance, if there is

some state action pair $\overline{s} = (s, a)$ that is not visited during training, $VC_\mathcal{N}(s, a)$ will remain initialized as 0; thus, for any new $\overline{s}$ visited at operation time, its VC-value will indicate that it is compliant. This potential pitfall will become an issue in larger environments where it is plausible that not all $\overline{s}$ were visited during training. It is notable that this remains an issue with tabular Q-learning in general; which is precisely why techniques like function approximation are employed in larger or more complex environments.

However, it is not clear how NGRL with violation counting can be extended to Q-learning with function approximation. Recall that the TLQ-learning NGRL approach was incompatible with linear function approximation (since the agent only chooses from actions which maximize $Q_\mathcal{N}$, the agent selects the course of action where there is no risk of violation – staying in place). If this technique is to be useful in more complex environments, a way to adapt NGRL with TLQ-learning to agents which use function approximation would be necessary.

6.4 Final Remarks

In the previous chapter, we introduced online compliance checking (OCC), a technique that, while accommodating changing normative systems and transparency measures, features a policy completely decoupled from the imposed normative constraints. This results in a few notable issues: most significantly, a policy that is no longer optimal and fails to avoid normative deadlock. Since the agent learns its policy in one MDP and operates into another (that is, the MDP resulting from the agent's actions being restricted to those that are normatively optimal), the resulting policy is being enacted in an MDP it is not optimal for. At the same time, since the policy does not take into account the objective of normative compliance, it does not implement a plan that avoids situations in which compliance is not possible. As a result, we have an agent that under-performs for both its primary and normative objective.

In this chapter, we have attempted to address these challenges, with progressively increasing degrees of success. We started with what is perhaps the simplest possible solution to the issues arising from the decoupling of the agent and the normative supervisor; constraining the agent's learning by performing compliance checking in each step during training. This way, the agent both learns and operates in the same MDP – what we have called a *sheltered MDP*. This method was somewhat successful under certain conditions; namely, our experiments showed that training in a sheltered MDP in an environment that does not allow for normative deadlock and under a normative system without any "loopholes" will significantly augment the agent's performance with respect to its primary objective. However, the presence of normative deadlock erases the effectiveness of the technique, and the presence of loopholes in the normative system will ensure the agent learns to exploit these loopholes, making OCC *less* effective. It is also notable that training in a sheltered MDP is almost completely ineffective in the absence of OCC. Overall, this technique is not worth the computational expenditure (which is the same as OCC) required.

Norm-guided reinforcement learning (NGRL), on the other hand, showed much more promise. This technique leverages multi-objective RL (MORL) to learn normative behaviour, and entails blending the primary objective with a normative objective defined by a reward function that assigns a punishment to each state-action pair for which the normative supervisor detects a violation. We introduced two means of consolidating behaviours maximizing both the primary and normative objectives; MORL via linear scalarization, and MORL via thresholded lexiographic Q-learning (TLQL). We saw through a series of experiments that this technique does what sheltered MDPs could not, teaching the agent to effectively pursue both its primary objective and compliance to a normative system. NGRL, when used as a precursor to OCC, remedied the problems we identified with OCC, improving performance with respect to both the primary and normative objective. Even when used on its own, it was fairly effective at balancing the primary and normative objectives and avoiding violations. This technique can be seen as an example of the reward engineering approach to learning normative behaviour, but since it uses the normative supervisor to dynamically and automatically determine the compliance of state-action pairs, it can easily accommodate complex normative systems containing constitutive norms, direct normative conflict, and strong permissions, unlike most of the approaches we reviewed in Chapter 4. However, it is not capable of accommodating all normative systems. We saw clearly that NGRL cannot deal with normative deadlock, and does not teach the agent compliance with contrary-to-duty obligations, failing to capture the norm violation facet of normative resilience. Additional shortcomings of NGRL on its own included the requirement to re-train each time the normative system changes – a rather inconvenient necessity if we have a normative system that changes not infrequently – and the fact that compliance to the normative system is collapsed into a simple utility function, hampering transparency in the way predicted by [AKS17].

In response to these shortcomings of NGRL, we introduced violation counting, which requires that we perform NGRL with the second reasoner model and via TLQL. Violation counting entails that – for each state transition occurring during training – when the normative supervisor detects a violation and the normative Q-function $Q_\mathcal{N}$ is updated, a function that maps every state-action pair to the number of violations incurred by that state action pair is also updated. Recording the number of violations that occur for each state-action pair during training allows the agent to cope with normative deadlock at operation time by choosing among non-compliant actions those actions that result in the fewest violations. However, picking actions which maximize the normative objective is still prioritized; we proved that maximizing the normative objective reduces violations in the long term. Additionally, having a function that tells us for every state-action pair whether $|viol(\bar{s}, \mathcal{N})| > 0$ gives us a clear indicator of whether a violation has occurred; when such a violation is detected, we can use the normative supervisor to check and generate a violation report for the specific state-action pair in question, allowing for some degree of transparency. However, the inability to handle normative system change without completely retraining the agent remains. What is more, it is not clear how to scale this technique. Because NGRL with violation counting is based on the TLQL-based

NGRL approach, there is no obvious way to implement NGRL with violation counting with function approximation (recall that the TLQL agent, when trained with linear function approximation, did not move from its initial position). If we are to scale the technique, we will need to develop novel methods to do so.

Throughout this journey toward learning normative behaviour, we have been able to gradually refine our approach to teaching norm compliance to RL agents. It was clear that the solution to the problems arising from OCC lay in incorporating the normative supervisor at the training phase, but the question of how, precisely, we can do that has been addressed in this chapter. After experimenting with one technique that created more problems than it solved, we were able to acheive some modicum of success with NGRL. Augmenting NGRL with violation counting solved two of the problems intrinsic to NGRL (its inability to deal with normative deadlock, and its lack of transparency), but this augmentation renders the technique incompatible with function approximation. This issue is part of a more general weakness of the techniques leveraging the normative supervisor which we have presented: a lack of scalability. Running a theorem prover thousands if not millions of times is a very computationally heavy endeavour, and a good next step would be to explore approaches where we can minimize use of the theorem prover. In the next chapter, we will briefly note this possibility for future research along with several others.

Conclusions

Over the past 6 chapters, we have travelled far. From the motivational roots of this dissertation, we isolated important goals and items of inquiry, presenting the aspirations of this work. These aspirations were largely centred on expanding the range of normative behaviours implementable through ostensibly "ethical" reinforcement learning methods, through the adoption of tools that would allow us to produce agents displaying normative resilience – a toleration of norm conflict, norm violation, and normative system change – and transparency in their norm-compliant decision making.

After establishing the necessary groundwork, we proceeded to look at how others have tried to tackle the problem of designing ethical reinforcement learning agents, analysing them and dissecting their shortcomings. We introduced a possible alternative route to normatively compliant RL through the established field of safe RL, and examined carefully the prospect of using LTL specifications to constrain RL agents with norms. Though we showed that representing obligations directly with LTL is impossible, we also discussed the alternative notion of a compliance specification, for which we introduced a synthesisation algorithm for transforming a normative system expressed in defeasible deontic logic into a compliance specification. Compliance specifications have limits, though, which we exposited on in detail.

We then introduced the framing of our novel approach to designing RL agents with normative reasoning capabilities – normative interference. Within this framing we introduced the architecture of a module for normative reasoning – the normative supervisor, which utilizes a theorem prover and translators to supply information to a reinforcement learning agent on which of the actions it can take are compliant and which are not. We first interfered with the point at which the policy chooses an optimal action; instead of inputting into the policy the list of possible actions, we input a list of normatively optimal actions created for the current state by the normative supervisor. Such a list is composed only of compliant actions, if any exist; if there are no compliant actions, the supervisor supplies instead a set of actions which violate the applicable normative system as little as

possible. We called this online compliance checking. We then attempted a second point of normative interference, by removing non-normatively-optimal actions during training, teaching the agent within what we called a sheltered MDP; we discarded this approach, since it caused more problems than it solved. Finally, we interfered with the reward signal of an RL agent during the training phase, adding an additional "non-compliance" function and situating the problem within a multi-objective MDP we call a compliance MDP. We called this norm-guided reinforcement learning. This proved to be a promising approach, especially after the addition of violation counting functions, which allowed us to better cope with features of normative resilience stemming from norm violation.

Both online compliance checking and norm-guided reinforcement learning had strengths and weaknesses, which were often complementary. Online compliance checking proved to be highly normatively resilient and transparent; many different normative systems were accommodated with ease, and switching between systems was seamless. However, this technique does not incorporate knowledge of applicable norms when it is learning its plans, and as a result, the agent exhibits sub-optimal behaviour and fails to plan ahead to avoid states of normative deadlock. Norm-guided reinforcement learning, on the other hand, balanced the objective of norm-compliance with the agent's other norm-agnostic objective, teaching the agent plans that incorporate both (while prioritizing norm compliance). The agent was then able to plan ahead to avoid situations where no compliance is possible. However, this technique failed to capture some aspects of normative resilience – which we did manage to remedy through the introduction of violation counting functions – and was much less transparent than online compliance checking. Additionally, it does not provide much of a guarantee that immediately non-compliant behaviours would be curbed.

Fortunately, online compliance checking and norm-guided reinforcement learning can be used complementarily. With norm-guided reinforcement learning with violation counting, we can teach the agent to act in a compliant way, teaching norm-conformance in a way that accommodates normative resilience, and with online compliance checking, we can bolster this learned compliance with more guarantees, supplying also a means to facilitate accountability for the agent by providing violation reports. Of course, using a theorem prover at every time step during and after training incurs significant computational cost. Because of this, these techniques will have limited utility in practise. Nevertheless, these techniques and our exploration of them have provided insights into the challenges – measuring degrees of non-compliance, planning with norms, tracking compliance without significant computational expenditure, and retraining for normative system change, to name some of the most prominent – that must be met in order to bestow RL agents with a capability for normative reasoning and the quality of normative resilience, and shed some light on how they might be tackled.

7.1 Future Work

This dissertation does not contain an exhaustive pursuit of every available avenue of the work presented. Below, in closing, we will give some final attention to some lanes of

research that could continue this work.

Compliance Specifications in LDL_f

In Chapter 4, we saw how to synthesize compliance specifications in LTL from a normative system expressed in DDL. We encountered some difficulties there; translating actions into combinations of state labels proved problematic, and there were limits to how well CTD obligations could be handled. We were not able to model sub-ideal CTDs, and though we were able, in some cases, to model weak compliance – accommodating compensatory CTDs in certain cases – we had to resort to removing the primary obligation altogether, which will not work in every case (see the preconditions for Proposition 4.3.2). The latter issue with CTDs is a function of LTL's limited expressivity.

LTL may be the most popular formalism for constraining RL, but it is not the only logic that has been utilized for that purpose. Linear Dynamic Logic over finite traces (LDL_f) [DGV13] has been proposed as a logic for constraining RL in [DGFI$^+$20]. LDL is an extension of LTL which employs the structure of Propositional Dynamic Logic (PDL) to specify formulas over traces. [DGDMG$^+$14] shows how LDL_f can be used to define *metaconstraints*; these are defined in the context of run-time monitoring semantics which evaluate formulas to *true* (the formula holds in the current state, and no extension of the trace can change that), *false* (the formula is false in the current state, and no extension of the trace can change that), *temp_true* (the formula true in the current state, but that can change), or *temp_false* (the formula is false in the current state, but that can change). [DGDMG$^+$14] shows how statements about the status of a specification with respect to these values can be encoded in LDL_f, facilitating the the specification of a constraint only triggered when another constraint is violated. This provides a natural way to express CTDs (namely, compensatory CTDs).

Preliminary work on combining LDL_f with RL has been put forward in [DGFI$^+$20], and LDL_f compliance specifications might prove a plausible replacement for LTL compliance specifications. It may be worthwhile to pursue an algorithm for synthesizing LDL_f formulas from normative systems.

Temporal Reasoning and Norms

In this dissertation, we have restricted our attention to maintenance obligations; we have ignored achievement obligations, and norms with deadlines. Being able to model the interplay between these different kinds of norms would greatly expand the number of normative systems we could work with. In order to do this, though, an effective temporal deontic logic and the means to automate reasoning ith it must be pursued.

Normative Multi-Agent Systems

We noted in Chapter 2 that normative systems represent an ideal social reality; this implies that at their core, normative systems are concerned not only with how agents

behave individually, but also with how agents *interact.* It is precisely the presence of other agents that shapes the normative systems that we have in this dissertation applied to a single agent. Normative multi-agent system (NorMAS) is a well established field in AI which has put forward many tools for modelling normative relationships between autonomous agents. Complementarily, there are several methods for multi-agent reinforcement learning. An interesting future research direction would be to explore whether the approaches taken in this dissertation could be expanded to the multi-agent case.

Computationally Feasible NGRL and OCC

One key shortcoming of online compliance checking and norm-guided reinforcement learning is computational cost; it is expensive to run a theorem prover at every time step. We have demonstrated the usefulness of a mechanism for automatically assigning punishments based on normative system violations and for monitoring an agent during operation; however, more efficient methods for accomplishing this must be pursued if the approach proposed in this dissertation is to be used more freely.

NGRL with Deep Reinforcement Learning

For very complex or involved learning tasks, often deep reinforcement learning is required. Deep reinforcement learning combines RL with, e.g., artificial neural networks (ANNs). We already encountered challenges with extending NGRL to RL with linear function approximation; presumably, we will encounter even more when we add ANNs into the mix. Still, this would be a worthwhile avenue to pursue, and would greatly expand the applicability of NGRL.

Expanding Available Case Studies

In this dissertation, we limited ourselves to simple demonstrations via case studies in the form of automated games. In the long term, however, if we would like to adapt these methods for use in wider society, they will likely need to be compatible with deep reinforcement learning (see above), and we will need to experiment with learning and enforcing normative behaviour in more complex environments, like driving simulators, where driving regulations could be modelled.

Synthesis of Policies Satisfying LTL Constraints

A.1 ω-Automata

An ω-automaton is an automaton over infinite words; essentially, given an infinite word over an alphabet Σ, an ω-automaton decides whether to *accept* a run in which the word is fed to the automaton, or not.

Perhaps the most straightforward example of a ω-automaton is the generalized Büchi automaton (GBA) [CVWY92] $\mathcal{A} = \langle Q, q_0, AP, \delta, \mathcal{F} \rangle$. These automata consist of a finite set of states Q, an initial state q_0, a finite alphabet Σ (the automaton runs over words $\sigma \in \Sigma^\omega$), a transition function $\delta : Q \times \Sigma \to 2^Q$, and an acceptance condition $\mathcal{F} = \{F_1, ..., F_n\}$ where each $F_i \subseteq Q$ is a set of states. A run is accepting if at least one of the states $s \in F_i$ is visited infinitely often during that run (for each F_i). In other words, given a word $\sigma \in \Sigma^\omega$, we can sequentially feed each letter $\sigma[i] \in \Sigma$ into the automaton at state q_i of a run $\overline{q} \in Q^\omega$, upon which the automaton will use the transition function to determine the possibilities for the next state $q_{i+1} \in \delta(q_i, \sigma[i])$. If there is some $q_i \in F_j$ that is visited infinitely many times (and this is true for all F_j), the automata accepts the word σ associated with the run $\overline{q}$.

It has been known for quite some time that it is possible to translate LTL formulas into GBA (see [GO01], for example); unfortunately, this kind of non-deterministic automaton has limited uses for probabilistic model checking and related tasks.

From here, we will describe two types of ω-automata that have been used repeatedly in the field of probabilistic model checking and in the literature regarding synthesizing RL policies that maximize the probability of satisfaction of an LTL formula.

149

A.1.1 Limit-Deterministic Büchi Automata

Limit deterministic Büchi automata (LDBA) [CY95] are GBA in which the the set of states can be partitioned into two sets; i.e., $Q = Q_N \cup Q_D$ and $Q_N \cap Q_D = \varnothing$. Q_D is a special deterministic subset of states such that for every $q \in Q_D$ and $\alpha \in \Sigma$, $|\delta(q, \alpha)| = 1$. Furthermore, for every $F_i \in \mathcal{F}$, $F_i \subseteq Q_D$. Meanwhile, $q_0 \in Q_N$ and and all the transitions from Q_N to Q_D are non-deterministic ϵ-transitions (transitions that do not require any input).

LTL formula can be turned into an LDBA, for example by using the algorithm described in [SEJK16]. LTL formulae transformed into LDBA were used effectively for learning RL policies satisfying LTL constraints in [HKA+19, HAK20, HKA20].

A.1.2 Deterministic Rabin Automata

Deterministic Rabin automata (DRA) [Rab69] have a set of states Q and a finite alphabet Σ as above, but its transition function is completely deterministic; that is, the transition function's signature is $\delta : Q \times \Sigma \to Q$. The accepting conditions are called *Rabin pairs* $\mathcal{F} = \{(F_1, G_1), ..., (F_n, G_n)\}$; a Rabin pair $(F_i, G_i) \in Q \times Q$ is satisfied by a word σ if the run $\bar{q}$ it induces visits a state in F_i infinitely many times and visits no state in G_i infinitely many times. In an accepting run, at least one of these Rabin pairs must be satisfied.
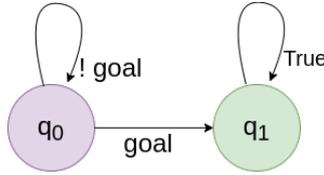
Figure A.1: The DRA $\mathcal{A}$ for the above formula $F(goal)$. There is one Rabin pair (F, G), where the purple state (q_0) is in G and the green state (q_1) is in F.

LTL formulas can be translated into DRA by means of, for example, the method described in [BBKS13]. DRA have been used in a number of approaches for synthesizing RL policies that maximize the probability of satisfying an LTL formula, including [DSBR11b, FT14, SKC+14, KS18].

A.2 Policy Synthesis

In [DSBR11b, DSBR11a], the authors introduce a method for taking a labelled MDP and synthesizing a policy that maximizes the probability of a given LTL formula of being satisfied over the sequence of state labels generated by following that policy. Below, we will walk through (a high level rendering) of what [FT14] call the standard quantitative synthesis method for MDPs with LTL specifications.

Most approaches to learning RL policies that satisfy an LTL formula with maximal probability rely on the concept of a *product MDP*.

Definition A.2.1 (Product MDP)**.** *Given a labelled MDP $\mathcal{M} = \langle S, s_0, A, P, L \rangle$ and an automaton $\mathcal{A} = \langle Q, q_0, AP, \delta, \mathcal{F} \rangle$ corresponding to an LTL formula ϕ, we define a product MDP $\mathcal{M}^{\times} = \langle S^{\times}, s_0^{\times}, A^{\times}, P^{\times}, \mathcal{F}^{\times} \rangle$, where $S^{\times} = S \times Q$, $s_0^{\times} = (s_0, q_0)$, $A^{\times}((s, q)) = A(s)$, and*

$$P^{\times}((s, q), a, (s', q')) = \begin{cases} P(s, a, s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases}$$

$\mathcal{F}^{\times}$ is $\mathcal{A}$'s acceptance condition over $S^{\times}$.

There are multiple ways to progress from here. In [SKC+14], where the authors employ DRAs to create a product MDP, a collection of reward functions is designed for the accepting condition $\mathcal{F}^{\times} = \{(F_1^{\times}, G_1^{\times}), ..., (F_n^{\times}, G_n^{\times})\}$, where one reward function corresponds to each $(F_i^{\times}, G_i^{\times})$. Each reward function assigns a positive reward w_F when a state in some $F_i^{\times}$ is visited, a negative reward w_G when a state in the corresponding $G_i^{\times}$ is visited, and a neutral reward (valuing 0) otherwise. A depiction of the underlying product MDP in this approach is shown in Figure A.2. A temporal difference learning algorithm is introduced that utilizes the resulting reward functions to learn a policy that maximizes the probability of reaching the acceptance condition, thereby maximizing the probability of satisfying ϕ by following the policy.

In [HAK18, HKA+19, HAK20], the authors employ LDBAs instead. A similar approach is taken, where reward shaping is once again employed, based on the LDBA acceptance condition. A positive reward is bestowed each time an accepting state is visited, and a neutral reward whenever another state is visited. The result is the same, where model-free RL can be used to learn a policy that maximizes the probability of visiting an accepting state infinitely often should it be followed, thereby satisfying ϕ.
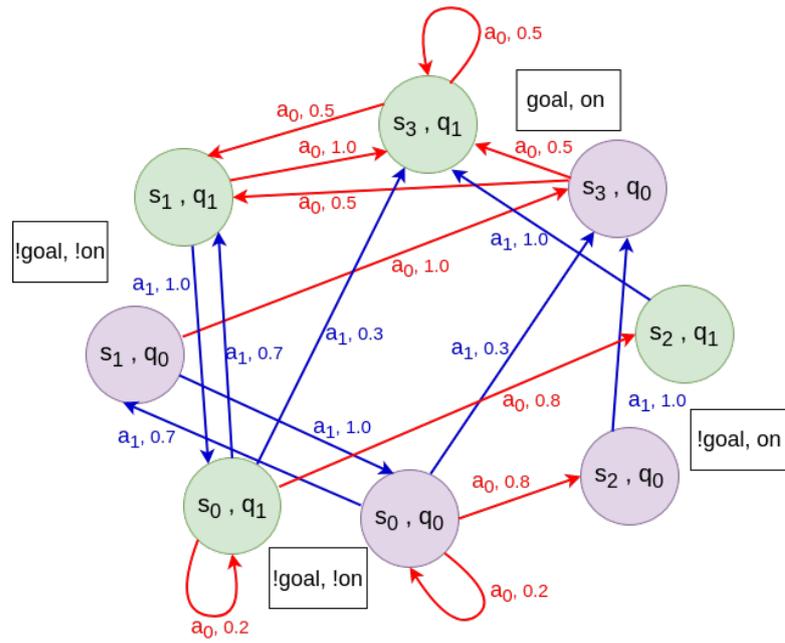
Figure A.2: The product MDP $M^\times$ generated from the DRA $\mathcal{A}$ in Figure A.1 and the labelled MDP $\mathcal{M}$ in Figure 2.2 (for clarity, transitions with action a_0 are in red and transitions with action a_1 are in blue). According to the approach in [SKC+14], in the green states $s^\times = (s_i, q_j)$ in $F^\times$, the positive w_F is assigned; in the purples states in $G^\times$, the negative reward w_G is assigned.

Synthesis Algorithm

Below is the code for selected methods from the class `Synthesizer.java`.

```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

import spindle.core.dom.*;
import supervisor.games.Environment;
import supervisor.normsys.*;
import supervisor.reasoner.DDPLReasonerCore;
import util.ProjectUtils;

public class Synthesizer {
    Mode obl = new Mode("O", false);
    Mode perm = new Mode("P", false);
    ProjectUtils util = new ProjectUtils();
    NormBase NS;
    ArrayList<String> AP;
    ArrayList<Transition> transitions;
    DDPLReasonerCore reasoner = new DDPLReasonerCore();
    Environment env;
    DDPLTranslator2 translator;
    ArrayList<ArrayList<String>> badStates = new
        ArrayList<ArrayList<String>>();
    HashMap<ArrayList<String>, String> mandatory = new
        HashMap<ArrayList<String>, String>();
    HashMap<ArrayList<String>, String> prohibited = new
        HashMap<ArrayList<String>, String>();
```

```
public Synthesizer(NormBase nb, ArrayList<String> ap,
  ArrayList<Transition> trans) {
  NS = nb;
  translator = new DDPLTranslator2(NS);
  AP = ap;
  transitions = trans;
  env = new Environment();
  env.configureLabels(AP);
  ArrayList<String> acts = new ArrayList<String>();
  for (Transition t : transitions) {
    acts.add(t.getAction());
  }
}


public boolean checkCompliance() {
  Theory copy = reasoner.getTheory().clone();
  Map<Literal, Map<ConclusionType, Conclusion>> conclusions =
      reasoner.drawConclusions()
        .entrySet().stream()
        .filter(p -> util.checkForLit(copy.getAllLiteralsInRules(),
          p.getKey().getName()))
        .collect(Collectors.toMap(map -> map.getKey(), map ->
          map.getValue()));
  Collection<Map<ConclusionType, Conclusion>> concl =
      conclusions.values();
  concl.removeIf(p ->
      !p.keySet().contains(ConclusionType.DEFEASIBLY_PROVABLE));
  Map<Literal, Map<ConclusionType, Conclusion>> obls = conclusions
        .entrySet().stream()
        .filter(p -> p.getKey().getMode().equals(obl))
        .filter(p -> concl.contains(p.getValue()))
        .collect(Collectors.toMap(map -> map.getKey(), map ->
          map.getValue()));
  Map<Literal, Map<ConclusionType, Conclusion>> bes = conclusions
        .entrySet().stream()
        .filter(p -> !p.getKey().getMode().equals(obl) )
        .filter(p -> concl.contains(p.getValue()))
        .collect(Collectors.toMap(map -> map.getKey(), map ->
          map.getValue()));
  obls.keySet().forEach(p -> p.removeMode());
  return bes.keySet().containsAll(obls.keySet());
}

public void findBadStates() throws RuleException {
  ArrayList<ArrayList<String>> subsets = getSubsets();
  for (ArrayList<String> gamma : subsets) {
    updateReasoner(gamma);
```

```java
        if(!checkCompliance()) {
          badStates.add(gamma);
        }
        for(String a : util.getActionList(env)) {
          Literal act = new Literal(a);
          Rule fact = new Rule(a, RuleType.DEFEASIBLE);
          fact.addHeadLiteral(act);
          ArrayList<Rule> f = new ArrayList<Rule>();
          f.add(fact);
          updateReasoner(gamma);
          reasoner.addFacts(f);
          if(checkCompliance()) {
            if (badStates.contains(gamma)){
              badStates.remove(gamma);
              for(Transition t : transitions) {
                if(t.getAction().equals(a)) {
                  mandatory.put(gamma, t.getTransition());
                }
              }
            }
          }
          else {
            if (!badStates.contains(gamma)){
              for(Transition t : transitions) {
                if(t.getAction().equals(a)) {
                  prohibited.put(gamma, t.getTransition());
                }
              }
            }
          }
        }
      }
    }
  }


}
```

List of Figures

158

List of Tables

List of Algorithms

Bibliography

[AA07]   Michael Anderson and Susan Leigh Anderson. Machine ethics: Creating an ethical intelligent agent. *AI Magazine*, 28(4):15, 2007.

[ABE+18]   Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[ABGM01]   Grigoris Antoniou, David Billington, Guido Governatori, and Michael Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic*, 2:255–287, 2001.

[ADL18]   Natasha Alechina, Mehdi Dastani, and Brian Logan. Norm specification and verification in multi-agent systems. *Journal of Applied Logics*, 5(2):457, 2018.

[AG13]   Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. In *International conference on machine learning*, pages 127–135. PMLR, 2013.

[AGS11]   Michael Abraham, Dov M Gabbay, and Uri Schild. Obligations and prohibitions in talmudic deontic logic. *Artificial Intelligence and Law*, 19(2):117–148, 2011.

[AKS17]   Thomas Arnold, Daniel Kasenberg, and Matthias Scheutz. Value alignment or misalignment - what will keep systems accountable? In *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence*, volume WS-17 of *AAAI Technical Report*. AAAI Press, 2017.

[AML16]   David Abel, James MacGlashan, and Michael L Littman. Reinforcement learning as a framework for ethical decision making. In *AAAI Workshop: AI, Ethics, and Society*, volume 16, 2016.

[Asi42]   Isaac Asimov. Runaround, 1942.

[BBKS13]   Tomáš Babiak, František Blahoudek, Mojmír Křetínskỳ, and Jan Strejček. Effective translation of ltl to deterministic rabin automata: Beyond the (f, g)-fragment. In *Automated Technology for Verification and Analysis*, pages 24–39. Springer, 2013.

[BBMR19]   Avinash Balakrishnan, Djallel Bouneffouf, Nicholas Mattei, and Francesca Rossi. Incorporating behavioral constraints in online ai systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3–11, 2019.

[BCE⁺13]   Jan M. Broersen, Stephen Cranefield, Yehia Elrakaiby, Dov M. Gabbay, Davide Grossi, Emiliano Lorini, Xavier Parent, Leendert W. N. van der Torre, Luca Tummolini, Paolo Turrini, and François Schwarzentruber. Normative reasoning and consequence. In *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*, pages 33–70. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[BdCPD⁺13] Tina Balke, Célia da Costa Pereira, Frank Dignum, Emiliano Lorini, Antonino Rotolo, Wamberto Vasconcelos, and Serena Villata. Norms in mas: definitions and related concepts. In *Dagstuhl Follow-Ups*, volume 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[Bel52]   Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8), 1952.

[BK08]   Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT press, 2008.

[BMB⁺18]   Edvard P. Bjørgen, Simen Madsen, Therese S. Bjørknes, Fredrik V. Heimsæter, Robin Håvik, Morten Linderud, Per-Niklas Longberg, Louise A. Dennis, and Marija Slavkovik. Cake, death, and trolleys: Dilemmas as benchmarks of ethical decision-making. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '18, page 23–29, 2018.

[BN08]   Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*, pages 41–47, 2008.

[Bos14]   Nick Bostrom. Superintelligence: Paths, dangers, strategies, 2014.

[BPvdT20]   Christoph Benzmüller, Xavier Parent, and Leendert van der Torre. Designing normative theories for ethical and legal reasoning: Logikey framework, methodology, and tool support. *Artificial Intelligence*, page 103348, 2020.

164

[BvdT03]   Guido Boella and Leendert van der Torre. Permissions and obligations in hierarchical normative systems. In *ICAIL*, 2003.

[Chi63]   Roderick M Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24(2):33–36, 1963.

[CJ02]   José Carmo and Andrew JI Jones. Deontic logic and contrary-to-duties. In *Handbook of philosophical logic*, pages 265–343. Springer, 2002.

[CM14]   Bibhas Chakraborty and Susan A Murphy. Dynamic treatment regimes. *Annual review of statistics and its application*, 1:447, 2014.

[CVWY92]   Costas Courcoubetis, Moshe Vardi, Pierre Wolper, and Mihalis Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal methods in system design*, 1(2):275–288, 1992.

[CY95]   Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.

[DBB+18]   Virginia Dignum, Matteo Baldoni, Cristina Baroglio, Maurizio Caon, Raja Chatila, Louise Dennis, Gonzalo Génova, Galit Haim, Malte S. Kließ, Maite Lopez-Sanchez, Roberto Micalizio, Juan Pavón, Marija Slavkovik, Matthijs Smakman, Marlies van Steenbergen, Stefano Tedeschi, Leon van der Toree, Serena Villata, and Tristan de Wildt. Ethics by design: Necessity or curse? In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '18, page 60–66. Association for Computing Machinery, 2018.

[DGDMG+14]   Giuseppe De Giacomo, Riccardo De Masellis, Marco Grasso, Fabrizio Maria Maggi, and Marco Montali. Monitoring business metaconstraints based on ltl and ldl for finite traces. In *Business Process Management*, pages 1–17. Springer International Publishing, 2014.

[DGFI+20]   Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi, and Alessandro Ronca. Temporal logic monitoring rewards via transducers. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 17, pages 860–870, 2020.

[DGV13]   Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery, 2013.

[Dig17]   Virginia Dignum. Responsible autonomy. In *IJCAI*, 2017.

[DK14]   John DeNero and Dan Klein. UC Berkeley CS188 intro to AI – course materials, 2014.

[DSBR11a]   Xu Chu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. Mdp optimal control under temporal logic constraints. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 532–538. IEEE, 2011.

[DSBR11b]   Xu Chu Dennis Ding, Stephen L Smith, Calin Belta, and Daniela Rus. Ltl control in uncertain environments with probabilistic satisfaction guarantees. *IFAC Proceedings Volumes*, 44(1):3515–3520, 2011.

[EL21]   Adrien Ecoffet and Joel Lehman. Reinforcement learning under moral uncertainty. In *International Conference on Machine Learning*, pages 2926–2936. PMLR, 2021.

[FF22]   Elisa Freschi and Marco Ferrante. The principle of suspension (*bādha*) in early mīmāṃsā: the case of *prāpta-* and *aprāptabādha*. *Submitted*, 2022.

[FH70]   Dagfinn Føllesdal and Risto Hilpinen. Deontic logic: An introduction. In *Deontic logic: Introductory and systematic readings*, pages 1–35. Springer, 1970.

[Foo67]   Philippa Foot. The problem of abortion and the doctrine of the double effect. *Oxford review*, 5, 1967.

[For84]   James William Forrester. Gentle murder, or the adverbial samaritan. *The Journal of Philosophy*, 81(4):193–197, 1984.

[FT14]   Jie Fu and Ufuk Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. In Dieter Fox, Lydia E. Kavraki, and Hanna Kurniawati, editors, *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, 2014.

[GF15]   Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[GH15]   Guido Governatori and Mustafa Hashmi. No time for compliance. In *Proc. of EDOC*, pages 9–18. IEEE, 2015.

[GHRR07]   Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Proc. of AUSAI*, LNCS, pages 486–496, 2007.

[GKS98]   Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning.*, volume 98, pages 197–205, 1998.

[GLPA22]   Umberto Grandi, Emiliano Lorini, Timothy Parker, and Rachid Alami. Logic-based ethical planning. *arXiv preprint arXiv:2206.00595*, 2022.

166

[GMAB04]   Guido Governatori, Michael Maher, Grigoris Antoniou, and David Billington. Argumentation semantics for defeasible logic. *Journal of Logic and Computation*, 14:675–702, 2004.

[GO01]   Paul Gastin and Denis Oddoux. Fast ltl to büchi automata translation. In *International Conference on Computer Aided Verification*, pages 53–65. Springer, 2001.

[GORS13]   Guido Governatori, Francesco Olivieri, Antonino Rotolo, and Simone Scannapieco. Computing strong and weak permissions in defeasible logic. *Journal of Phil. Logic*, 42(6):799–829, 2013.

[Gov15]   Guido Governatori. Thou shalt is not you will. In *Proceedings of the 15th International Conference on Artificial Intelligence and Law*, pages 63–68, 2015.

[Gov18]   Guido Governatori. Practical normative reasoning with defeasible deontic logic. In *Reasoning Web International Summer School*, pages 1–25. Springer, 2018.

[GR06]   Guido Governatori and Antonino Rotolo. Logic of violations: A gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 4:193–215, 2006.

[GR08a]   Guido Governatori and Antonino Rotolo. Bio logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, 17(1):36–69, 2008.

[GR08b]   Guido Governatori and Antonino Rotolo. BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Journal of Autonomous Agents and Multi Agent Systems*, 17(1):36–69, 2008.

[GRC12]   Guido Governatori, Antonino Rotolo, and Erica Calardo. Possible world semantics for defeasible deontic logic. In *DEON*, 2012.

[GRS05]   Guido Governatori, Antonino Rotolo, and Giovanni Sartor. Temporalised normative positions in defeasible logic. In *Proceedings of the 10th international conference on Artificial intelligence and law*, pages 25–34, 2005.

[HAK18]   Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[HAK20]   Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Cautious reinforcement learning with logical constraints. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent*

Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020, pages 483–491, 2020.

[HKA+19]   Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 5338–5343. IEEE, 2019.

[HKA20]   Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–22. Springer, 2020.

[HM13]   Risto Hilpinen and Paul McNamara. Deontic logic: A historical survey and introduction. *Handbook of deontic logic and normative systems*, 1:3–136, 2013.

[Hor01]   John F Horty. *Agency and deontic logic*. Oxford University Press, 2001.

[HR07]   Ian Hodkinson and Mark Reynolds. Temporal logic. In Patrick Blackburn, Johan Van Benthem, and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3, pages 655–720. Elsevier, 2007.

[HRB+22]   Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):1–59, 2022.

[JIV19]   Anna Jobin, Marcello Ienca, and Effy Vayena. The global landscape of ai ethics guidelines. *Nature Machine Intelligence*, 1(9):389–399, 2019.

[JKJ+20]   Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe Reinforcement Learning Using Probabilistic Shields (Invited Paper). In *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:16, 2020.

[JP85]   Andrew JI Jones and Ingmar Pörn. Ideality, sub-ideality and deontic logic. *Synthese*, pages 275–290, 1985.

[JS93]   Andrew JI Jones and Marek Sergot. On the characterisation of law and computer systems: The normative systems perspective. *Deontic logic in computer science: normative system specification*, pages 275–307, 1993.

[KBP13]   Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[KS18]   Daniel Kasenberg and Matthias Scheutz. Norm conflict resolution in stochastic domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[LG09]   Ho-Pun Lam and Guido Governatori. The making of spindle. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, volume 5858 of *LNCS*, pages 315–322. Springer, 2009.

[LMFL15]   Jiaqi Li, Felipe Meneguzzi, Moser Fagundes, and Brian Logan. Reinforcement learning of normative monitoring intensities. In *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pages 209–223. Springer, 2015.

[Mah01]   Michael J Maher. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, 1(6):691–711, 2001.

[McN06]   Paul McNamara. Deontic logic. In *Handbook of the History of Logic*, volume 7, pages 197–288. Elsevier, 2006.

[MKS+15]   Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[Moo06]   James Moor. The nature, importance, and difficulty of machine ethics. *IEEE Intelligent Systems*, 21:18–21, 08 2006.

[MP90]   Zohar Manna and Amir Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 377–410, 1990.

[MP12]   Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer Science & Business Media, 2012.

[MRA+01]   Michael J Maher, Andrew Rock, Grigoris Antoniou, David Billington, and Tristan Miller. Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(04):483–501, 2001.

[MSA+15]   Bertram F Malle, Matthias Scheutz, Thomas Arnold, John Voiklis, and Corey Cusimano. Sacrifice one for the good of many? people apply different moral norms to human and robot agents. In *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 117–124. IEEE, 2015.

[Mur04]   Yuko Murakami. Utilitarian deontic logic. *AiML-2004: Advances in Modal Logic*, 287:287–302, 2004.

[MVDP22]   Paul McNamara and Frederik Van De Putte. Deontic Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2022 edition, 2022.

[NBC22]   Emery Neufeld, Ezio Bartocci, and Agata Ciabattoni. On normative reinforcement learning via safe reinforcement learning. In *PRIMA 2022*, 2022.

[NBCG21]   Emery Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. A normative supervisor for reinforcement learning agents. In *Proceedings of CADE 2021*, pages 565–576, 2021.

[NBCG22]   Emery A Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. Enforcing ethical goals over reinforcement-learning policies. *Journal of Ethics and Information Technology*, 2022.

[NBM+19]   Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. Teaching ai agents ethical values using reinforcement learning and policy orchestration. *IBM Journal of Research and Development*, 63(4/5):2–1, 2019.

[Neu22]   Emery A. Neufeld. Reinforcement learning guided by provable normative compliance. In *Proceedings of ICAART 2022*, pages 444–453, 2022.

[Nut93]   Donald Nute. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3. Oxford University Press, 1993.

[Nut01]   Donald Nute. Defeasible logic. In *International Conference on Applications of Prolog*, pages 151–169. Springer, 2001.

[OCSL16]   Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*, pages 2790–2799, 2016.

[Pes21]   Markus Peschl. Training for implicit norms in deep reinforcement learning agents through adversarial multi-objective reward optimization. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 275–276, 2021.

[PGR+11]   Monica Palmirani, Guido Governatori, Antonino Rotolo, Said Tabet, Harold Boley, and Adrian Paschke. Legalruleml: Xml-based rules and

norms. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 298–312. Springer, 2011.

[Pnu77]   Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[Rab69]   Michael O Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the american Mathematical Society*, 141:1–35, 1969.

[RDT15]   Stuart Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.

[RH16]   Mark O. Riedl and Brent Harrison. Using stories to teach human values to artificial agents. In *AI, Ethics, and Society, Papers from the 2016 AAAI Workshop*, volume WS-16-02 of *AAAI Technical Report*. AAAI Press, 2016.

[RSLSRA20] Manel Rodriguez-Soto, Maite Lopez-Sanchez, and Juan A Rodriguez-Aguilar. A structural solution to sequential moral dilemmas. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1152–1160, 2020.

[RSLSRA21] Manel Rodriguez-Soto, Maite Lopez-Sanchez, and Juan A Rodriguez-Aguilar. Multi-objective reinforcement learning for designing ethical environments. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 1–7, 2021.

[RSSLSRA22] Manel Rodriguez-Soto, Marc Serramia, Maite Lopez-Sanchez, and Juan Antonio Rodriguez-Aguilar. Instilling moral value alignment by means of multi-objective reinforcement learning. *Ethics and Information Technology*, 24(1):1–17, 2022.

[RVWD13]   Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

[SA21]   Colin Shea-Blymyer and Houssam Abbas. Algorithmic ethics: Formalization and verification of autonomous vehicle obligations. *ACM Trans. Cyber Phys. Syst.*, 5(4):38:1–38:25, 2021.

[SAPY17]   Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[SB98]   Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction.* Adaptive computation and machine learning. MIT Press, 1998.

[SB18]   Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[Sea69]   John Rogers Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge, England: Cambridge University Press, 1969.

[Seg21]   Samuel T Segun. Critically engaging the ethics of ai for a global audience, 2021.

[SEJK16]   Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic büchi automata for linear temporal logic. In *International Conference on Computer Aided Verification*, pages 312–332. Springer, 2016.

[SKC+14]   Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 1091–1096. IEEE, 2014.

[SSS+17]   David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[The19]   The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems. *IEEE standard review — Ethically aligned design: A vision for prioritizing human wellbeing with artificial intelligence and autonomous systems.* IEEE, first edition, 2019.

[TKS+20]   Suzanne Tolmeijer, Markus Kneer, Cristina Sarasua, Markus Christen, and Abraham Bernstein. Implementations in machine ethics: a survey. *ACM Computing Surveys (CSUR)*, 53(6):1–38, 2020.

[Var99]   Moshe Y Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *International AMAST Workshop on Aspects of Real-Time Systems and Concurrent and Distributed Software*, pages 265–276. Springer, 1999.

[VBC+19]   Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

172

[vBCF+21]   Kees van Berkel, Agata Ciabattoni, Elisa Freschi, Francesca Gulisano, and Maya Olszewski. The gentle murder paradox in sanskrit philosophy. In *Deontic Logic and Normative Systems - 15th International Conference, DEON 2020/21, Munich, Germany [virtual], July 21-24, 2021*, pages 17–35. College publications, 2021.

[VDB+11]   Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, 84(1):51–80, 2011.

[VDF+18]   Peter Vamplew, Richard Dazeley, Cameron Foale, Sally Firmin, and Jane Mummery. Human-aligned artificial intelligence is a multiobjective problem. *Ethics and Information Technology*, 20(1):27–40, 2018.

[VKCM16]   John Voiklis, Boyoung Kim, Corey Cusimano, and Bertram F Malle. Moral judgments of human vs. robot agents. In *2016 25th IEEE international symposium on robot and human interactive communication (RO-MAN)*, pages 775–780. IEEE, 2016.

[W+00]   Marco A Wiering et al. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

[Wat89]   Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[WL18]   Yueh-Hua Wu and Shou-De Lin. A low-cost ethics shaping approach for designing reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[WR19]   Aimee Wynsberghe and Scott Robbins. Critiquing the reasons for making artificial moral agents. *Science and Engineering Ethics*, 25, 06 2019.