TU WIEN Informatics

Representing Normative Reasoning in Answer Set Programming Using Weak Constraints

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Christian Hatschka, BSc.
Matrikelnummer 01525634

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr. Thomas Eiter
Mitwirkung: Prof. Dr. Agata Ciabattoni

Wien, 4. Oktober 2022

Christian Hatschka	Thomas Eiter

TU WIEN Informatics

Representing Normative Reasoning in Answer Set Programming Using Weak Constraints

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Christian Hatschka, BSc.
Registration Number 01525634

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Thomas Eiter
Assistance: Prof. Dr. Agata Ciabattoni

Vienna, 4th October, 2022

Christian Hatschka	Thomas Eiter

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Christian Hatschka, BSc.

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, 4. Oktober 2022

Christian Hatschka

Danksagung

Anfangs möchte ich mich bei meinen Betreuern Univ.Prof. Dr. Agata Ciabattoni und O.Univ.Prof. Dr. Thomas Eiter für ihre Unterstützung und ihren Rat ausdrücklichst bedanken. Des Weiteren will ich mich bei ihnen dafür bedanken, dass sie mir bei den Teilen dieser Arbeit, mit denen ich zu kämpfen hatte, mit Geduld und Hilfe zur Seite standen. Vor allem bin ich dankbar für ihr wiederholtes ausführliches Feedback, das es mir ermöglichte, diese Arbeit unter ihrer Aufsicht fertig zu stellen. Darüber hinaus möchte ich mich besonders bei Univ.Prof. Dr. Agata Ciabattoni dafür bedanken, dass sie mich mehrfach über Veranstaltungen informiert hat, die für meine Forschungsinteressen relevant sind, und es mir dadurch ermöglicht hat, meinen Horizont zu erweitern.

Zweitens möchte ich mich bei dem WWTF Projekt "Reasoning Tools for Deontic Logic and Applications to Indian Sacred Texts" für die finanzielle Unterstützung meiner Arbeit bedanken.

Drittens möchte ich Emery Neufeld dafür danken, dass er mir das Framework für seinen normativen Überwacher zur Verfügung gestellt hat, und für seine immense Geduld mit mir, als ich Schwierigkeiten hatte, Teile davon zu verstehen. Ohne seine Hilfe hätte ich die Implementierung unseres Reasoners in seinem Framework sicher nicht fertigstellen können.

Viertens möchte ich mich bei meinen Eltern und Großeltern für ihre Liebe und Unterstützung bedanken. Insbesondere möchte ich meinem Vater danken, der meine Arbeit mehrfach Korrektur gelesen und mich auf grammatikalische Fehler hingewiesen hat.

Schließlich möchte ich mich bei meinen Freunden für all die Hilfe bedanken, die ich von ihnen erhalten habe. Insbesondere möchte ich Nikita Kholodnyi danken, der mir mit wertvollem Rat zur Seite gestanden ist, wenn ich Fragen zu Java hatte. Auch will ich Esra Ceylan und Fatih Bozdemir danken, die meine Arbeit Korrektur gelesen haben und mir wichtiges Feedback gegeben haben.

Acknowledgements

Firstly, I would like to thank my advisors Univ.Prof. Dr. Agata Ciabattoni and O.Univ.Prof. Dr. Thomas Eiter for their guidance and overall support. Furthermore, I would like to thank them for providing me with patience and help for the parts of this thesis I struggled with. Additionally, I would like to thank Univ.Prof. Dr. Agata Ciabattoni, for informing me about events relevant to my research interests on multiple occasions and thereby allowing me to broaden my horizons. Most importantly, I am grateful for their patience and repeated elaborate feedback that allowed me to finish this thesis under their supervision.

Secondly, I would like to thank the WWTF project "Reasoning Tools for Deontic Logic and Applications to Indian Sacred Texts" for financial support during my work on this thesis.

Thirdly, I would like to thank Emery Neufeld for providing me with his framework for the normative supervisor and his immense patience with me, when I had troubles understanding parts of it. Without his help, I surely could not have finished the implementation of our reasoner in his framework.

Fourthly, I would like to thank my parents and grandparents for their love and support. In particular, I would like to thank my father, who proofread my work multiple times and pointed out grammatical errors in my work.

Lastly, thanks to my friends for all the help I have received from them. In particular, I would like to thank Nikita Kholodnyi, who helped me with valuable advice, when I had questions about Java. Also I want to thank Esra Ceylan and Fatih Bozdemir, who proofread my work and gave important feedback.

Kurzfassung

Durch die zunehmende Bedeutung von ethischer KI hat das Durchsetzen von ethischen, gesellschaftlichen und rechtlichen Normen gegenüber autonom agierenden Agenten ebenfalls an Bedeutung gewonnen. Über Normen zu argumentieren (normative reasoning) ist in vielen Fällen schwer, da sich Normen oft widersprechen, nur unter bestimmten Umständen gültig sind oder verletzt werden müssen. Eine Art über Normen zu argumentieren ist es, sie in logischer Form zu kodieren (Genaueres dazu auf Seite 63) und Prioritäten unter ihnen festzulegen. Über die Jahre wurden mehrere Ansätze vorgeschlagen. Diese Masterarbeit stellt eine einfache Methode zur Kodierung normativer Systeme unter Verwendung von "weak constraints" (schwache Einschränkungen) in Answer Set Programming vor. Answer Set Programming ist ein deklarativer Problemlösungsansatz, der seine Wurzeln in der Wissensrepräsentation, der logischen Programmierung und der nicht-monotonen Argumentation hat. Wir haben uns aufgrund der Ausdrucksstärke von Answer Set Programming und der Effizienz der verfügbaren Löser für diesen Ansatz entschieden. Standard Deontic Logic ist das erste logische System, das eingeführt wurde, um über Verpflichtungen, Erlaubnisse und verwandte Konzepte zu argumentieren. Wir beschreiben einige der bekanntesten deontischen Paradoxa, die zeigen, dass Standard Deontic Logic manche Aspekte, welche wir im Alltag verwenden um über Normen zu argumentieren, nicht erfassen kann. Wir kodieren diese Paradoxa fallweise unter Verwendung einer gemeinsamen Basis. Anfangs kodieren wir einige der deontischen Paradoxa. Wir abstrahieren und verallgemeinern diese Kodierungen um eine einfache Methodik zur Kodierung normativer Systeme zu entwickeln. Anhand von zwei Fallstudien wird diese Methodik demonstriert. In der ersten Fallstudie kodieren wir eine vereinfachte Version eines Beispiels aus der realen Welt, nämlich die Pflichten eines Agenten beim Autofahren. In der zweiten Fallstudie verwenden wir unsere Methodik, um die Normen für "ethische" Versionen des Pacman-Videospiels von Neufeld et al. (CADE, 2021) zu kodieren und zu implementieren.

Abstract

As ethical AI is becoming increasingly important so is the topic of enforcing ethical, legal and social norms on agents acting autonomously. Reasoning about norms (normative reasoning) is in many cases quite hard as often norms contradict each other, might only hold under certain circumstances or need to be violated. A way to reason about norms, is to encode them in some logic (discussed further on page 63) and determine prioritisation among these. Multiple approaches have been proposed over the years. This master thesis introduces a simple methodology to encode normative systems using weak constraints in Answer Set Programming. Answer Set Programming is a declarative problem solving approach, with roots in knowledge representation, logic programming and non-monotonic reasoning. We chose this approach because of the expressivity of Answer Set Programming and the efficiency of available solvers. Standard Deontic Logic is the first logical system introduced to reason about obligations, permissions and related concepts. We discuss some of the most famous deontic paradoxes which show that Standard Deontic Logic fails to capture aspects of real world reasoning about norms. We encode these paradoxes on a case by case basis using a common core. We start by encoding some of the deontic paradoxes. Through abstracting and generalising those encodings we develop a simple methodology for encoding normative systems. Our methodology is demonstrated on two case studies. In the first case study we encode a simplified version of a real world example, namely the obligations of an agent while driving. For the second case study, we use our methodology to encode and implement the norms for "ethical" versions of the video game Pacman by Neufeld et al. (CADE, 2021).

Contents

CHAPTER 1

Introduction

Norms are an integral part of human society. There are many different kinds of norms, such as social norms where violating them may lead to a loss of standing, embarrassment or even ostracization. Ethical norms can lead to emotional distress if ones own ethics are violated as well as loss of opportunities in the future. An example of this would be companies that practice unethical behaviour and lose customers as a result. Legal norms can, if violated, lead to sanctions, e.g., in the form of jail time enforced by the state [Bic06]. As norms have played a big part in most of human history, the logic behind these norms has fascinated many philosophers over time. In the 12th century Peter of Poitiers noted that, although if a sinner repents of sin he is guilty of sin, it does not follow that if a sinner wills to repent of sin he wills to be guilty of sin. A similar case can be seen in the *Good Samaritan Paradox*, where it is obligatory for one to help his neighbour who is in trouble although it is not obligatory for the neighbour to be in trouble. In the 1950s eontic logics gained traction through the introduction of Standard Deontic Logic by von Wright. Deontic logic is an area of logic that reasons obligation, permission and related concepts. See, e.g., Hilpinen and McNamara in [GHP+13] or [vBCF+22] for more about the history of deontic logics.

Through the advent of autonomous agents, their accountability and ethicality has become more important. This has also reinvigorated interest in the logic behind the norms these systems are supposed to act on. A functioning logic for legal norms could take away bias brought forth by the human part of the legal system. There are multiple works on these topics, e.g., works on deontic logic in law [JS92] and deontic logics as means of imposing norms on reinforcement learning agents [NBCG21].

Among the various applications of deontic logics this master thesis focuses on Artificial Intelligence (AI). More precisely, we are interested in encoding norms starting with deontic paradoxes. A methodology for encoding norms could be used to implement ethical behaviour for autonomous agents. However, currently deontic logics face some challenges in AI. First of all, there are no efficient provers available for deontic logics. Furthermore, reasoning about obligations and related concepts oftentimes requires defeasibility. A rule, obligation, prohibition,... is defeasible if there are circumstances under which it is annulled or in other words if given certain conditions it does not hold. When thinking about norms in the context of law the need for defeasibility becomes apparent. A lot of laws have exceptions given extraordinary circumstances. As an example, it is generally illegal to break windows of homes (unless they are your own). This prohibition is however lifted if, e.g., a person is hurt and the only way to give first aid to said person would be to break the window and go through. Deontic logics are mostly monotonic (meaning that conclusions cannot be retracted given new information) and static. Although there have been some defeasibility mechanisms introduced, such as in Defeasible Deontic Logic (DDL) [GORS13, GR08], there is no commonly accepted defeasible deontic logic.

We aim to address these problems by using Answer Set Programming (ASP) [MNT11] to reason about deontic concepts. This logic programming approach has become a popular topic in the logic programming and knowledge representation communities. Informally speaking, in Answer Set Programming a problem is modelled as a theory in a language of logic. Solutions for an instance of that problem correspond to models of the theory gained by encoding it. Through a long and systematic effort of the knowledge representation community effective tools were developed that are capable of processing programs in ASP fast [MNT11]. Lack of monotonicity is not a problem when using ASP, as default-negation and weak constraints allow us to add defeasibility to our encodings of norms.

In this master thesis we introduce a methodology for encoding normative systems in Answer Set Programming (ASP) using weak constraints. We first encode desired basic properties in a common core that will be used in all further encodings. By encoding multiple famous deontic paradoxes (e.g., *Chisholm's Paradox*, *Good Samaritan Paradox*,...) we show that ASP is capable of handling the deontic paradoxes in a satisfactory manner. By abstracting and generalising the encodings of the paradoxes we then present a simple methodology for encoding normative systems in ASP. This methodology is also tested in two case studies. In the first case study we encode a simplified real world example of a normative system and then test it. The second case study compares our methodology with the Defeasible Deontic Logic (DDL) approach taken by Neufeld et al. [NBCG21].

The thesis is organised as follows.

In Chapter 2 we introduce the preliminaries necessary for this work. We start by explaining the basics of propositional and first-order logic. This chapter also introduces the

syntax and semantics of Answer Set Programming in a general sense before introducing DLV (**D**ata**L**og with Disjunction, where **V** represents the symbol for logical disjunction), our ASP solver of choice. We introduce the syntax and semantics of DLV using examples and explain weak constraints which take a central role in our encodings. Furthermore, this section presents Standard Deontic Logic (SDL) and introduces the deontic paradoxes which will be our basis for developing a method of encoding normative systems. Deontic paradoxes showcase examples where SDL fails to capture aspects of common sense reasoning about norms. These paradoxes also show different kinds of obligations and their behaviour which will be integral for our methodology. This part also goes into detail on why SDL is unable to handle these paradoxes.

In Chapter 3 we introduce the vocabulary and common core used in all the encodings. Using this we encode the paradoxes presented in Chapter 2. This chapter also compares the approach of this master thesis with similar approaches to encode normative systems seen in the multi-agent system community, and the logic programming community.

In Chapter 4 we abstract and generalise the encodings presented in the previous chapter. We start by grouping the paradoxes as in Hilpinen and McNamara [GHP+13] and show the similarities in the encodings of the paradoxes that are grouped together. Using these we list different kinds of obligations seen in the paradoxes and explain them. For each kind of obligations a way of encoding is given as well. The presented encodings are consistent with the way the obligations were formalised in the encodings of the paradoxes. Using the encodings for the different types of obligation we propose a general method of encoding normative systems. The chapter concludes with a case study that considers a normative system containing multiple different kinds of obligations, which presents a simplified version of the obligations of an agent driving a car. We encode this normative system using our proposed methodology. We then test the correctness of our encoding in DLV using three different situations and checking whether the correct obligations are derived.

In Chapter 5 we test the viability of encoding normative systems in practice using the game Pacman as a simple toy example. Neufeld et al. [NBCG21] used a theorem prover for defeasible deontic logic in order to impose "ethical" constraints on a reinforcement learning agent. The goal of this work was to let the agent keep the optimal play pattern it has learned through training while still enforcing norms on the agent. Using the framework and reinforcement training algorithm in [NBCG21], we train a reinforcement learning agent and then impose two sets of "ethical" constraints on the agent using our methodology for encoding normative systems. We compare the results of our encoding with the results presented by Neufeld et al.

Chapter 6 serves as the conclusion for our work. We talk about the results of our case studies and discuss the advantages and disadvantages of our methodology. Possible future work will be addressed as well.

CHAPTER 2

Preliminaries

In this chapter, we introduce the basic notions used in this work.

2.1 Propositional and First-Order Logic

Propositional logic and First-order logic are the basis of Standard Deontic Logic resp. Answer Set Programming.
Here we follow [BKI19, chapter 3] and [Rau10, Bar77] to recall some of the basic terminology used in this work.

2.1.1 Propositional Logic

Definition 1. (Propositional signature)
A propositional signature Σ is a set of symbols referred to as propositional variables.

These propositional variables are used to build the set of propositional formulas $Formula(\Sigma)$.

Definition 2. (Set of propositional formulas)
$Formula(\Sigma)$ is constructed in the following way:

1. All atomic formulas are contained in $Formula(\Sigma)$. Atomic formulas are formulas which only consist of a propositional variable.
2. If A and B are propositional formulas, then the following formulas are also propositional formulas:
   - $(\neg A)$ which is read as "not A"
   - $(A \wedge B)$ which is read as "A and B"

- $(A \vee B)$ which is read as "A or B"
- $(A \rightarrow B)$ which is read as "if A, then B"
- $(A \leftrightarrow B)$ which is read as "A if and only if B"

The operators $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ are referred to as logical connectives. In order to simplify notation and omit parentheses, the following binding priorities are commonly used:

$$\neg > \wedge > \vee > \rightarrow > \leftrightarrow$$

For example $\neg$ binds stronger than $\wedge$, $\wedge$ binds stronger than $\vee$ and so on, see e.g., [BKI19, chapter 3]. In order to evaluate propositional formulas, a so called interpretation is used:

Definition 3. (Interpretation function)
Let Σ be a propositional signature. Then a function $I : \Sigma \rightarrow \{0, 1\}$ is called an interpretation.

A propositional variable is interpreted as *True* if it is mapped to 1, else it is interpreted as *False*.

Definition 4. (Truth value of a formula)
Let I be an interpretation for Σ. For a propositional formula A, its truth value $[\![A]\!]_I$ for the interpretation I is given by the function:

$$[\![.]\!]_I : Formula(\Sigma) \rightarrow \{0, 1\}$$

for which the following conditions hold:

- If A is an atomic formula, then $[\![A]\!]_I = I(A)$
- If A is of the form $\neg B$, then $[\![A]\!]_I = 1$ if $[\![B]\!]_I = 0$ and vice versa.
- If A is of the form $B \wedge C$, then $[\![A]\!]_I = 1$ if $[\![B]\!]_I = [\![C]\!]_I = 1$. Else $[\![A]\!]_I = 0$.
- If A is of the form $B \vee C$, then $[\![A]\!]_I = 0$ if $[\![B]\!]_I = [\![C]\!]_I = 0$. Else $[\![A]\!]_I = 1$.
- If A is of the form $B \rightarrow C$, then $[\![A]\!]_I = 0$ if $[\![B]\!]_I = 1$ and $[\![C]\!]_I = 0$. Else $[\![A]\!]_I = 1$.
- If A is of the form $B \leftrightarrow C$, then $[\![A]\!]_I = 1$ if $[\![B]\!]_I = [\![C]\!]_I$. Else $[\![A]\!]_I = 0$.

A commonly used rule of inference is the so-called modus ponens.

Definition 5. (Modus Ponens)
Given two formulas of the form:

$$A \to B$$

and

$$A$$

where each formula is assigned the truth value true, one can derived that the truth value of B is true as well. In other words, if A implies B and A holds, then B must hold as well.

Definition 6. (Satisfiability, validity)
A formula A is satisfied by an interpretation I, if $[\![A]\!]_I = 1$.
A formula A is satisfiable, if there exists an interpretation I, such that $[\![A]\!]_I = 1$.
A formula A is valid, if for all interpretations I, it holds that $[\![A]\!]_I = 1$.
A formula A is unsatisfiable, if for all interpretations I, it holds that $[\![A]\!]_I = 0$.

Many propositional formulas behave in the same way. Formulas such as $A \wedge A$ and A are satisfied by the same interpretations. Formulas that have the same truth value are referred to as semantically equivalent. Examples of semantically equivalent formulas can be seen in [BKI19, chapter 3].

2.1.2 First-Order Logic

First-order Logic (FOL) is a lot more expressive than Propositional Logic. By using functions and predicates one can describe objects, their relations, attributes, and functions on these objects.

This added expressive power can be seen in the added complexity of the signature:

Definition 7. (Signature)
A (FOL-)signature $\Sigma = (Func, Pred)$ consists of a countable set $Func$ of function-symbols and a countable set $Pred$ of predicate symbols. Each symbol $s \in Func \cup Pred$ has a fixed arity≥ 0. A function symbol of arity 0 is referred to as a constant.

A Σ-interpretation assigns meanings over a non-empty set of elements (called the universe U) to the elements in Σ.

The universe U can be any non-empty set.

Definition 8. (Interpretation)
Let Σ be a signature. A Σ-interpretation $I = (U_I, Func_I, Pred_I)$ consists of:

- A non-empty set U_I called the universe.
- A set $Func_I = \{f_I : \underbrace{U_I \times \ldots \times U_I}_{n \text{ times}} \to U_I \mid f \in Func \text{ of arity } n\}$ of functions.
- A set $Pred_I = \{p_I \subseteq \underbrace{U_I \times \ldots \times U_I}_{n \text{ times}} \mid p \in Pred \text{ of arity } n\}$ of relations.

The Σ-interpretation assigns meaning to function- and predicate-symbols in the following way:

- 0-ary functions are interpreted through objects in the universe U. Note that, although every constant must be mapped to an object in U, multiple constants may be mapped to the same object.
- Other functions are interpreted through functions, mapping objects in the universe to objects in the universe.
- 0-ary predicates are treated like propositional variables. Therefore, the interpretation assigns a truth value to those predicates.
- Unary predicates are interpreted as subsets of the universe, e.g., the predicate *yellow* could be interpreted as the subset of objects in U that are yellow.
- Predicates of higher arity are interpreted as relations of that arity over the universe U. As an example, the binary relation *father* could denote the relation of one individual to its father.

Unlike propositional logic FOL allows terms. Terms are functional expressions that are built using function-symbols in the signature and are interpreted by elements in the universe.

Definition 9. (Term)
The set $Term_\Sigma(V)$ of terms over a signature Σ and a set V of variables is the smallest set containing the following elements:

1. x for every $x \in V$
2. c for every $c \in Func$ with arity 0
3. $f(t_1, \ldots, t_n)$ for $f \in Func$ with arity $n > 0$ and $t_1, \ldots, t_n \in Term_\Sigma(V)$

Some examples for terms are:

$$x_1, x_2, c, f_2(c_2, f_1(x_1, c_1), x_2)$$

Terms can then be evaluated using a variable assignment (given an interpretation).

Definition 10. (Variable assignment)
Given a Σ-interpretation I and a set V of variables, a variable assignment is a function $\alpha : V \to U_I$, such that each variable is assigned to an element from the universe. Note that multiple variables may be assigned to the same element and there may be elements that no variable is assigned to.

Definition 11. (Evaluation of a term)
Given a term $t \in Term_\Sigma(V)$, a Σ-interpretation I and a variable assignment V the term evaluation of t in I under α, written as $[\![t]\!]_{I,\alpha}$ is given through a function:

$$[\![.]\!]_{I,\alpha} : Term_\Sigma(V) \to U_I$$

and is defined in the following way:

1. $[\![x]\!]_{I,\alpha} = \alpha(x)$, where x is a variable.
2. $[\![f(t_1, \ldots, t_n)]\!]_{I,\alpha} = f_I([\![t_1]\!]_{I,\alpha}, \ldots, [\![t_n]\!]_{I,\alpha})$, where f is an n-ary $(n \geq 0)$ function symbol.

Using terms atomic formulas can now be defined as follows.

Definition 12. (Atomic formula)
An atomic formula (or atom) over a signature Σ and a set of variables V is constructed in the following way:

1. p if $p \in Pred$ and p has arity 0
2. $p(t_1, \ldots, t_n)$ if $p \in Pred$ and p has arity $n > 0$ and $t_1, \ldots, t_n \in Term_\Sigma(V)$

Some examples for atomic formulas are:

$$p_2, p_1(c, x_1, f(x_3, x_4), f_2(c_2, f_1(x_1, c_1), x_2)), p_3(c), p_4(x_1, x_2, f(x_3)).$$

Definition 13. (Truth value of an atomic formula)
The truth value of an atomic formula $p(t_1, \ldots, t_n)$ given a variable assignment α is true (or 1) if and only if the evaluation of the terms $t_1, \ldots t_n$ given the interpretation I under α is contained in the relation that gets assigned to p under I. Otherwise the truth value is false (or 0). In other words:

$$[\![p(t_1, \ldots, t_n)]\!]_{I,\alpha} = 1 \text{ if } ([\![t_1]\!]_{I,\alpha}, \ldots, [\![t_n]\!]_{I,\alpha}) \in p_I$$
$$[\![p(t_1, \ldots, t_n)]\!]_{I,\alpha} = 0 \text{ else}$$

Example 14. Consider the following example:

- The signature Σ contains two constants *Max* and *Frank*, and the predicate *Grandfather* of arity 2.

- The universe $U = \{me, my\_grandfather\}$
- $I(Max) = me$; $I(Frank) = my\_grandfather$;
  $I(Grandfather) = \{(me, my\_grandfather)\}$

Then, the formula $Grandfather(me, my\_grandfather)$ would be evaluated to true.

Finally the set of formulas and their truth value can be defined.

Definition 15. (Formulas)
The set $Formula_\Sigma(V)$ is the set of formulas generated by the signature Σ and a set of variables V. This set is the smallest set containing the following elements:

1. P, if P is an atomic formula over Σ and V
2. $(\neg F)$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \Rightarrow F_2)$, $(F_1 \Leftrightarrow F_2)$
3. $(\exists x F)$, $(\forall x F)$

where $x \in V$ and $F, F_1, F_2 \in Formula_\Sigma(V)$.

Note that atoms and negated atoms are referred to as literals. Let A be an atom, then A is a positive literal and $\neg A$ is a negative literal. A literal, atom or formula is ground if it contains no variables. We will use c to refer to constants and x to refer to variables.

Example 16.
- literals: $p(x_1, f(x_2, x_1)), \neg p_3(c), p_2(f(x), c_2)$
- ground literals: $p_3(c), \neg p(c_1, f(c_2, c_3))$

Definition 17. (Clause)
A disjunction of literals is referred to as a clause.

Example 18. An example of a clause would be:

$$\neg p_3(c) \vee p(x_1, f(x_2, x_1)) \vee p(c_1, f(c_2, c_3)),$$

The truth values for formulas containing logical connectives are handled in the same way as in propositional logic. Now, we define the truth values for formulas containing quantifiers $(\forall, \exists)$.

Definition 19. (Truth value of a quantified formula)
Given a quantified formula $F \in Formula_\Sigma(V)$, a Σ-interpretation I and a variable assignment α, the truth value of F (written as $[\![F]\!]_{I,\alpha}$) is defined by:

$$[\![\forall x G]\!]_{I,\alpha} = 1 \text{ if and only if for all } a \in U_I \ [\![G]\!]_{I,\alpha_{x/a}} = 1 \text{ holds}$$

$$[\![\exists x G]\!]_{I,\alpha} = 1 \text{ if and only if for some } a \in U_I \ [\![G]\!]_{I,\alpha_{x/a}} = 1 \text{ holds}$$

where $\alpha_{x/a} : V \to U_I$ is the variable assignment which maps x to a and is otherwise identical to α.

Note that if no variable assignment is given, a formula is interpreted as true if and only if it is true under all variable assignments.

2.2 Answer Set Programming

Since its inception logic programming has found multiple practical uses such as in Expert Systems (a system designed with experts on a certain field that aims to reproduce their decision-making abilities) [BKI19, chapter 1]. This section will introduce the necessary background on Answer Set Programming needed in this master thesis. The definitions follow [BKI19, chapter 9] where further information can be found as well.

A logic program is a set of rules.

Classical logic programming uses a rather simple syntax, which only allows Horn clauses as rules. Horn clauses are clauses that contain at most one unnegated literal and can be represented as rules, where the head of the rule consists of at most one atom and the body of the rule contains only atoms. This leads to less expressiveness. The Horn clause

$$\{H, \neg B_1, \ldots, \neg B_n\}$$

represents the following rule, which is read as "H holds if B_1 and... and B_n hold".

$$H \leftarrow B_1, \ldots, B_n$$

Extended logic programs do not have the syntactic restriction of only using Horn clauses. Furthermore, strong negation as well as default-negation can be used in rules. Default-negation, also referred to as negation as failure, allows the representation of non-monotonic assumptions. The distinction between not-knowing (denoted as default-negation) and definite falsity (denoted as strong negation) allows for a realistic processing of knowledge, which is more akin to that of human reasoning.

A rule in an extended logic program takes the following form:

$$H \leftarrow A_1, \ldots A_n, \ \mathit{not}\ B_1, \ldots, \ \mathit{not}\ B_m, \tag{2.1}$$

where $A_1, \ldots, A_n, B_1, \ldots, B_m$ are (negated) literals.
Informally, it can be read as: "If $A_1, \ldots, A_n$ hold and none of $B_1, \ldots, B_m$ are found to be true, deduce H".

However, this gain in expressiveness comes at the cost of a more complicated semantics. In 1998, Gelfond and Lifschitz introduced stable-model semantics, which has turned out to be one of the most successful approaches to setting a semantics for extended logic programs. Some of the extended logic programming approaches allow the use of disjunction in the head of rules as well. Such a rule can look like this:

$$H_1 \vee H_2 \vee \dots H_l \leftarrow A_1, \dots A_n,\ not\ B_1, \dots,\ not\ B_m.$$

Informally, one can say that if the body of the rule holds, at least one of the literals in the head of the rule must hold as well.
In this thesis we consider extended logic programs with disjunctions.

2.2.1 Semantics

In order to accurately describe the answer set semantics, we start with some basic definitions.

Definition 20. (Herbrand universe, Herbrand base, Herbrand interpretation)
Let $\mathcal{P}$ be a logic program and the signature Σ be the set of all function- and predicate-symbols appearing in Σ. Then, the Herbrand universe is the set of all ground terms over Σ and the Herbrand base $\mathcal{H}(\mathcal{P})$ is the set of all ground atoms over Σ. Every $M \subseteq \mathcal{H}(\mathcal{P})$ is a Herbrand interpretation.

Example 21. Consider the following signature $\Sigma = (Func = \{c/0, f_1/1, f_2/1\}, Pred = \{p/1\})$. Here the number after the slash specifies the arity of the function resp. predicate. Then, the Herbrand universe takes the following form:

$$\{c, f_1(c), f_2(c), f_1(f_1(c)), \dots\}$$

The Herbrand base then look like this:

$$\mathcal{H}(\mathcal{P}) = \{p(c), p(f_1(c)), p(f_2(c)), p(f_1(f_1(c))), \dots\}$$

We continue with defining the operator $T_{\mathcal{P}}$.

Definition 22. ($T_{\mathcal{P}}$)
Let $M \subseteq \mathcal{H}(\mathcal{P})$. The monotone operator $T_{\mathcal{P}} : 2^{\mathcal{H}}(\mathcal{P}) \rightarrow 2^{\mathcal{H}}(\mathcal{P})$ is defined in the following way:

$A \in T_{\mathcal{P}}(M)$ if, there exists a clause $H \leftarrow B_1, \dots, B_n$ in $\mathcal{P}$ and a ground substitution σ, such that $A = \sigma(H)$ and $\{\sigma(B_1), \dots, \sigma(B_n)\} \subseteq M$

A ground substitution is a substitution that maps all variables to ground terms.

Using this operator we can define the meaning of $M \models_\Sigma \mathcal{P}$.

$$M \models_\Sigma \mathcal{P} \text{ if } T_\mathcal{P}(M) \subseteq M$$

In other words, M may not contain the body of a rule if it does not contain the head.

Definition 23. (Herbrand model)
$M \subseteq \mathcal{H}(\mathcal{P})$ is a Herbrand model if $M \models_\Sigma \mathcal{P}$.

Note that every Herbrand model is a Herbrand interpretation. Informally one could say, that a Herbrand interpretation denotes the atoms, which are true in that interpretation and a Herbrand interpretation is a Herbrand model if it is compatible with the logic program $\mathcal{P}$.

Example 24. Consider the following small logic program:

$$\boxed{grandfather(X, Z) \leftarrow father(X, Y), father(Y, Z).}$$

$$M_1 = \{father(Max, James), father(James, Frank)\}$$

The Herbrand interpretation M_1 is not a Herbrand Model, as
$grandfather(Max, Frank) \notin M_1$ but $grandfather(Max, Frank) \in T_\mathcal{P}(M_1)$.

$M_2 = \{father(Max, James)\}$
$M_3 = \{father(Max, James), father(James, Frank), grandfather(Max, Frank)\}$

M_2 and M_3 on the other hand are Herbrand models, as $T_\mathcal{P}(M_2) \subseteq M_2$ resp. $T_\mathcal{P}(M_3) \subseteq M_3$.

Since we are looking at extended logic programs, it is not enough to only consider ground atoms, as we differentiate between definite falsity and absence of knowledge. As such we also need to consider negated ground literals. Rather than looking at Herbrand models we now consider states.

The literals $P(t_1, \ldots, t_n)$ and $\neg P(t_1, \ldots, t_n)$ are referred to as complementary. A set S of ground literals is consistent, if it contains no complementary literals. A consistent set of ground literals is referred to as a state.

We now introduce the fundamental concept of a reduct.

Definition 25. (Reduct of an extended logic program $\mathcal{P}$)
Let $\mathcal{P}$ be an extended logic program comprised of rules of the form (2.1). Then the reduct $\mathcal{P}^S$ for a state S is defined as:

$$\mathcal{P}^S := \{H \leftarrow A_1, \ldots, A_n \mid H \leftarrow A_1, \ldots A_n,\ not\ B_1, \ldots,\ not\ B_m \in \mathcal{P}, \{B_1, \ldots, B_m\} \cap S = \emptyset\}$$

Example 26. Consider the following logic program:

$$\boxed{\begin{array}{l} P(a) \leftarrow not\ \ Q(a). \\ Q(a) \leftarrow not\ \ P(a). \end{array}}$$

Then, for $S_1 = \{P(a)\}$ and $S_2 = \{Q(a)\}$ we have $\mathcal{P}^{S_1} = \{P(a).\}$ and $\mathcal{P}^{S_2} = \{Q(a).\}$.

We now define what it means for a state S to be closed under a logic program $\mathcal{P}$.

Definition 27. (Closed state)
Let $\mathcal{P}$ be an extended logic program without default negation and let S be a state. S is closed under $\mathcal{P}$ if for every rule r in $\mathcal{P}$ the following holds:

$$\text{If } pos(r) \subseteq S, \text{ then } head(r) \cap S \neq \emptyset,$$

where $pos(r)$ are the non-default negated literals in the body of the rule and $head(r)$ is the head of the rule.

Using this we can define answer sets for extended logic programs. We start by defining answer sets for extended logic programs without default-negation.

Definition 28. (Answer sets for logic programs without default negation)
Let $\mathcal{P}$ be an extended logic programs without default-negation and let S be a state. Then, S is an answer set of $\mathcal{P}$ if it is minimal (under set inclusion) and closed under $\mathcal{P}$.

Using this we can define answer sets for all extended logic programs, as the reduct $\mathcal{P}^S$ does not contain default negation.

Definition 29. (Answer sets)
Let $\mathcal{P}$ be an extended logic program and let S be a state. Then, S is called answer set of $\mathcal{P}$, if S is an answer set of the reduct $\mathcal{P}^S$.

Note that, due to the condition $head(r) \cap S \neq \emptyset$, this semantics works for disjunctive extended logic programs as well.

Finally we define safety of a rule r, which is required by most compilers of logic programs:

Definition 30. (Safe rule)
A rule r is safe, if each variable that appears in the head of the rule appears non-default negated in the body of the rule.

2.2.2 DLV

DLV (**D**ata**L**og with Disjunction, where **V** represents the symbol for logical disjunction) is a deductive database system based on disjunctive logic programming. It was developed by a research team from the University of Calabria and the Vienna University of Technology [BFI+20]. All information in this subsection can be found in [BFI+20]

DLV's language is an extension of Disjunctive datalog, which contains constraints, true negation and queries.

Syntax
Constants in DLV must either start with a lowercase letter and may contain letters, underscores and digits or be a number (e.g., a, 2, alphaBeta23, d_).

Variables must always start with a capital letter and can contain letters, underscores and digits (e.g., A, Person12, Car_1).

There is also the concept of an anonymous variable, which is denoted by an underscore. It represents a variable which does not appear in another part of the rule. It can be understood as a variable which can be ignored in the current rule.

More complex terms can be built using function symbols or lists.

Functional symbols must begin with a lowercase letter and can only be comprised of letters, underscores and digits. Some examples for functional terms are:

$$f(x); result(a, Person, f(1)); sUm\_of(1, 2)$$

List terms can either be represented through a list of terms in square brackets (e.g., $[a, b, c, \ldots]$) or by representing it as a head of a list h and a tail of a list t (which is a list term) (e.g., $[h \mid t]$).

Predicate symbols must begin with a letter (either upper or lower case) and are comprised of letters, underscores and digits.

Note that *not* is not a valid predicate symbol or constant, as it is reserved for representing default negation.

Atoms are predicate symbols, with some or possibly no terms, representing the tuples in the relation defined by the predicate. Some examples for atoms can be seen below:

$$danger; friendly(Person1, albert); Equal(Value1, 3)$$

Literals are atoms, which may be strongly or weakly negated. Strong negation is represented by $-$ or $\sim$ (although in our encodings only $-$ will be used), while default negation is represented by *not*. Note that while default negation may precede strong negation the inverse is not valid syntax.

Rules are of the form

$$H_1 \vee \ldots \vee H_m :- A_1, \ldots, A_n.$$

Where $A_1, \ldots, A_n$ are literals in the body of the rule and $H_1, \ldots H_m$ are atoms or strongly negated atoms, which form the head of the rule.

Note that disjunction may appear in the head of the rule.

Facts are rules that have an empty body. Disjunction may also appear in facts. Below are some examples for facts:

$$edible(plant) \vee unedible(plant).$$
$$edible(apple).$$

Constraints on the other hand are rules that have an empty head. In other words, these rules do not allow for the body of the rule to be satisfied. This can be used to filter out unwanted answer sets. Below is an example for constraints:

$$:- edible(X), unedible(X).$$

Note that each rule (and therefore constraint and fact as well) must be ended by a period.

In order to guarantee that a rule is logically equivalent to the set of its Herbrand instances, DLV imposes a safety condition on variables in rules.

A variable X is safe if at least one of the following conditions holds:

- X occurs in a non-default negated predicate in the body of the rule;
- X occurs in a non-default negated strongly negated predicate in the body of the rule.

A rule is safe if all variables that appear in said rule are safe.

Below are some examples of unsafe rules:

$$c(X) \vee -c(X).$$
$$c(X) :- not\ \ a(X).$$
$$c(X) :- person(Y), relation(Y, Z).$$

Example 31. A small program which recognizes danger when driving a vehicle could look like this:

$$\boxed{\begin{array}{l} Danger(X) :- overheat(X), fast(X). \\ Danger(X) :- overheat(X), heavy(X). \end{array}}$$

Additional facts could be entered by a separate file which contains the speed of the car and whether different tools overheat. The contents of this file could look like this:

$$\boxed{fast(car). \quad heavy(car).}$$

This file could be generated by a program, that checks whether the car goes over a certain speed, carries a heavy load, or is overheated.

Weak constraints
Weak constraints are another construct that can be used in DLV. These are constraints that are only satisfied when possible. In the DLV syntax they are specified in the following way:

$$:\sim A_1, \ldots, A_n.[x : y]$$

where $A_1, \ldots, A_n$ are literals and x and y are a weight resp. level. The weight or level may be omitted. In that case the weight resp. level is assumed to be the same for all constraints.

Intuitively, answer sets for logic programs containing weak constraints are found by taking all answer sets that have minimal weight for violated weak constraints at the highest level. Afterwards among these answer sets those with minimal weight for violated weak constraints at the second highest level are taken. This process is repeated until the lowest level is reached. The remaining answer sets are then the answer sets considering the weak constraints.

Definition 32. (Answer sets for logic programs with weak constraints) Let $\mathcal{P}$ be a logic program containing weak constraints and let $\mathcal{P}'$ be the logic program that is generated by removing all weak constraints from $\mathcal{P}$.

Every answer set of $\mathcal{P}$ must also be an answer set of $\mathcal{P}'$.

An answer set A of the logic program $\mathcal{P}'$ is an answer set for $\mathcal{P}$ if and only if for any other answer set A' of $\mathcal{P}'$ the weight of the violated weak constraints is either the same at each level or if for the highest level where the weights of the violated weak constraints differ, the weight of the violated weak constraints is less for A'.

2.3 Standard Deontic Logic and Its Paradoxes

The logic of norms (that deal with obligations, permissions and related concepts) has been an area of interest for many centuries. "Deontology", which is derived from the greek word "déon" (obligation/necessity), was used to describe the "science of morality" by Jeremy Bentham. In the 1920s, Austrian philosopher Ernst Mally proposed "Deontik", a system of the "fundamental principles of the logic of ought", which already dealt with the common notions of deontic logic. In the early 1950s von Wright introduced Standard Deontic Logic, SDL for short. Since then multiple different deontic logics motivated by philosophical considerations as well as by applications in various fields have been proposed. A detailed overview of the research in deontic logic and related topics can be found in [GHP+13, GHP+21]. Most of these deontic logics face similar difficulties, as they end up not being able to capture certain aspects of common sense reasoning that one would want these logics to capture.

This stems from these logics either being too weak or too strong. In other words, they might not derive statements which are intuitive or derive statements which are counter-intuitive using common-sense. This is due to the complexity of the considered subject. Deontic logics may not capture the details needed to reason about norms, as there are a lot of hidden assumptions, which can also seen in the encodings shown in the later chapters. So called "deontic paradoxes" are a common way of showcasing examples for which these deontic logics fail.

This section explores Standard Deontic Logic, some of the most famous paradoxes and their classification following [JC02].

Standard Deontic Logic, as introduced by von Wright, builds upon classical propositional logic and is part of the class of normal modal logics. It is a so-called monadic deontic logic, as the operators O (obligation) and P (permission) are one-place operators, meaning they apply only to a single formula. The following subsections will follow [JC02] and the work of Hilpinen and McNamara in [GHP+13].

2.3.1 Syntax of SDL

Let $\mathbb{AT}$ be a set of atomic propositions. The following Backus Normal form generates the language of monadic deontic logic (and therefore also SDL):

$$\varphi := p \in \mathbb{AT} \mid \neg\varphi \mid (\varphi) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \rightarrow \varphi \mid O\varphi \mid P\varphi \mid F\varphi$$

As one can see every propositional formula is a part of the language of monadic deontic logic, as well. $O\varphi$ is read as "it is obligatory that φ", $P\varphi$ is read as "it is permissible that φ", and $F\varphi$ is read as "it is forbidden that φ" [PT18].

Axioms of SDL:
The axiomatization of SDL is obtained by adding the following axioms and rules to any axiomatization of classical propositional logic:

$$\text{If } \varphi \text{ is a theorem, } O\varphi \text{ is a theorem} \tag{RND}$$
$$O(\varphi \rightarrow \psi) \rightarrow (O\varphi \rightarrow O\psi) \tag{KD}$$
$$O\varphi \rightarrow \neg O\neg\varphi \tag{DD}$$

Note that in the above axioms and rules φ and ψ are schemes representing arbitrary formula.

(RND) is referred to as the deontic necessitation rule or normality axiom and is added because of theoretical simplicity as well as continuity with classical propositional logic and modal logic. In a more general sense (RND) guarantees that for any theorem, the claim that this theorem is obligatory is also a theorem. This can be understood as a rule ensuring that any absolute truth is also obligatory. This leads to there always being obligations; a fact that leads to certain problems as can be seen later on.

(KD) is an analogue to the rule K, which is found in all normal modal logics. It states that if a condition and its antecedent are obligatory, so is its consequent. While this seems like a sensible axiom to include, when thinking about reasoning over obligations, it can be seen that this axiom also leads to some problems.

(DD) states that if p is obligatory, then its negation $\neg p$ is not obligatory. This prevents conflicting obligations from occurring, as obviously it is not possible to fulfill both p and $\neg p$.

Due to the axioms (KD) and (DD), SDL is sometimes referred to as the system KD or system D.

Defining Operators and Useful Theorems

The so called "Traditional Definitional Scheme" for the operators is usually presupposed in formulations of SDL:

$$Pp =_{df} \neg O \neg p$$
$$Fp =_{df} O \neg p$$

The first equivalence simply states that something being permissible is equivalent to its negation not being obligatory, e.g., it is permissible to stay home iff it is not obligatory to not stay home (go out).
The second equivalence states that an action being forbidden is equivalent to its negation being obligatory, e.g., being forbidden from parking in a spot is equivalent to being obliged not to park in that spot [PT18].

Using the axiomatization of SDL, the following useful theorems can be derived:

$O(p \wedge q) \rightarrow (Op \wedge Oq)$	(Conjunctive Distributivity of O)
$(Op \wedge Oq) \rightarrow O(p \wedge q)$	(Aggregation for O)
$O\top$	(ON)
$\neg O\bot$	(OD)
If $p \rightarrow q$ is a theorem, then $Op \rightarrow Oq$ is a theorem	(RMD)

Note that by combining the theorems (Conjunctive Distributivity of O) and (Aggregation for O) one can derive the following equality:

$$O(p \wedge q) \leftrightarrow (Op \wedge Oq)$$

This shows that SDL is incapable of differentiating between an obligation over a conjunction of actions and the conjunction of obligations. As seen later, this lack of expressiveness will lead to problems.

(Conjunctive Distributivity of O) is a rather intuitive theorem. It represents the notion that, should a conjunction of two actions be obligatory, then so are the two actions themselves. While this works in a lot of cases, such as deriving the obligation to pay taxes and the obligation to work from the obligation to pay taxes and work, it does not always make sense, such as when two obligations depend on each other to make sense.

(Aggregation for O) simply formalises the fact that should two actions be obligatory, then so is the aggregation of those two actions. For example, if it is obligatory to pay rent and it is obligatory to keep the apartment clean, then it is obligatory to pay rent and keep the apartment clean. As a consequence of this theorem, SDL is unable to handle conflicting obligations. Consider the following example, also known as *Sartre's Dilemma* [JC02]:

It is obligatory that x
It is obligatory that not x,

which can be formalised in SDL in the following way:

$$Ox$$
$$O(\neg x)$$

Using (Aggregation for O) this instantly leads to the obligation $O(x \wedge \neg x)$, from which any obligation can be derived using the axioms (KD) and (RND) by the following steps:

$(x \wedge \neg x) \rightarrow \bot$	(Theorem)
$O((x \wedge \neg x) \rightarrow \bot)$	(Application of (RND))
$O(x \wedge \neg x) \rightarrow O(\bot)$	(Application of (KD))
$O(\bot)$	(Application of modus ponens)
$\bot \rightarrow \varphi$	(Theorem)
$O(\bot \rightarrow \varphi)$	(Application of (RND))
$O(\bot) \rightarrow O(\varphi)$	(Application of (KD))
$O(\varphi)$	(Application of modus ponens)

Since φ can be substituted by any formula, one can see that any obligation is derivable. This problem is often referred to as "Deontic Explosion", as the amount of deontic obligations becomes unbounded [PT18].

The theorems (ON) and (OD) state that $\top$ is always obligatory and $\bot$ is never obligatory. (ON) is directly derived from (RND), since $\top$ is a theorem. Using (ON), (OD) can be derived using (DD) [PT18]:

$$O(\top) \rightarrow \neg O(\neg \top)$$

(RMD) intuitively states that should an action p imply an action q, then if p is obligatory so is q. As an example, consider the obligation to eat the whole meal. Since eating the entire meal would imply eating the soup as well as the main course, the obligation to eat the soup as well as the obligation to eat the main course could be derived.

2.3.2 Semantics of SDL

This subsection will take a look at two different semantics for SDL.

"Standard Semantics"
The semantics of SDL uses the well known Kripke Semantics of modal logic. As in Kripke Semantics sentences can be interpreted with regard to possible worlds.

For this purpose $M = \langle W, R, I \rangle$, a possible worlds model of SDL is used. Such a model consists of:

- A universe of possible situations/worlds W, which would correspond to the nodes in a Kripke frame.
- A binary relation R on W, which is understood as a relation of deontic alternativeness, i.e., sRt denotes that t is an "ideal" successor to s, as it complies with the obligations which are active at s.
- An interpretation function I, which assigns to each propositional atom p the largest subset $W' \subseteq W$ such that p is deemed true at all $u \in W'$.

The truth of a formula p under M at a possible situation $u \in W$ is then written as $M, u \models p$ (the situation u in the model M satisfies p) or $u \models p$ (the situation u satisfies p), when the model M is known. $M, u \models p$ resp. $u \models p$ is defined recursively, as in Kripke Semantics:

- If p is a propositional atom, then $M, u \models p$ holds if $u \in I(p)$.
- If p is of the form $p_1 \wedge p_2$, then $M, u \models p$ holds if $M, u \models p_1$ and $M, u \models p_2$.
- If p is of the form $p_1 \vee p_2$, then $M, u \models p$ holds if $M, u \models p_1$ or $M, u \models p_2$.
- If p is of the form $p_1 \rightarrow p_2$, then $M, u \models p$ holds if $M, u \not\models p_1$ or $M, u \models p_2$.

The truth conditions of the deontic operators O and P are formulated analogously to the truth conditions of the modal operators $\Box$ and $\Diamond$:

- $u \models Op$ holds if $v \models p$ holds for all v such that uRv.
- $u \models Pp$ holds if $v \models p$ holds for some v such that uRv.

M must fulfill seriality, as else the axiom (DD) would be violated.

$$\text{For every } u \in W, \;\; uRv \text{ for some } v \in W.$$

Given additional assumptions about the structure of the relation R, it is possible to have further requirements. These requirements can lead to different systems of deontic logic with different properties.

"Utilitarian Semantics"
A different approach to SDL semantics can be taken as well. Once again consider the possible worlds model from above, except that R denotes currently reachable situations,

rather than "ideal" situations.

For each of the possible situations $u \in W$ there exists a relation ranking the reachable situations. The set of these relations is called R. Let $\geq_u$ denote the relation for $u \in W$, where $v \geq_u w$ indicates that v is at least as "good" as w for u. We require these relations to be reflexive, transitive and connective:

- $v \geq_u v$ must hold. (reflexivity)
- If $v \geq_u w$ and $w \geq_u x$ then $v \geq_u x$ must hold as well. (transitivity)
- Either $v \geq_u w$ or $w \geq_u v$ must hold. (connectivity)

In addition, the so called "Limit assumption" is added for every u:

$$\forall u \exists v \forall w : v \geq_u w \tag{LA}$$

This assumption ensures that there is always at least one world v relative to u, s.t. v is at least as good as any other world relative to u. Intuitively, it can be thought of as ensuring the existence of an u-best world. In this framework the truth of Op in a situation $u \in W$ can be defined as follows:

Op is true at u if p holds in all the u-best worlds

The limit assumption, is however a controversial assumption to make, as in a model without any best worlds, there would be no obligations and everything would be permissible. To give an instance where this would be nonsensical, consider the deontic dial. The deontic dial is a dial, which can be turned to any real number r, with $0 \leq r \leq 1$. Turning the dial to 0 or 1 leads to disaster, while for $0 < r < 1$, increasing values lead to increasingly positive results. E.g. turning the dial to $r = 0.7$ is preferable to turning it to $r = 0.5$.

In this model, there obviously doesn't exist a best world, as the dial could always be turned higher, without reaching 1. Still the obligations to not turn the dial to 0 or 1 should be a given. In order to accommodate this dilemma, more complex definitions of obligations were given, as in the work of Hilpinen and McNamara in [GHP+13].

2.3.3 Deontic Paradoxes and Their Classification

As a basis for the work in this thesis, this subsection looks at different deontic paradoxes and their classification. The considered deontic paradoxes are examples for which SDL fails, i.e., is unable to capture the nuances of common sense reasoning.

As the number of deontic paradoxes is large, only some of the paradoxes for each of the classes will be considered and categorised according to the reason for their failure as seen

in [JC02]. Note that deontic paradoxes are also sometimes referred to under different names, such as e.g. "puzzles" or "dilemmas".

This subsection is structured in the following way and contains the following paradoxes:

1. Paradoxes centering around RMD
   - *Ross's Paradox*
   - *Good Samaritan Paradox*
   - *Åqvist's Paradox of Epistemic Obligation*
2. Puzzles centering around DD and OD
   - *Sartre's Dilemma*
   - *Plato's Dilemma*
3. Puzzles centering around deontic conditionals
   - *Broome's Counterexample*
   - *Chisholm's Contrary-to-Duty Paradox*
   - *Forrester's Paradox*
   - *Considerate Assassin Paradox*
   - *Asparagus Paradox*
   - *Fence Paradox*
   - *Alternative Service Paradox*
4. A problem with RND: The logical necessity of obligations
5. A problem with the idea of deontic logic: Jørgensen's dilemma

As a reminder the referenced rules and theorems are:

If $p \to q$ is a theorem, then $Op \to Oq$ is a theorem	(RMD)
$O\varphi \to \neg O \neg \varphi$	(DD)
$\neg O \bot$	(OD)
If φ is a theorem, $O\varphi$ is a theorem	(RND)

Deontic conditionals refer to obligations that arise situationally. Those conditionals (sometimes written as $O(A \mid B)$, meaning "it is obligatory that A if B") have been introduced to cope with contrary-to-duty obligations that arise due to another obligation not being fulfilled.
Note that the problem with RND and Jørgensen's dilemma are not really paradoxes but

of a philosophical nature.

Paradoxes centering around RMD
There are multiple paradoxes which arise due to the rule of inference RMD.
In general, these paradoxes show that SDL is too strong as they derive obligations, which might be seen as nonsensical using common sense reasoning.
Consider first *Ross's Paradox*:

Paradox 33.

It is obligatory that the letter is mailed.	(1)
It is obligatory that the letter is mailed or burned.	(2)

This paradox can be formalised as:

$O(m)$	(1)
$O(m \vee b)$	(2)

Since $m \to (m \vee b)$ is a theorem, the second obligation follows from the first via RMD in the following way:

$m \to (m \vee b)$	(Theorem)
$O(m) \to O(m \vee b)$	(Application of RMD)
$O(m \vee b)$	(Application of Modus Ponens)

One of the properties of an obligation is the possibility of not being satisfied. This can later be seen in the contrary-to-duty obligations. It seems unintuitive that one can derive an obligation that is satisfied by burning the letter, when failing to mail the letter. Some might actually consider the burning of the letter to be worse than simply failing to satisfy the obligation to mail the letter [JC02].

Another paradox is the *Good Samaritan Paradox*:

Paradox 34.

It is obligatory Jones helps Smith who is being mugged.	(1)
It is obligatory that Smith is being mugged.	(2)

One can now consider the following equivalence:

> Jones helps Smith who is being mugged if and only if Jones helps Smith and Smith is being mugged.

This equivalence then leads to the following formalization of the paradox:

$$O(h \wedge m) \tag{1}$$
$$O(m) \tag{2}$$

RMD once again leads to the second obligation arising from the first. The obligation of Smith being mugged is however an obligation, which would not be derived from the first obligation using common sense. Multiple variations of this paradox, such as the victim paradox have been proposed over time. They all consider situations, where an obligation arises from a certain act being performed, therefore making the act obligatory.

Finally we consider *Åqvist's Paradox of Epistemic Obligation*:

Paradox 35.

The bank is being robbed. (1)
It is obligatory that the guard knows that the bank is being robbed. (2)
It is obligatory that the bank is being robbed. (3)

Let K be an operator denoting knowledge of the guard. Then Kr would denote the guard knowing that the event r is happening (r in this case may denote the bank being robbed). Then a natural way to formalize this problem is:

$$r \tag{1}$$
$$O(Kr) \tag{2}$$
$$O(r) \tag{3}$$

Since knowledge about an event happening implies the event happening, one can assume that $K\varphi \to \varphi$ is a theorem. Then the obligation (3) follows directly from (2) in the following way:

$$Kr \to r \qquad \text{(Theorem)}$$
$$O(Kr) \to O(r) \qquad \text{(RMD)}$$
$$O(r) \qquad \text{(Application of modus ponens)}$$

Once again (3) is a rather nonsensical obligation to be derived.

Puzzles centering around DD and OD
Paradoxes that arise from DD and OD

$$Op \to \neg O\neg p \tag{DD}$$
$$\neg O\bot, \tag{OD}$$

are centered around obligations which are unfulfillable. Consider *Sartre's Dilemma*:

Paradox 36.

It is obligatory, that I meet Mary.	(1)
It is obligatory, that I do not meet Mary.	(2)

which intuitively would be formalised as:

$$O(m) \tag{1}$$
$$O(\neg m) \tag{2}$$

Such a situation could arise, when promising Mary to meet her, while promising another friend, not to meet Mary. The presented dilemma leads to a conflict, as (1) leads to $\neg O(\neg m)$, which directly contradicts (2). Another way to show the conflict that arises would be to use the theorem "aggregation for O":

$$(Op \wedge Oq) \rightarrow O(p \wedge q) \qquad \text{(Aggregation for O)}$$

Using this theorem we get:

$$O(m) \wedge O(\neg m) \rightarrow O(m \wedge \neg m)$$

and via Modus Ponens (as $m \wedge \neg m \rightarrow \bot$):

$$O(\bot)$$

Using the theorem "conjunctive distributivity of O", we can get the opposite direction of this implication:

$$O(p \wedge q) \rightarrow (Op \wedge Oq) \qquad \text{(Conjunctive Distributivity of O)}$$

This leads to a conflation of impossible obligations and conflicting obligations. While it is common to argue that something impossible cannot be obligatory, the same cannot be said about conflicting obligations. As such, conflict-allowing deontic logics have separated impossible obligations from conflicting obligations by ranking the obligations, for instance.

Another very similar dilemma is *Plato's Dilemma*:

Paradox 37.

It is obligatory that I meet my friend for dinner.	(1)
It is obligatory that I rush my child to the hospital.	(2)

In this scenario, a medical emergency has arisen, which necessitates immediate intervention. Due to temporary constraints, it is not possible that both obligations are fulfilled. Once again SDL is incapable of handling this concept. Using common sense reasoning,

most people would arrive at the conclusion that the second obligation invalidates the first obligation, as it is of higher importance.

Puzzles centering around deontic conditionals
Further paradoxes center around obligations that arise only under certain circumstances. First, consider the more abstract paradox of derived obligations, centered around their representation in SDL. Consider the statement:

Promising to meet Bob commits you to meeting him.

The two natural ways to represent this statement are:

$$O(p \to m) \tag{1}$$
$$p \to O(m) \tag{2}$$

However, both representations do not work. This leads to the question of whether a standard system has the resources to represent conditional obligations.

Consider first (1). The following theorems (derived through RMD), imply that anything which is forbidden would commit us to everything and for every obligation, everything would commit us to it.

$$O(\neg p) \to O(p \to m)$$
$$O(m) \to O(p \to m)$$

The first derived statement would then read: "If it is obligatory to not promise to meet Bob, then promising to meet Bob commits you to meeting him". While this specific statement may make sense, note that p and m could take any meaning, as the above formula is a theorem regardless of the meaning assigned to p and m! Therefore if one were to read p as parking illegally and m as commiting murder one could deduce from the obligation to not park illegally the obligation to commit murder, should one park illegally!

The second statement would read: "If it is obligatory to meet Bob, then promising to meet him commits you to meeting him". This statement once again makes sense, but the formula is a theorem regardless of the meaning assigned to p and m. Reading p as promising to not meet Bob and keeping the meaning of m would then lead to this nonsensical, yet derivable statement: "If it is obligatory to meet Bob, promising to not meet him commits you to meeting him".

(2) fails for similar reasons. Using the following logical tautologies we obtain the same absurd claims as above:

$$\neg p \to (p \to O(m))$$
$$O(m) \to (p \to O(m))$$

Therefore, there is no suitable way to represent these conditional obligations, without being able to deduce nonsensical obligations and statements.

(1) however fails for another reason as well, as the formulation in combination with (KD) leads to some unintuitive conclusions.

$$O(p \to q) \to (Op \to Oq) \tag{KD}$$

An example of such is *Broome's Counterexample* [Bro13], which shows the paradoxical nature of the rule (KD):

Paradox 38.

> It is obligatory, that one exercises.
> It is obligatory, that if one exercises one eats more.

Using the form of derived obligations (1) this leads to the obligation to eat more, which should not be derivable, as without exercising, eating more would be counterproductive.

The belief that a standard system does not have the capabilities to represent these conditional obligations and notions of commitment, was reinforced by the next paradoxes.

When talking about derived obligations one cannot omit the class of contrary to duty (CTD) obligations. CTD obligations arise due to some other obligation(s) not being fulfilled. An example can be seen in the following situation:

> You are obligated to go to the meeting.
> If you do not go to the meeting, you are obligated to let your boss know.

The second obligation is an example of a CTD obligation. It arises only when one should fail to satisfy the first obligation.

Chisholm's Contrary-to-Duty Paradox, which consists of the following four statements, exemplifies the issue with CTD obligations:

Paradox 39.

It ought to be that Jones goes to the assistance of his neighbors. (1)
It ought to be that if Jones goes to the assistance of his neighbors, then he tells them he is coming. (2)
If Jones doesn't go to the assistance of his neighbors, then he ought not tell them he is coming. (3)
Jones does not go to their assistance. (4)

These four sentences are widely considered as mutually consistent and logically independent. Treating mutual consistency and logical independence as desiderata, however leads to no working formulation, as formulations either fail due to a contradiction arising or the sentences ceasing to be independent. In general the following three possible formalisations are considered, when talking about *Chisholm's Paradox*.

Formalisation 1 takes the following form:

$$O(g) \quad (1)$$
$$O(g \to t) \quad (2)$$
$$\neg g \to O(\neg t) \quad (3)$$
$$\neg g \quad (4)$$

Using KD, one can derive $O(g) \to O(t)$ from (2), which using (1) leads to $O(t)$. However, applying modus ponens to (4) and (3) leads to $O(\neg t)$, which contradicts the theorem DD.

$$Op \to \neg O \neg p \quad \text{(DD)}$$

The alternative Formalisation 2 is:

$$O(g) \quad (1)$$
$$O(g \to t) \quad (2')$$
$$O(\neg g \to \neg t) \quad (3)$$
$$\neg g \quad (4)$$

It does not consist of four logically independent sentences, as $O(g) \vdash O(\neg g \to \neg t)$. This holds as $g \to (g \vee \neg t) \leftrightarrow (\neg g \vee \neg t)$.

Formalisation 3 is:

$$O(g) \quad (1)$$
$$g \to O(t) \quad (2)$$
$$\neg g \to O(\neg t) \quad (3')$$
$$\neg g \quad (4)$$

It similarly fails, as $\neg g \vdash g \rightarrow O(t)$.

Another paradox in this class is *Forrester's Paradox*, also referred to as the *"Gentle Killer Paradox"*:

Paradox 40.

Smith ought not kill Jones.	(1)
If Smith will kill Jones, then Smith ought to kill Jones gently.	(2)
Smith will kill Jones.	(3)

Formalised this paradox takes this form:

$O(\neg k)$	(1)
$k \rightarrow O(g)$	(2)
k	(3)

Applying modus ponens we derive $O(g)$. In this formulation the implication $g \rightarrow k$ holds, as killing someone gently obviously implies killing someone. Using RMD $O(g) \rightarrow O(k)$ can be derived, which using modus ponens leads to a contradiction.

A similar paradox is the *Considerate Assassin Paradox* [PS96]:

Paradox 41.

You should not kill the witness.	(1)
If you kill the witness, you should offer him a cigarette.	(2)
You should not offer a cigarette.	(3)
You kill the witness.	(4)

This paradox is created by combining two different moral codes. The first two obligations are part of the supposed mafia rules and the third obligation is part of general morals. The formalisation of the paradox takes the following form:

$O(\neg k)$	(1)
$k \rightarrow O(oc)$	(2)
$O(\neg oc)$	(3)
k	(4)

Through (4) and (2) $O(oc)$ can be derived by applying modus ponens. This obviously contradicts (3) thereby leading to this paradox.

A slightly different kind of derived obligation can be seen in the (slightly altered) *Asparagus Paradox* [vdT94, Hor97]:

Paradox 42.

Don't eat with your fingers.	(1)
If you are served cold asparagus, eat it with your fingers.	(2)
You are served cold asparagus.	(3)

In this case (2) is obviously intended as an exception to (1). However, the formalisation of these statements takes this form:

$O(\neg fingers)$	(1)
$asparagus \rightarrow O(fingers)$	(2)
$asparagus$	(3)

One can once again see that (3) and (2) lead to $O(fingers)$ via modus ponens, thereby creating a contradiction.

An interesting paradox that combines two different weaknesses of SDL is the unnamed paradox that we refer to as the *Fence Paradox* [PS96]:

Paradox 43.

There must be no fence.	(1)
If there is a fence then it must be a white fence.	(2)
If the cottage is by the sea, there may be a fence.	(3)

Here (2) serves as a contrary-to-duty obligation that is active when obligation (1) is violated. (3) serves as an exception to the obligation generated by (1). Note that interpreting it this way does not necessitate a fence being white if the cottage is by the sea.

In order to better showcase how this paradox should be understood, we rephrase it in the following way:

> There must be no fence, unless the cottage is by the sea.
>
> If there is a fence in violation of an obligation, then it must be a white fence.

This paradox fails for reasons seen before in the *Asparagus Paradox* and *Forrester's Paradox*. The reason for considering this paradox will be apparent later when looking at the encoding on page 60.

A slightly different inadequacy SDL faces is the lack of expressibility brought forth through the conjunctive distributivity of O [Han08]:

$$O(p \wedge q) \rightarrow (Op \wedge Oq) \qquad \text{(Conjunctive Distributivity of } O\text{)}$$

This theorem makes it impossible to model obligations, where failing a part of the obligation makes satisfying the rest unnecessary. Consider *Broome's Counterexample*, discussed on page 29, rephrased in the following way:

> It is obligatory to exercise and eat more (in order to live a healthier life).

Due to the conjunctive distributivity one derives the obligation to eat more, which should not be given when the other part of the obligation is not satisfied.

Another such example can be seen when one promises to bring a salad to a party hosted by a friend [Han08]. In order to bring a salad, one would have the obligation to buy lettuce and buy dressing. However, failing to buy the lettuce (and therefore not satisfying the obligation) would render the obligation to buy dressing moot. (Appearing to the party with only the dressing might even be considered worse than simply forgetting the salad.)

The final problem to be mentioned is the *Alternative Service Paradox* [Hor94], as it shows the inability of SDL to adequatly deal with obligations over disjunctions:

Paradox 44.

> You are obligated to fight in the army or perform alternative service.
>
> You are obligated to not fight in the army.

One would hope to derive the obligation to perform alternative service from this, however as there does not exist disjunctive distributivity of O, this is not possible [Hor94].

A solution for this might be to simply add disjunctive distributivity of O to the axioms. However, this leads to its own problems. Consider the following obligation:

> You are obligated to pay 500 euros or not buy the bike.

However, neither the obligation to pay 500 euros nor the obligation to not buy the bike holds, thereby rendering disjunctive distributivity somewhat nonsensical.

A problem with RND: The logical necessity of obligations

We now consider a paradox that, although it is syntactically not part of SDL, as it goes

beyond propositional logic, is of interest as it shows a more fundamental problem with SDL. Consider the following statement:

Nothing is obligatory,

which could be formalised as:

$$\neg\exists q O q.$$

or

$$\forall q \neg O q.$$

Although this may be a valid statement in certain situations, it leads to problems when considering the axiom RND,

If p is a theorem, Op is a theorem. (RND)

As $\top$ is a theorem it follows that $O\top$ must be a theorem as well. As $Op \vdash \exists p Op$ (propositional quantification), it then follows that $O\top \vdash \exists p Op$ and as $O\top$ is a theorem $\vdash \exists p Op$ holds. As such it seems that SDL necessitates the existence of obligations, although a lack of obligations could plausibly exist given fitting circumstances.

Multiple works have considered this problem, with von Wright's opinion being that the obligations which arise from tautologies are a "principle of contingency" for SDL. As it is neither possible to fulfill nor violate these obligations this problem is considered by some researchers to "not be a pressing concern". However, there exist deontic logics that do not contain the axiom RND and therefore do not run into this problem.

A problem with the idea of deontic logic: Jørgensen's dilemma

The origin of Jørgensen's dilemma was the view that evaluative sentences, such as "This is wrong/right", are not sentences to which a truth value could be assigned. This is still a common viewpoint of researchers working on deontic logics and metaethics.

However, this viewpoint leads to a problem, as deductive logic tries to deduce the truth of statements given the truth value of other statements. This problem is Jørgensen's dilemma. Given that the sentences considered in deontic logic are evaluative ones, this would render deontic logic an impossibility. On the other hand, deontic statements seem to stand in some logical relationship when using common sense reasoning, which would suggest that the existence of a deontic logic should be possible.

Multiple solutions to this problem have been proposed, see [JC02] or McNamara and Hilpinen in [GHP+13]. One is to distinguish between "norms" and "normative

propositions". Consider the following statement:

It is forbidden to park in this spot.

This statement could be made by some authority, such as the municipality or by a passerby, intending to inform you of the no-parking zone.

Such a statement being made by an authority can be referred to as "norming". The authority creates the norm by using the sentence (thereby arguably rendering it true), whereas the usage of the statement by the passerby is descriptive (it could be read as "Authority forbids parking in this spot") and can therefore be evaluated to true or false.

All in all, Jørgensen's dilemma is more of a philosophical problem rather than a problem with the logic itself. Therefore, it will not be considered in the next chapter, when encoding the paradoxes in DLV.

While there are further expressive inadequacies for SDL, they will not be considered in this work. Some of them can be found in the work of Hilpinen and McNamara in [GHP+13].

CHAPTER 3

Encoding the Classes of Paradoxes

We encode the deontic paradoxes presented in the previous chapter in DLV. We start by discussing the intuition behind our encoding. Finally, we discuss related work.

3.1 Vocabulary

Our encodings use the following predicates:

- $O(X)$ denotes X being obligatory.
- $F(X)$ denotes X being forbidden.
- $act(X)$ denotes that we want to reason about whether X is obligatory or not. The name *act* was chosen for the predicate as we usually reason about actions. There are some cases where we reason about obligations that do not necessarily constitute as actions, however we also use this predicate in those cases for the sake of consistency.
- $Do(X)$ denotes that the agent has chosen to take the action X. Note that $-Do(X)$ denotes that the agent will not take the action X.
- $Diamond(X)$ is an auxiliary predicate used to denote that an action X is possible. The naming is a reference to modal logics, where the diamond operator represents possibility. Note that $-Diamond(X)$ can either mean that the agent cannot take the action or that the agent has chosen not to take the action.
- $Happens(X)$ is an auxiliary predicate that denotes an event X happening. It is sometimes used in encodings to denote events happening which are usually outside the agents control.

3.2 Methodology

We start by explaining the meanings of the answer sets in the encodings.

Answer sets represent "ethically optimal" ways for an agent to handle the encoded situations. In other words, each answer set represents a way of handling the imposed obligations, with none of the answer sets being preferable to any other. Consider the following example:

Example 45. (Answer sets)
It is obligatory for an agent to either rest or walk. There are two ways to fulfil this obligation. The agent can either rest or walk. (Arguably both but this might be considered impossible.) This could be represented by the following two answer sets:

- $\{Do(walk), O(walk)\}$
- $\{Do(sleep), O(sleep)\}$

The agent then has the choice between the two derived answer sets. Although it may seem like either the obligation to sleep or the obligation to walk is derived, that is actually not the case. The agent has the choice between the two answer sets at that point. The choice between the two options could for example be taken by a reinforcement learning agent that can choose the optimal option under given circumstances. This will be seen in a later chapter which encodes a normative supervisor using ASP as in [NBCG21].

The following sections will encode the paradoxes presented in the preliminaries. The goal is to uniformly encode the paradoxes in order to later get a general method for encoding paradoxes. Therefore, all encodings will share this common core, which forms the basis of the encodings.

$$
\begin{array}{ll}
O(X) \vee -O(X) :- act(X). & (1)\\
F(X) \vee -F(X) :- act(X). & (2)\\
:- O(X), -Diamond(X). & (3)\\
-Diamond(X) :- -Do(X), act(X). & (4)\\
:- O(X), F(X). & (5)\\
Do(X) \vee -Do(X) :- act(X). & (6)\\
:- F(X), Do(X). & (7)\\
Happens(X) :- Do(X). & (8)\\
:- Do(X), -Diamond(X). & (9)\\
:\sim O(X).[1:1] & (10)\\
:\sim F(X).[1:1] & (11)
\end{array}
$$

In the encoding, the predicate *act* denotes a constant as an action. This distinction is made in order for the logic program to only reasons about whether or not taking an action is obligatory. The first two rules simply denote that each action is either obligatory or not and also each action is either forbidden or not.

$$O(X) \vee -O(X) :- act(X).$$
$$F(X) \vee -F(X) :- act(X).$$

The third rule states that an obligation to take an action X requires that it is possible to take that action, or in other words, it is not possible that an action is obligatory and not possible:

$$:- O(X), -Diamond(X).$$

This notion stems from Kant's law (see Hilpinen and McNamara in [GHP+13]), which states:

> Anything morally obligatory for an agent must be within the agent's ability.

Kant's law originated in discussion about ethical morality.
The fourth rule then simply links $-Do(X)$ and $-Diamond(X)$. As explained previously, $-Diamond(X)$ can either mean an action not being possible or the agent choosing not to take the action.

$$-Diamond(X) :- -Do(X), act(X).$$

The third and fourth rule in combination commit the agent to taking an action that he is obliged to take in the given answer set. This follows as $-Do(X)$ and $act(X)$ imply $-Diamond(X)$, while $O(X)$ implies $Diamond(X)$, thereby leading to a conflict should $O(X)$ and $-Do(X)$ be in the same answer set. Note that this necessitates conflicts between actions/obligations to be encoded as well.

In order to avoid deducing that an action is both obligatory and forbidden, the fifth rules leads to a conflict should such an action exist:

$$:- O(X), F(X).$$

Rule (6) states that for each action the agent must decide whether he takes that action or not.

$$Do(X) \vee -Do(X) :- act(X).$$

In order to secure that the agent does not take a forbidden action, we add rule (7):

$$:- F(X), Do(X).$$

This rule might raise the question of why there is no corresponding rule for $O(X)$ that ensures that the agent takes an obligatory action:

$$:- O(X), -Do(X).$$

The reason for this is that this rule is implicitly given by rules (3), (4) and (6). As either $Do(X)$ or $-Do(X)$ must be in the answer set by rule (6), assume that $-Do(X)$ and $O(X)$ were in the same answer set. Due to $-Do(X)$, we can derive $-Diamond(X)$ via rule (4). ($act(X)$ is given as else $O(X)$ would not be derived.) However $-Diamond(X)$ and $O(X)$ cannot be in the same answer set because of (3). Therefore an equivalent rule to (7) is given implicitly.

Rule (8) links the two predicates $Happens$ and Do. Intuitively taking an action makes that action happen:

$$Happens(X) :- Do(X).$$

Rule (9) links the predicates Do and $Diamond$ in a different way. It intuitively forbids the agent from taking actions which are deemed not possible:

$$:- Do(X), -Diamond(X).$$

As the first two rules could create answer sets with obligations or prohibitions, which arise simply because there is no argument against them arising, the following weak constraints are introduced to filter out those answer sets.

$$:\sim O(X).[1:1]$$
$$:\sim F(X).[1:1]$$

Note that these weak constraints are set at the lowest level. All obligations are created through weak constraints at a higher level as all obligations are considered defeasible and we do not want an obligation not to be derived, because of the weak constraints (10) and (11). Note that because DLV minimises the weight of the violated weak constraints (prioritising the higher levels) these obligations are derived unless a conflicting obligation or task is derived.

In general all obligations are generated using weak constraints as will be shown in the following sections.

We will start by considering the correctness of the common core. In the beginning we will only consider the rules (1) – (9) and ignore the weak constraints (10) and (11).

To test the common core we add only one action to reason about. Let this be denoted by *action*. This reduced program then looks like this:

$$
\begin{array}{ll}
O(X) \vee -O(X) :- act(X). & (1)\\
F(X) \vee -F(X) :- act(X). & (2)\\
:- O(X), -Diamond(X). & (3)\\
-Diamond(X) :- -Do(X), act(X). & (4)\\
:- O(X), F(X). & (5)\\
Do(X) \vee -Do(X) :- act(X). & (6)\\
:- F(X), Do(X). & (7)\\
Happens(X) :- Do(X). & (8)\\
:- Do(X), -Diamond(X). & (9)\\
act(action). &
\end{array}
$$

The idea behind the common core is to create all possible consistent ways of combining $O(action), F(action), Happens(action), Do(action), Diamond(action)$. Note that $Diamond(action)$ cannot appear as a positive predicate. Furthermore, $act(action)$ must appear in every answer set. We want the following combinations of predicates to appear:

1. One of the answer sets should deduce $O(action)$, the obligation to take the action. The action should not be forbidden. Consequently, $-F(action)$ should be in the answer set. In this case we also want the agent to take the action, therefore $Do(action)$ should appear in the answer set as well. Therefore, $Happens(action)$ should also be in the answer set. We therefore want one answer set to take the following form:

$$\{act(action), O(action), -F(action), Do(action), Happens(action)\}$$

2. One of the answer sets should deduce $F(action)$, forbidding the action to be taken. The action should not be obligated. Consequently, $-O(action)$ should be in the answer set. In this case we also do not want the agent to take the action, therefore $-Do(action)$ should appear in the answer set as well. Therefore, $-Diamond(action)$ should be in the answer set too (as $-Diamond(action)$ can also mean that the agent does not choose to take the action). We therefore want one answer set to take the following form:

$$\{act(action), F(action), -O(action), -Do(action), -Diamond(action)\}$$

3. We want two answer sets where the action is neither obligatory nor forbidden. In one the agent chooses to take the action, and in one the agent does not. Using

similar reasoning as in the cases above we want the two answer sets to take the following form:

$$\{act(action), -F(action), -O(action), -Do(action), -Diamond(action)\}$$
$$\{act(action), -F(action), -O(action), Do(action), Happens(action)\}$$

The common core would create the following answer sets:

1. An answer set where the action is guessed to be obligatory due to rule (1). Rule (2) can then only guess $-F(action)$, as rule (5) forbids $O(action)$ and $F(action)$ from being in the same answer set. Rule (6) can then only guess $Do(action)$, as $-Do(action)$ leads to $-Diamond(action)$ due to rule (4). However, $-Diamond(action)$ cannot appear in the answer set because of rule (3). Due to rule (8), $Happens(action)$ is in the answer set as well. This leads to the following answer set:

$$\{act(action), O(action), -F(action), Do(action), Happens(action)\}$$

2. An answer set where the action is guessed to be forbidden due to rule (2). Rule (1) can then only guess $-O(action)$, as rule (5) forbids $O(action)$ and $F(action)$ from being in the same answer set. Rule (6) can then only guess $-Do(action)$, as rule (7) would forbid it. Due to rule (4), $-Diamond(X)$ is in the answer set as well. This leads to the following answer set:

$$\{act(action), F(action), -O(action), -Do(action), -Diamond(action)\}$$

3. An answer sets where in rule (1) and (2) the action is guessed to neither be obligatory nor forbidden and $-Do(action)$ is chosen in rule (6). Due to rule (4) $-Diamond(action)$ is in the answer set as well. This leads to the following answer set:

$$\{act(action), -F(action), -O(action), -Do(action), -Diamond(action)\}$$

4. An answer sets where in rule (1) and (2) the action is guessed to neither be obligatory nor forbidden and $Do(action)$ is chosen in rule (6). Due to rule (8) $Happens(action)$ is in the answer set as well. This leads to the following answer set:

$$\{act(action), -F(action), -O(action), Do(action), Happens(action)\}$$

As those answer sets represent all desired possible cases, completeness is given.

Let us now consider combinations of predicates for an action that we want to exclude as they are inconsistent:

- $\{-Diamond(action), O(action)\}$ is not allowed, as we do not want an action to not be possible to take and obligatory at the same time. Due to rule (4) this combination of predicates cannot appear as a subset of an answer set.

- $\{O(action), -Do(action)\}$ is not allowed, as we do not want the agent to not take an action that has been deemed obligatory under given circumstances. Such a combination cannot appear as a subset of an answer set as $-Do(action)$ would lead to $-Diamond(action)$ for an action due to rule (4). As seen above $-Diamond(action)$ and $O(action)$ cannot appear in the same answer set and therefore this combination of predicates cannot appear as a subset of an answer set.

- $\{O(action), F(action)\}$ is not allowed, as we do not want an action to be both obligatory and forbidden. Due to rule (5) this combination of predicates cannot appear as a subset of an answer set.

- $\{F(action), Do(action)\}$ is not allowed as we do not want an agent to take an action which has been deemed forbidden under given circumstances. Due to rule (7) this combination of predicates cannot appear as a subset of an answer set.

- $\{-Diamond(action), Do(action)\}$ is not allowed as we do not want an agent to take an action which has been deemed impossible under current circumstances. Due to rule (9) this combination of predicates cannot appear as a subset of an answer set.

The answer sets $1-4$ all fulfil the above conditions. Therefore, our methodology fulfils soundness as well.

Adding the weak constraints removes the last two answer sets as the obligation and the prohibition increase the weight of the violated weak constraints (10) and (11). The goal is to filter out obligations resp. prohibitions which are not created through weak constraints at higher levels. This is done in order to ensure that no baseless obligations and prohibitions can be found in any answer set.

In general obligations that are always active (unless a conflicting obligation arises) are encoded as a weak constraint on the second level with weight 1. Let *assist* be an obligatory action. This would be written in the following way in DLV:

$$:\sim O(assist).[1:2]$$

There are also different ways of filtering out unwanted answer sets. A more intricate way is given by [BDRS15] or [ADMR20] which allow for more sophisticated preference models. Since weak constraints are able to fulfill the desiderata set forth by this work, this is not used.

3.2.1 Properties of the Common Core and the Axioms of SDL

This subsection will take a look at some properties of the common core as well as discuss how the axioms of SDL are handled in the common core.

The first property we will consider is that any answer set that contains $O(action)$ also contains $Do(action)$ for any action and on the other hand any answer set that contains $F(action)$ also contains $-Do(action)$. In other words, the agent has to take any action that is determined as obligatory. An explanation for this behaviour can be seen on page 42.

While this may seem like an undesired property on the first glance, it is explained through the semantics of our methodology. In our semantics actions are determined as obligatory, if and only if taking that action is the optimal way to act given the encoded normative system. Intuitively, obligations that are less important are waived, should fulfilling them hinder the agent from fulfilling more important obligations. Note that if an action is neither determined as obligatory nor forbidden, the agent can choose to take or not take the action, if it is possible for the agent to take said action.

The predicate *Diamond* generally only appears negatively in answer sets. As DLV can only derive predicates that appear in the head of a rule and *Diamond* does not appear in any rule it does not appear in any answer set. The reason for us not having a rule that derives $Diamond(X)$ once again lies in the semantics of our methodology. $-Diamon(action)$ encodes the fact that it is not possible for the agent to take said action. In our encoding any action that is not specifically deemed impossible is considered possible. (One could theoretically add code that determines an action to be possible unless specified otherwise, we chose to omit this to keep the code simple.)

The predicate *Happens* generally only appears positively in answer sets. The reason for this is the same as for *Diamond*. If an event does not happen, it is not specifically encoded in our methodology. Note that *Happens* is not only used to denote that an action that the agent has chosen to take has happened, but it is also used to denote events that are outside the agent's control.

A final notable property of our methodology is that DLV only reasons about actions that are specifically denoted as such through the predicate *act*. This is due to the fact that all rules that would lead to an obligation or prohibition being derived (rule (1) and (2)) contains $act(X)$ in the body of the rule.

We will now take a look at the axioms of SDL (that are not part of the axiomatization of classical propositional logic) and how they are encoded or why we have chosen to omit an encoding for them in our common core.

$$\text{If } \varphi \text{ is a theorem, } O\varphi \text{ is a theorem} \tag{RND}$$
$$O(\varphi \to \psi) \to (O\varphi \to O\psi) \tag{KD}$$
$$O\varphi \to \neg O \neg \varphi \tag{DD}$$

The axiom (RND) is not encoded in our common core for multiple reasons. As in our encoding O is a predicate rather than an operator, logical connectives cannot appear inside O. Although $O(a \vee \neg a)$ is a theorem in SDL, it would be nonsensical to encode it in our methodology. As the agent has no choice but to take an action that always happens (such as either doing or not doing something) encoding it would only make the code more complicated without actually benefitting us in any way.

(KD) is not encoded in the common core for similar reasons. It is also related to how we encode derived obligations. This will be discussed further on page 69.

(DD) is encoded indirectly. Note that for an act *action* we would encode $O(\neg action)$ as $F(action)$. This can be understood as the obligation to not take an action being equivalent to that action being forbidden. So we can rephrase the axiom (DD) as:

$$O\varphi \to \neg F\varphi$$

which is equivalent to

$$\neg(O\varphi \wedge F\varphi)$$

Through this transformation we can see that (DD) is part of our common core through rule (5):

$$:- O(X), F(X).$$

3.3 Ross's Paradox

We first encode *Ross's Paradox*:

> It is obligatory that the letter is mailed.
> (It is obligatory that the letter is mailed or burned.)

The second sentence (in brackets) is a sentence which would be derived in SDL, but is nonsensical using common sense reasoning. To show that such an obligation is not derived in DLV, we encode it by adding the following rules and facts to the core:

$$\boxed{\begin{array}{l} :\sim -O(mail).[1:2] \\ act(mail). \\ act(burn). \end{array}}$$

Note that a disjunction over obligations is symbolised by two different answer sets that each contain one possible way to satisfy the obligation over the disjunction.

First, the obligation is created using the following weak constraint:

$$:\sim -O(mail).[1:2]$$

Since *mail* is specified as an action, the logic program has to denote it as obligatory or not obligatory, as this is specified in the core of the program. The weak constraint penalises the system at the highest level if the program does not specify mailing the letter as obligatory. Since the program minimises the constraints violated at the highest level, all answer sets deduce the obligation to mail the letter, should such an answer set exist.

The final two facts simply denote *mail* and *burn* as actions to reason about.

This logic program yields two answer sets:

$$\{act(mail), act(burn), Do(mail), -Do(burn), Happens(mail), \\ -Diamond(burn), -F(mail), -F(burn), O(mail), -O(burn)\}$$

and

$$\{act(mail), act(burn), Do(mail), Do(burn), Happens(mail), \\ Happens(burn), -F(mail), -F(burn), O(mail), -O(burn)\}$$

Note that neither of these answer sets derive the obligation to burn the letter. The only difference between the answer sets is whether the agent chooses to burn the letter or not. This is valid as it is not forbidden to burn the letter and it is not specified that it is not possible to both burn and mail the letter.

One may add a rule specifying additional information about the paradox:

$$:- Do(mail), Do(burn).$$

This rule specifies that it is not possible to both mail and burn the letter.

This logic program then yields only a single viable answer set:

$$\{act(mail), act(burn), Do(mail), -Do(burn), Happens(mail), \\ -Diamond(burn), -F(mail), -F(burn), O(mail), -O(burn)\}$$

This answer set derives the obligation to mail the letter and does not deduce any further obligations, which would be the natural way to solve this problem.

One may also add an additional rule stating that mailing the letter implies mailing or burning the letter as this causes the paradox in SDL. However, adding such a rule does not change the outcome, as a predicate in the head that is not in the body would never be derived.

3.4 Good Samaritan Paradox

The *Good Samaritan Paradox* gets encoded similarly.

> It is obligatory Jones helps Smith who is being mugged.
> (It is obligatory that Smith is being mugged.)

Once again the sentence in the bracket is an obligation, which is derived in SDL. The difference to the previous paradox is that the obligation of helping Smith arises as Smith is being mugged. The agent Jones therefore has to reason with the knowledge of this mugging happening. This gets formalised through the auxiliary predicate *Happens* that denotes an event taking place. This leads to the following addition to the core:

$$\boxed{\begin{array}{l} :\sim -O(help), Happens(mug).[1:2] \\ Happens(mug). \\ act(help). \end{array}}$$

$Happens(mug)$ encodes the information that the mugging of Smith is currently happening and $act(help)$ specifies that the logic program is to reason about the obligation of helping.

The weak constraint argues that answer sets may either have no obligation to help or have the mugging happen, but not both. Deriving an obligation to help while not having a mugging happen would not violate this rule, but would not be derivable because the common core would filter such a solution.

This encoding returns one valid answer set:

$$\{act(help), Happens(mug), Do(help), Happens(help), -F(help), O(help)\}$$

This answer set derives the obligation to help and therefore solves the paradox in an intuitive way. Note that if $Happens(mug)$ was not a fact, the obligation to help would

not be derived. In that case, the following answer sets are derived:

$$\{act(help), -Do(help), -Diamond(help), -F(help), -O(help)\}$$

and

$$\{act(help), Do(help), Happens(help), -F(help), -O(help)\}$$

Neither of these answer sets derived any obligations. The difference between the answer sets simply lies in whether the agent decides to help Smith although no mugging is happening. As there is no rule prohibiting the agent from helping Smith when there is no emergency this is valid. Should one want to establish such a constraint one could add a weak constraint or rule that specifies this. Such a constraint could take the following form:

$$:\sim Do(help), not\ \ Happens(mug).[1:2]$$

The above way of encoding does not give the agent the option to reason about whether to mug Smith. In order to show that the agent does not derive the obligation to mug Smith even when given the option, we consider the following slightly edited encoding:

$$\begin{array}{|l|}\hline :\sim -O(help), Happens(mug).[1:2] \\ act(help). \\ act(mug). \\ \hline \end{array}$$

The fact $act(mug).$ denotes mugging as an action that the agents needs to reason about. Note that rule (8) in the common core specifies that mugging Smith implies that a mugging is happening.

This encoding leads to two different answer sets:

$$\{act(help), act(mug), -Do(help), -Do(mug), -Diamond(help),\\ -Diamond(mug), -F(help), -F(mug), -O(help), -O(mug)\}$$

and

$$\{act(help), act(mug), Do(help), -Do(mug), Happens(help),\\ -Diamond(mug), -F(help), -F(mug), -O(help), -O(mug)\}$$

Neither of these answer sets derive the obligation to mug Smith. The only relevant difference between the two answer set is whether the agent still chooses to help Smith, although Smith is not being mugged. As there is no prohibition on helping Smith both answer sets make sense in this case and capture the intuitive solutions to the given

situation.
When given the additional fact $Happens(mug)$ the following two answer sets are derived:

$$\{act(help), act(mug), Happens(mug), Do(help), -Do(mug), Happens(help), \\ -Diamond(mug), -F(help), -F(mug), O(help), -O(mug)\}$$

and

$$\{act(help), act(mug), Happens(mug), Do(help), Do(mug), \\ Happens(help), -F(help), -F(mug), O(help), -O(mug)\}$$

Both answer sets in this case still derive the obligation to help Smith. The difference between the answer sets is whether the agent mugs Smith. As there is no rule specifying that it is not possible to both help and mug Smith or even a prohibition stopping the agent from mugging Smith, this is a valid answer in this case. Should one add such a rule then the second answer set would not be derived.

3.5 Åqvist's Paradox of Epistemic Obligation

Next, we consider *Åqvist's Paradox of Epistemic Obligation*:

> The bank is being robbed.
> It is obligatory, that the guard knows the bank is being robbed.
> (It is obligatory, that the bank is being robbed.)

Like in the previous paradox we use the auxiliary predicate *Happens*. Additionally we introduce the auxiliary function $k(x)$, which denotes the knowledge of x happening. We formulate this in the following way:

$$\begin{array}{|l} :\sim -O(k(robbery)), Happens(robbery).[1:2] \\ Happens(robbery). \\ act(k(robbery)). \end{array}$$

Note at this point that the predicate *act* denotes $k(robbery)$ as an action in a broader sense. While it is debatable whether knowing is an action, knowing whether a robbery is happening is still the "action" being reasoned about.

The weak constraint once again ensures that should a robbery happen, taking notice of the robbery happening is obligatory for the agent. Finally, the last rule specifies the

knowledge of something happening as an action to reason about. The single answer set returned by this problem is:

$$\{act(k(robbery)), Happens(robbery), Do(k(robbery)),$$
$$Happens(k(robbery)), -F(k(robbery)), O(k(robbery))\}$$

Once again the same solution is reached that would be reached using common sense reasoning. In order to more precisely model the semantics of the k operator an additional rule could be added. This rule specifies that knowing a robbery is happening is impossible when no robbery is happening. A more general form of this constraint looks like this:

$$:- notHappens(X), Do(k(X)).$$

This however does not influence the solution.

3.6 Sartre's Dilemma

Consider *Sartre's Dilemma* in which the obligations are equally important:

It is obligatory, that I meet Mary.
It is obligatory, that I do not meet Mary.

In cases where two (or more) obligations are equally important one would expect multiple possible solutions, as there is no real reason to prefer one obligation over the other. Actually the existence of multiple solutions in such cases is a feature that is crucial in certain use cases. Consider an implementation of the normative supervisor in [NBCG21]. The agent chooses which action to take from "ethically optimal" actions, which the normative supervisor passed on to the agent. In case only one of those "ethically equivalent" actions were passed on to the agent, the agent would possibly have to take a suboptimal action.

Considering this requirement *Sartre's Dilemma* can be encoded in the following way (once again the code is given excluding the common core):

$$\boxed{\begin{array}{l} :\sim -O(m).[1:2] \\ :\sim -F(m).[1:2] \\ act(m). \end{array}}$$

The first two weak constraints denote the obligations in the same way known from previous encodings. m denotes the act of meeting Mary here. The final rule once again denotes m as an action to be reasoned about.

Due to the rule $:- O(X), F(X).$ in the core of the logic program it is not possible for an answer set to contain both $O(m)$ and $F(m)$. Therefore, an optimal solution violates at least one of the weak constraints set at the second level. As it is possible to satisfy either one of the weak constraints at the second level an optimal solution satisfies exactly one of these weak constraints leading to the following two answer sets:

$$\{act(m), Do(m), Happens(m), -F(m), O(m)\}$$

and

$$\{act(m), -Do(m), -Diamond(m), F(m), -O(m)\}$$

The two answer sets fulfill the desiderata, as they derive the obligation to meet Mary resp. the obligation to not meet Mary.

3.7 Plato's Dilemma

Since in general the conflicting obligations may not be directly conflicting but indirectly conflicting, it may be necessary to add a rule specifying that it is not possible to take both actions. This can be nicely seen in the encoding of *Plato's Dilemma*, in which due to a medical emergency an obligation of higher priority arises:

> It is obligatory, that I meet my friend for dinner.
> It is obligatory, that I rush my child to the hospital.

Due to time constraints it is not possible for the agent to satisfy both of the obligations.

The desired outcome of *Plato's Dilemma* would be for the agent to take their child to the hospital, thereby violating the obligation of meeting the agent's friend for dinner.

The two interesting aspects of this encoding are the prioritisation of the obligations and the impossibility of satisfying of taking both actions. This can be encoded in the following way:

$$\begin{array}{|l}
\hline
:\sim -O(help), Happens(emergency).[1:3] \\
:\sim -O(meet).[1:2] \\
act(meet). \\
act(help). \\
:- Do(help), Do(meet). \\
Happens(emergency). \\
\hline
\end{array}$$

The encoding starts with a weak constraint at level 3 (the highest level in this encoding), which penalises answer sets in which *emergency* is true but the obligation to help is not derived. In other words, it derives the obligation to help the child in case of an emergency. *emergency* is an auxiliary predicate, which denotes that an emergency is currently occurring. The second weak constraint simply encodes the obligation to meet the friend for dinner.

Due to the first weak constraint being at a higher level than the second, the program always ensures that the first weak constraint is satisfied before considering the second weak constraint.

$:- Do(help), Do(meet).$ encodes that it is not possible to both help the child and meet the friend. The rest of the assertions simply state that an emergency is currently occuring and which actions to reason about.

This encoding leads to the following answer set:

$$\{act(help), act(meet), Happens(emergency), Do(help), -Do(meet), Happens(help), \\ -Diamond(meet), -F(help), -F(meet), O(help), -O(meet)\}$$

As desired only a single possible solution is computed, as the program prioritises the obligation to help the child.

3.8 Broome's Counterexample

Broome's Counterexample involves an obligation that is derived by taking an action. We encode derived obligations of that kind in the following way:

$$:\sim Do(p), -O(m).[1:2]$$

This weak constraint can informally be understood as: "Should one take action p, then one should be obligated to take action m". Note that the weight and level of the weak constraint depends on whether any conflicting obligations arise, as seen in *Plato's Dilemma*. Using this way of encoding derived obligations, we encode *Broome's Counterexample*:

> It is obligatory, that one exercises.
>
> It is obligatory, that if one exercises one eats more.

Using the encoding we add the following to the common core:

$$
\begin{array}{l}
:\sim -O(exercise).[1:2] \\
:\sim Do(exercise), -O(eat).[1:2] \\
act(exercise). \\
act(eat).
\end{array}
$$

As expected, there is only one answer set, which derives the obligation to exercise, as well as the obligation to eat more:

$$\{act(exercise), act(eat), Do(exercise), Do(eat), Happens(exercise), \\ Happens(eat), -F(exercise), -F(eat), O(exercise), O(eat)\}$$

However, the problem in *Broome's Counterexample* arises due to the obligation of exercising not being fulfilled but the obligation to eat more still being derived. Therefore, we add $-Do(exercise).$ to the encoding.

This leads to two optimal answer sets, which both do not derive the obligation to eat.

$$\{act(exercise), act(eat), -Diamond(exercise), -Do(exercise), -Do(eat), \\ -Diamond(eat), -F(exercise), -F(eat), -O(exercise), -O(eat)\}$$

and

$$\{act(exercise), act(eat), -Diamond(exercise), -Do(exercise), Do(eat), \\ Happens(eat), -F(exercise), -F(eat), -O(exercise), -O(eat)\}$$

The difference in the two answer sets simply lies in whether or not the agent does eat more, which is a valid option, as there is no prohibtion on that action. If one wanted the agent to not eat more, should the agent not exercise, one could add a fitting constraint.

A problem that was mentioned was the inability of SDL to distinguish between obligations for which fulfilling a part of the obligation makes sense and obligations for which it does not. If satisfying a part of the obligation does make sense the encoding is quite simple as one can encode both of the obligations separately, as in the previous encodings. If necessary the level and weight of the weak constraint can be adjusted. However, encoding the other kind of obligations proves to be a bit more tricky.

Since a weak constraint only tries to stop a specific combination of predicates from appearing in an answer set, these kinds of constraints could be encoded in multiple weak constraints penalizing each of the cases to violate the obligation. Assume that the obligation is a conjunction of n different actions, where violating any of them renders the rest of them moot. Then there are 2^n possible ways to act (for each of the

actions we can either take it or not). Since the only possible way to fulfil the obligation is to take all of the actions, it would require $2^n - 1$ weak constraints in order to model this obligation. This exponential growth could prove problematic in future work.

However, there is also an alternative way to encode this problem, which leads to equivalent results using less weak constraints. The idea of this approach is to introduce a auxiliary predicate, which is only active when all obligations are active. Then the weak constraints penalise solutions in which this predicate is not active.

This leads to this rephrasing of *Broome's Counterexample*, where both parts need to be satisfied in order to make sense:

> It is obligatory to exercise and eat more (in order to live a healthier life).

This can be encoded by adding the following code to the common core:

$$
\begin{array}{l}
:\sim -O(eat), -O(exercise).[1:2] \\
:\sim O(eat), -O(exercise).[1:2] \\
:\sim -O(eat), O(exercise).[1:2] \\
act(exercise). \\
act(eat).
\end{array}
$$

Here one can see a case of $2^2 - 1$ weak constraints being used in order to encode an obligation that is a conjunction of n different actions.

Alternatively, it can be encoded in the following way by introducing an auxiliary predicate *Health*:

$$
\begin{array}{l}
:\sim not\ \ Health.[1:2] \\
Health :- O(eat), O(exercise). \\
act(exercise). \\
act(eat).
\end{array}
$$

Here, default negation is used in order to penalise answer sets which do not contain both the obligation to eat and the obligation to exercise. Note that such auxiliary predicates can also be used to model exceptions in which certain obligations do not hold, which can be seen in the encoding of the *Asparagus Paradox*.

These programs then fulfill our desiderata. Should no other facts be specified, the programs deduce the obligation to eat and exercise in the following answer set:

$$\{act(eat), act(exercise), Do(eat), Do(exercise), Happens(eat), Happens(exercise), \\ -F(eat), -F(exercise), O(eat), O(exercise), Health\}$$

Note that the answer set for the first way of encoding does not contain the auxiliary predicate *Health*, but is otherwise identical.

Should one of the actions not be possible to take, e.g., by specifying $-Do(eat)$, then the other obligation is not deduced either leading to the following two answer sets for the two programs:

$$\{act(eat), act(exercise), -Diamond(eat), -Do(eat), -Do(exercise), \\ -Diamond(exercise), -F(eat), -F(exercise), -O(eat), -O(exercise)\}$$

and

$$\{act(eat), act(exercise), -Diamond(eat), -Do(eat), Do(exercise), \\ Happens(exercise), -F(eat), -F(exercise), -O(eat), -O(exercise)\}$$

These two answer sets only differ in whether the agent chooses to exercise, although he is unable to eat. As no constraint specifies that the agent should not exercise when not eating, the two answer sets are once again considered to be equally good solutions given the constraints.

3.9 Chisholm's Contrary-to-Duty Paradox

Having decided on how to encode derived obligations, the next step is to check how to encode contrary-to-duty (CTD) obligations. Although these CTDs may seem to be an entirely different kind of derived obligations, they can be handled in the same way as other derived obligations. The only difference to the previous case is that these obligations arise due to an obligatory action not being taken (unless a prohibition is violated) rather than due to an action being taken.

The most famous example showing SDL's inability to handle CTD obligations is probably

Chisholm's Contrary-to-Duty Paradox:

> It ought to be that Jones goes to the assistance of his neighbors.
> It ought to be that if Jones goes to the assistance of his neighbors,
> then he tells them he is coming.
> If Jones doesn't go to the assistance of his neighbours,
> then he ought not tell them he is coming.
> Jones does not go to their assistance.

This paradox is encoded by adding the following code to the common core:

$$
\begin{array}{l}
:\sim -O(assist).[1:2] \\
:\sim Do(assist), -O(tell).[1:2] \\
:\sim -Do(assist), -F(tell).[1:2] \\
-Do(assist) :- . \\
act(assist). \\
act(tell).
\end{array}
$$

The encoding is rather simple with the first three weak constraints encoding the first three sentences and the final rule specifying the fact that the agent does not go to the assistance of its neighbors. Finally, the last two facts specify that the agent is to reason about whether to assist and whether to tell.

Using this encoding only one answer set is returned:

$$
\begin{aligned}
\{&act(assist), act(tell), -Diamond(assist), -Do(assist), -Do(tell), \\
&-Diamond(tell), -F(assist), F(tell), -O(assist), -O(tell)\}
\end{aligned}
$$

Same as with using common sense reasoning only the obligation to not tell the neighbors is derived. The obligation to assist is not derived as it cannot be fulfilled, as the agent has already chosen not to assist. Furthermore the obligation to tell is not derived as the agent does not assist. If one were to remove the fact that the agent chooses to not assist, then the single answer set would derive the obligation to assist and the obligation to tell, thereby not violating any second level constraints.

3.10 Forrester's Paradox

Next, we encode another famous CTD paradox, *Forrester's Paradox*:

> Smith ought not kill Jones.
> If Smith will kill Jones, then Smith ought to kill Jones gently.
> Smith will kill Jones.

This leads to the following encoding using the same recipe:

$$
\begin{array}{l}
:\sim -F(kill).[1:2] \\
:\sim Do(kill), -O(gently(kill)).[1:2] \\
Do(kill). \\
act(kill). \\
act(gently(kill)). \\
Do(kill) :- Do(gently(kill)).
\end{array}
$$

Note that the final rule encodes the connection between killing gently and killing, as killing someone gently implies killing that person. Just as expected only one answer set is generated, which only derives the obligation to kill gently:

$$
\{act(kill), act(gently(kill)), Do(kill), Happens(kill), Do(gently(kill)), \\
Happens(gently(kill)), -F(kill), -F(gently(kill)), -O(kill), O(gently(kill))\}
$$

3.11 Considerate Assassin Paradox

The *Considerate Assassin Paradox* combines CTD paradoxes with priorities among paradoxes:

> You should not kill the witness. (1)
> If you kill the witness, you should offer him a cigarette. (2)
> You should not offer a cigarette. (3)
> You kill the witness. (4)

We encode this by adding the following code to the common core:

$$
\begin{array}{l}
:\sim -F(kill).[1:2] \\
:\sim Do(kill), -O(offer).[1:3] \\
:\sim -F(offer).[1:2] \\
Do(kill). \\
act(kill). \\
act(offer).
\end{array}
$$

There are multiple ways of setting the levels for the weak constraints in this paradox. One can argue that the agent values the mafia code higher than general ethics and therefore put the first two weak constraints at a higher level. In this case we put the second weak constraint at the third level, while setting the other two weak constraints at the second level. This is due to the fact that the first prohibition is not influenced by any of the other obligations or prohibitions. The second weak constraint specifying that the agent is supposed to offer the witness a cigarette, should he kill him is set at a higher level than the prohibition stopping the agent from offering a cigarette, as in that case the agent should still offer a cigarette (although it should not be done under more general circumstances).

This encoding leads to only a single answer set, deriving the obligation to offer a cigarette:

$$
\begin{array}{c}
\{act(kill), act(offer), Do(kill), Happens(kill), Do(offer), \\
Happens(offer), -F(kill), -F(offer), -O(kill), O(offer)\}
\end{array}
$$

3.12 Asparagus Paradox

The *Asparagus Paradox* shows an obligation that has an exception:

Don't eat with your fingers. (1)

If you are served cold asparagus, eat it with your fingers. (2)

You are served cold asparagus. (3)

We first encode this paradox using this formulation.

$$
\begin{array}{l}
:\sim -F(fingers).[1:2] \\
:\sim Served(asparagus), -O(finger).[1:3] \\
Served(asparagus). \\
act(fingers).
\end{array}
$$

Once again, the second weak constraint is set at a higher level, as the obligation to eat with your fingers, should you be served cold asparagus, is more important than the

prohibition to not eat with your fingers.

Only one answer set is returned for this program:

$\{act(fingers), Served(asparagus), Do(fingers), Happens(fingers), -F(fingers), O(fingers)\}$

As one would expect, the obligation to eat with fingers is derived.

The paradox could also be interpreted in the following way:

Don't eat with your fingers, unless you are served cold asparagus. (1)
You are served cold asparagus. (2)

This can be interpreted as a prohibition on eating with your fingers unless one is served asparagus. Meaning that being served asparagus does not obligate the agent to eat with his fingers. This can be encoded in the following way:

$:\sim -F(fingers),\ not\ Served(asparagus).[1:2]$
$Served(asparagus).$
$act(fingers).$

For this encoding two answer sets are derived:

$\{act(fingers), Served(asparagus), -Do(fingers), -Diamond(fingers), -F(fingers), -O(fingers)\}$

and

$\{act(fingers), Served(asparagus), Do(fingers), Happens(fingers), -F(fingers), -O(fingers)\}$

Neither of the answer sets derive any obligations or prohibitions. The difference between the two answer sets is only whether the agent decides to eat with his fingers. (As both options are compliant with the imposed rules.)

This rephrasing nicely shows how an exception to an obligation can be handled using default negation.

3.13 Fence Paradox

The *Fence Paradox* is an interesting paradox as it combines a CTD obligation with an exception to said obligation. The paradox takes the following form:

There must be no fence. (1)
If there is a fence then it must be a white fence. (2)
If the cottage is by the sea, there may be a fence. (3)

One might think that a contrary-to-duty obligation could be handled like an exception to an obligation. While one could formulate the contrary-to-duty obligation as "There may be a fence if it is white", it would not have the same meaning as in the paradox. Handling a contrary-to-duty obligation as an exception leads to losing the original obligation to a certain degree. A contrary-to-duty obligation could in this case be seen as the least thing to do to set things right. Although the fence being white does better the situation the fence itself should still not be there [PS96]. A similar example would be if one were to forget to wish their friend a happy birthday. In such a case one should still congratulate their friend a few days later, although congratulating on their actual birthday would have been better.
The important fact to consider is that should the cottage be by the sea then (as obligation (1) is not active due to the exception in (3)) the fence does not necessarily need to be white. This will be encoded by adding the following code to the common core:

$$
\begin{array}{l}
:\sim -F(have\_fence),\ \ not\ \ Location(sea).[1:2] \\
:\sim Do(have\_fence),\ \ -O(have\_white\_fence),\ \ not\ \ Location(sea).[1:2] \\
act(have\_fence). \\
act(have\_white\_fence).
\end{array}
$$

The exception is encoded in the same way as in the *Asparagus Paradox*. The new part of this encoding is that the exception is also part of the contrary-to-duty obligation as can be seen above. This needs to be done as the fence has to be white only when the cottage is not by the sea.

In order to check whether the obligation for the fence to be white is deduced when the cottage is by the sea the following facts are added to the common core:

$$
\begin{array}{l}
Location(sea). \\
Do(have\_fence).
\end{array}
$$

These facts specify that the cottage is by the sea, and furthermore that there is a fence. Then two answer sets are derived:

$$\{act(have\_fence), act(have\_white\_fence), Location(sea), Do(have\_fence), \\ Happens(have\_fence), -Do(have\_white\_fence), \\ -Diamond(have\_white\_fence), -F(have\_fence), \\ -F(have\_white\_fence), -O(have\_fence), -O(have\_white\_fence)\}$$

and

$$\{act(have\_fence), act(have\_white\_fence), Location(sea), Do(have\_fence), \\ Happens(have\_fence), Do(have\_white\_fence), Happens(have\_white\_fence), \\ -F(have\_fence), -F(have\_white\_fence), -O(have\_fence), \\ -O(have\_white\_fence)\}$$

As required, neither answer set derives the obligation for the fence to be white.

3.14 Alternative Service Paradox

The *Alternative Service Paradox* poses the question on how to encode disjunctions. As mentioned in the previous section, including disjunctive distributivity would not solve this problem as this leads to a whole separate set of issues. While disjunctive distributivity is a problem in SDL, it is not when using ASP. Consider the following weak constraint:

$$:\sim -O(a), -O(b).[1:2]$$

Due to answer sets being interpreted as different ways of optimally satisfying the given obligation, this weak constraint is semantically equivalent to a disjunction over the obligation to take action a and the obligation to take action b. When added to the common core, this leads to two answer sets:

One in which $O(a)$ is derived and one in which $O(b)$ is derived. This is however different to adding disjunctive distributivity to SDL as neither of the obligations is actually derived. The agent is simply given two possible answer sets modelling possible ways to optimally comply with the given constraints. Similar to real life where one could either choose to satisfy the obligation by taking action a or satisfying the obligation by taking action b, the agent is given two answer sets that model each of these options.

Using the above way to model obligations over disjunctions, we can encode the *Alternative Service Paradox*:

> You are obligated to fight in the army or perform alternative service.
>
> You are obligated to not fight in the army.

The encoding is done by adding the following to the common core:

$$\boxed{\begin{array}{l} :\sim -O(fight), -O(alt).[1:2] \\ :\sim -F(fight).[1:2] \\ act(fight). \\ act(alt). \end{array}}$$

This program leads to a single answer set:

$$\{act(fight), act(alt), -Do(fight), Do(alt), Happens(alt), \\ -Diamond(fight), F(fight), -F(alt), -O(fight), O(alt)\}$$

As one can see, two obligations/prohibitions are in the answer set. Firstly, the obligation to perform alternative service and secondly the prohibition on fighting. This once again derives the same obligations as common sense reasoning would.

3.15 The Logical Necessity of Obligations

The logical necessity of obligations is a paradox, which does not pose a problem for encodings in DLV. As all the programs specify which actions to reason about, there is no way for the logic program to deduce any obligations unless it is specifically requested. Only using the core would be an example of a program, which does not deduce any obligation.

One might argue that the weak constraint $:\sim O(X).[1:1]$ in the core is an encoding of the statement: "Nothing is obligatory.". Although a valid obligation arising would render this moot. Mirroring the real world this constraint is only relevant until the situation changes and another stronger obligation arises which renders the statement invalid.

Final Remarks
Using weak constraints, all considered paradoxes could be encoded in a way, such that the answer sets satisfied all the desiderata posed by common sense reasoning. This was done using consistent formalisms throughout. It is important to note that the encoded scenarios showcased necessary parts of encoding a normative system. The next chapter will generalise them in order to generate a way to encode normative systems in DLV.

3.16 Related Work

Being a topic of great interest multiple approaches to handling normative systems have been proposed. Many of the approaches related to the multi-agent systems community can be seen in [ADL18]. We will discuss below the approaches most similiar to our work.

One of the earlier works on encoding normative systems is [SSK+86]. There Sergot et al. encoded the British nationality act using logic programming. Their goal was to show that logic programming was capable of representing the complexities of statutory law. Due to the rules all being restricted to definite Horn clauses, there were some syntactic restrictions such as not allowing disjunctions in the head of the rule. Using these tools they aim to determine whether the British nationality act applies to certain individuals. Although [SSK+86] does not use deontic logic there has been a lot of research on deontic logic in law, e.g., [JS92] or the work of Governatori, Rotolo and Sartor in [GHP+21].

[SBD+00] is one of the earliest works on reasoning about obligations and prohibitions of agents. Subrahmanian et al. introduced semantics (as well as syntax) for this task. In addition to the operators O, Do and F used in this thesis, their work featured operators P and W, which classified actions as permissible resp. waived obligations to take an action. Although multiple semantics (e.g., Feasible status sets, Rational status sets...) were presented in their work, the operators behave differently in most semantics compared to our semantics. In their semantics it is possible for obligations to be violated. In other words it is possible for $O(\alpha)$ to be in a status set for an action α but not $Do(\alpha)$. Furthermore, an agent does not choose to take an action unless there is a justification for taking that action. In our semantics we "guess" what actions to take, thereby generating multiple answer sets that differ only on actions which are not necessarily forbidden or obligated. This "guessing" is not present in their work. The operator P acts as strong permission, when using the open world assumption (where everything not strictly forbidden is permissible), thereby waiving prohibitions which would otherwise be in place. Similarly, the operator W waives obligations which would otherwise be in place. Note that the operators P and W can be avoided by a different encoding and were added for user convenience. In this master thesis exceptions to obligations and prohibitions are encoded as part of the weak constraints generating them and any action that is not strictly forbidden is permissible. Therefore, we do not have a need for operators P and W, as an obligation resp. a prohibition would only be waived due to an exception (that would be encoded as part of weak constraints) or a conflicting obligation resp. prohibition (that would be encoded in seperate weak constraints) that is of equal or higher priority. Although also referring to deontic logic the proposed way of dealing with conflicting obligations in [SBD+00] is to satisfy a maximal subset of obligations. This can be seen as a weakness of this approach in case a clear preference among obligations exists. An example of such is *Plato's Dilemma*, introduced in Section 2.3.3. In this paradox the obligation to rush a child to the hospital conflicts with the obligation to meet a friend for dinner. In this case one needs the agent to fulfill the most important obligation of helping

the child. This would however not be possible with the framework introduced in [SBD+00].

In [KS18] Kowalski and Satoh utilised abductive logic programs to encode the notion of obligation and many paradoxes. Although there are some similarities to our work, there are also notable differences. An obvious difference is the choice of tools. Since in abductive logic programming integrity constraints are hard constraints, their obligations (which in general can be violated) are formulated as hard constraints. They achieved this by using the Andersonian reduction [And58] that reformulates an obligation $O(x)$, to take an action x, as $x \vee sanction$. Thereby they turn the obligation into a hard constraint by giving the choice between fulfilling the obligation or being sanctioned. Rather than trying to derive all optimal ways of fulfilling given obligations, [KS18] focuses on finding a best model that satisfies given goals (that must be satisfied). Optimality of a given model is not part of the abductive logic program. They provide a partial ordering of the models through which it is determined whether a model is part of the best models. This is for example done by preferring models which do not contain certain sanctions over others. In our encoding the ordering of the models is part of the ASP program itself. The paradoxes they consider are resolved in the same way as here, however the authors did analyse only a subset of the paradoxes considered in this work. Similar to this work, the encodings of the paradoxes are applicable to more general cases. Our work also presents a more general way of encoding the different types of obligations as well as a methodology for encoding systems in general. As they only encode a subset of the paradoxes encoded in our work, it is unclear whether their approach would work for all paradoxes and case studies considered in our work.

Using a combination of input-output logic and game theoretic methods, van der Torre and Boella encoded the behaviour of agents in a multi agent system under a normative system [BvdT04]. They used the BOID architecture [BDHvdT02] in order to model the agents. This architecture is used to encode agents that are capable of a more human kind of reasoning. Van der Torre and Boella establish different kinds of norms and rules with different properties. In this master thesis a similar differentiation between rules is given by rules resp. weak constraints and their levels. The approach presented in [BvdT04] is more complex than the approach taken in this master thesis, as it is not only capable of having agents take into consideration other agents' probable actions, but furthermore it considers the agent's own goals. This lets the agent violate given norms should the agent believe the violation to be worth the sanctions incurred. While the goals of an agent can theoretically be modeled using weak constraints by simply considering them as obligations the agent imposes upon himself, the game-theoretic aspect of [BvdT04] would be extremely hard if not impossible to encode given the tools presented in this master thesis. The acronym BOID [BDHvdT02] stands for beliefs, obligations, intentions, and desires. The architecture, as the acronym would suggest is used to model the beliefs, obligations, intentions and desires of an agent. There are also different kinds of agents, such as selfish or social agents as examples, influ-

encing how agents act upon the goals generated from the architecture. The idea is to create consistent sets of goals from the beliefs, obligations, intentions and desires of the agent and then rank these consistent sets. Afterwards, the agent is supposed to plan how to fulfill the goals. This kind of reasoning more closely mimics human reasoning.

A type of logic, often seen when reasoning about norms, is Temporal Logic. An advantage given by Temporal Logic approaches is that they are good at enforcing norms that indirectly prohibit certain actions [ABDL15, BDK13]. As an example, while it is obvious to an agent how to satisfy a norm that forbids him from taking a specific action, it is a lot harder for an agent to satisfy a norm that states that an agent sees to it that a certain condition is maintained. Consider the obligation: "See to it that the child stays dry." For such an obligation it does not suffice to not wet the child, but also take actions to prevent the child from getting wet by other means. As such it may require stopping the child from leaving the house if it is about to rain. Like in Modal Logic possible worlds are given where reaching a certain world from another requires taking or refraining from certain actions. Temporal Logic approaches aim to choose obligatory or forbidden actions for the agent by ensuring that the agent taking or refraining from an actions leads to a situation where the agent can once again take an action to ensure the obligation is still satisfied. However, building such a model requires a lot of work and may be hard to build for complex situations.

The approach in [ABDL15] focuses on how and when to restrict an agent's behaviour in multi-agent systems. As mentioned before, a model is built that simulates the states of the system after the agent takes an action. For large universes with lots of worlds it is a large computational effort to check which actions can safely be taken without landing in a world that leads to the obligation being violated regardless of which action is taken. The basic idea is that a guard function gives safe successor states if possible, taking into account the history of system behaviour up until that point. In order for the runtime to stay bounded the look-ahead in states is bounded, with different norms requiring different look-aheads. [BDK13] similarly aims to create monitors that check whether certain runs satisfy specific Linear temporal logic-formulas in order to check compliance.

A combination between Temporal Logics approaches, deontic logics, and Answer Set Programming is given by [GMD13]. In order to better verify the compliance of business processes with norms, Giordano et al. extended temporal logics with deontic modalities calling the resulting logic DDLTL (Deontic Dynamic Linear Time Temporal Logic). This allows for Temporal Logics formulas to appear within deontic operators. Within business process norms, contrary-to-duty obligations take an important role as well. They also work with types of obligations that are not considered in this master thesis. For example, their work considers maintenance obligations (where one has to see to it that a certain state is maintained) and obligations with deadlines (obligations that require one to take an action until a certain point in time, e.g., repay a debt). By combining DDLTL with

ASP they generate answer sets that include all the obligations generated during the run of a business process, given the specification of the business process as well as business rules.

An ASP approach different from this master thesis is given by deontic logic programs, as can be seen in [GA12]. Deontic logic programs, which allow atoms in rules to take the form of complex SDL formulas have found some application. Gonçalves and Alferes [GA12] were able to embed input-output logic, a deontic logic proposed in [MvdT01], into such deontic logic programs. Although they do not encode various deontic paradoxes in their work as they focus only on the embedding of input-output logic, the expressiveness of deontic logic programs (and more specifically their embedding) should allow one to encode the deontic paradoxes in a suitable manner. Deontic logic programs as a tool are however more complicated to use, as they require an understanding of the embedded logic, whereas our presented methodology can be used without a deeper understanding of deontic logics.

The usage of weak constraints in ASP, introduced in [BLR97], has been well studied. Buccafurri et al. also provided complexity results as well as use cases for weak constraints. Additional problems solved using weak constraints in ASP can be found in the DLV user manual [BFI+20]. Note that these works do not deal with deontic logics or obligations, but rather solve general problems such as the minimum vertex cover problem.

This master thesis is set apart from the other works by the simplicity of the method of encoding presented. Although our method of encoding works when encoding scenarios and rather simple normative systems, more complex systems may be hard to encode. Furthermore, the number of considered paradoxes and obligation led to a precise way of encoding the different kinds of obligations and prohibitions.

CHAPTER 4

Generalising the Encodings

We abstract and generalise the encodings of the paradoxes presented in the previous chapter. First, the encodings will be generalised for paradoxes grouped in the same classification and afterwards a method will be given that aims to encode all kinds of paradoxes.

4.1 Paradoxes Centering Around RMD

The paradoxes that center around RMD do not pose problems for the logic programming approach, as the logic program can only use the rules specified in the logic program in order to deduce knowledge. As a reminder, the paradoxes encoded for this class are:

- *Ross's Paradox*
- *Good Samaritan Paradox*
- *Åqvist's Paradox of Epistemic Obligation*

As there is no rule in the common core that would correspond to RMD,

$$\text{If } p \rightarrow q \text{ is a theorem, then } Op \rightarrow Oq \text{ is a theorem.}$$

there are no special methods needed to encode these paradoxes.

In general, obligations are encoded as:

$$:\sim -O(obl).[1:2]$$

Here *obl* represents the action that is obligatory. Note that the weight and level can take different values should the situation require them to. Should an obligation only

situationally arise, the encoding slightly changes.

As an example, should an event happening lead to the rise of an obligation (as can be seen in the *Good Samaritan Paradox*) this can be encoded in the following way:

$$:\sim -O(obl), Happens(event).[1:2]$$

In general the paradoxes centred around RMD are rather simple to encode using auxiliary predicates in order to better reflect the situations shown in the paradoxes.

4.2 Puzzles Centered Around DD and OD

These paradoxes arise due to multiple obligations which cannot be fulfilled simultaneously. In general, we can distinguish two cases for these paradoxes:

1. Satisfying either of the obligations is fine, as there is no priority among the obligations.
2. There is a priority among conflicting obligations.

The two considered paradoxes each fit into one of these cases:

1. *Sartre's Dilemma* has no priority among the obligations.
2. *Plato's Dilemma* has a priority for the given obligations.

Note that most contradictions between actions need to be encoded in DLV. An example of such a contradiction would be both going south and going north. This is done in the following way. Let the mutually exclusive actions be $act1$ and $act2$. Then, this can be encoded in the following way:

$$: -Do(act1), Do(act2).$$

Obvious contradictions, such as an obligation to do something and an obligation not to do something, do not need to be encoded. The paradoxes centered around DD and OD are encoded in the following way:

1. Start by encoding which actions are mutually exclusive, should the obligations not be directly contradictory.
2. Determine the priorities of the obligations and encode them as weak constraints at levels reflecting these priorities.

- Start by encoding the least important obligations at level 2 and then increase the levels at which the obligations are encoded as weak constraints as the importance of obligations increases.
- Obligations of equal importance are encoded at the same level.
- If necessary, add any preconditions for obligations arising (such as the emergency in *Plato's Dilemma*).

4.3 Puzzles Centered Around Deontic Conditionals

This class of paradoxes, which is the largest one considered in this work, is the most interesting. The paradoxes in this class are indeed quite diverse. As is often the case with obligations in real life, the obligations in this class are only active under certain circumstances. The paradoxes encoded for this class are:

- *Broome's Counterexample*
- *Chisholm's Contrary-to-Duty Paradox*
- *Forrester's Paradox*
- *Considerate Assassin Paradox*
- *Asparagus Paradox*
- *Fence Paradox*
- *Alternative Service Paradox*

We start by considering the paradox of derived obligation. In this paradox, the possible ways of representing a derived obligation are considered. It was shown that either of the representations leads to nonsensical claims in SDL. The following sentence is to be encoded:

Promising to meet Bob commits you to meeting him.

The two proposed ways to encode this are:

$$O(p \to m) \tag{1}$$
$$p \to O(m) \tag{2}$$

While encoding the first is theoretically possible, it requires a lot of additional work compared to the second option. The first option can be encoded by introducing a new constant to represent the formula. The above sentence can then be encoded in the following way:

$$
\begin{array}{l}
:\sim -O(q).[1:2] \\
-Do(p) \vee Do(m) :- Do(q). \\
Do(q) :- -Do(p). \\
Do(q) :- Do(m). \\
act(q). \\
act(p). \\
act(m).
\end{array}
$$

The three rules specify the relation between fulfilling the more complex formula q representing $p \rightarrow m$ and fulfilling the parts p and q. Here q is used as an auxiliary predicate, as DLV does not allow for disjunction inside of operators.

Note that q (and therefore also $p \rightarrow m$) is considered as an action. While one may argue that $p \rightarrow m$ does not represent an action, one can rephrase it as $\neg p \vee m$. This can be understood as an action that can be taken in two different ways. Either by not doing p or by doing m.

Considering the added complexity, we chose option (2). As an example consider the obligation *obl* that arises due to the agent taking an action *action*, as in *Broome's Counterexample*. This can be encoded in the following way (once again leaving open the possibility of changing the weight and level of the weak constraint):

$$:\sim Do(p), -O(m).[1:2]$$

Note that contrary-to-duty obligations are encoded in the same way. In case of a prohibition on an action p the encoding looks the same as above. The case of an obligation arising due to another obligation being violated, p as an example, can be encoded in the following way:

$$
\begin{array}{l}
:\sim -O(p) \\
:\sim -Do(p), -O(m).[1:2]
\end{array}
$$

Exceptions to obligations are handled in a similar fashion, as can be seen in the *Asparagus Paradox*. Here, usually an auxiliary predicate is introduced that models the exception. Note that this auxiliary predicate can also be derived in a seperate rule if the exception depends on multiple factors. Assuming the auxiliary predicate is called *Exception* such a case can then be encoded in the following way:

$$:\sim -O(m), not\ \ Exception.[1:2]$$

Note that in case there exists a contrary-to-duty obligation for an obligation with an exception, *not Exception* needs to be added to the contrary-to-duty obligation as well,

as in the case of the exception the obligation is not violated.

Auxiliary predicates are used in order to model obligations over conjunctions that only make sense to be satisfied in their entirety. This is encoded in the following way using $Conjunction$ as the auxiliary predicate and $obl1, \ldots, obln$ as the actions in the conjunction:

$$:\sim not\ Conjunction.[1:2]$$
$$Conjunction :- O(obl1), \ldots, O(obln).$$

Note that once again weight and level of the weak constraint can vary.

In the *Alternative Service Paradox* obligations over disjunctions are considered. Given two possible ways of satisfying the obligation a or b, it was encoded in the following way:

$$:\sim -O(a), -O(b).[1:2]$$

There is also an alternative way of encoding disjunctions. One could indeed use an auxiliary action c describing the disjunction in the following way:

$$Do(c) : -Do(a).$$
$$Do(c) : -Do(b).$$
$$Do(a) \vee Do(b) : -Do(c).$$

Afterwards, the obligation to do c can be added in the usual way (after denoting a, b, c as actions). Note that this encoding does not return answer sets in which the agent chooses to take both actions (if no additional rules are given). We chose the first method of encoding for multiple reasons. First, there are cases in which the agent may want to take both actions although only one of the actions is obligatory. Furthermore, the first encoding is simpler and fits with the semantics established in Section 3.2.

Although the puzzles centered around deontic conditionals are similar in nature, they present multiple different kinds of obligations. These different types of obligations (such as obligations over conjunctions or derived obligations) require different methods for encoding. Due to the large number of paradoxes and obligations considered in this section, we sometimes had to choose between multiple possible methods for encoding the same kind of obligation. All in all, the puzzles centered around deontic conditionals are the most challenging and interesting classification among those we considered in this master thesis.

4.4 General Encoding

Here we abstract and combine the methods of encoding introduced in the previous sections in order to encode normative systems containing multiple different kinds of

obligations. The paradoxes not only show cases which SDL is incapable of handling, but also show intricacies which an encoding of a normative system should be able to handle. Note that the weights and levels in the examples are placeholders and can take different values in practice. The intricacies are presented in the following subsections.

4.4.1 Conditional Obligations

These obligations arise due to a condition being met. This condition could for example be an event taking place. Such obligations can be seen in one version of *Broome's Counterexample*. Assuming that the event that leads to the obligation is outside the control of the agent it can be formulated in the following way:

$$:\sim Happens(event), -O(obl).[1:2]$$

The event being outside of the agent's control is encoded through the predicate $Happens$. Events that are in the agent's control (such as the agent taking an action) would be encoded using the predicate Do (as an example $Do(event)$). Note that $event$ and obl are placeholders denoting the event and the obligation, respectively.

4.4.2 Obligations Over Disjunctions

Obligations over disjunctions are obligations which are satisfied by satisfying any of the actions in a disjunction. The *alternate service paradox* showcases such an obligation. Assume $obl1, \ldots, obln$ are the actions in the disjunction. This can be simply encoded in the following way:

$$:\sim -O(obl1), -O(obl2), \ldots, -O(obln).[1:2]$$

4.4.3 Conjunctions of Obligations That All Need To Be Satisfied

These obligations are only satisfied when all parts of the obligation are satisfied. For this type of obligations satisfying only part of a conjunction is not preferable to satisfying none. An example for such an obligation can be seen in the rephrasing of *Broome's Counterexample*. Two possible ways of encoding these obligations were presented. The shorter way involves an auxiliary predicate. Assume the latter is $Conj$ and $obl1, \ldots, obln$ are the actions in the conjunction. The encoding could then take the following form:

$$:\sim not\ \ Conj.[1:2]$$
$$Conj :- O(obl1), \ldots, O(obln).$$

Here $Conj$ is the auxiliary predicate. Note that if satisfying parts of the conjunction can be seen as preferable over not satisfying any part, the individual obligations are encoded separately.

4.4.4 Obligations With Exceptions

Often obligations do not hold in certain circumstances. A common example of such an obligation that is often encountered is an exception to a no-parking zone during certain times. The *Asparagus Paradox* shows an exception to an obligation under social norms. Exceptions can be modelled using auxiliary predicates. Using such an auxiliary predicate, called *Exception* in the next example, such an obligation can be encoded in the following way:

$$:\sim -O(obl),\ \ not\ \ Exception.[1:2]$$

4.4.5 Contrary-to-Duty Obligations

Contrary-to-duty obligations arise due to another obligation not being fulfilled. An example of such obligations can be seen, e.g., in *Chisholm's Contrary-to-Duty Paradox*. Note that these obligations can be considered as a special case of conditional obligations. In the encodings, they are handled in the following way:

$$:\sim -O(obl1).[1:2]$$
$$:\sim -Do(obl1), -O(obl2).[1:2]$$

Note that $obl1$ and $obl2$ refer to obligatory actions. The second weak constraint is only of relevance, should the agent not take the obligatory action (or choose not to).
In case of the violated obligation having an exception, the latter must be encoded as part of the contrary-to-duty obligation. This could take the following form, where e is an auxiliary predicate that is active when the exception is given:

$$:\sim -O(obl1),\ \ not\ \ e.[1:2]$$
$$:\sim -Do(obl1), -O(obl2),\ \ not\ \ e.[1:2]$$

4.4.6 Conflicting Obligations and Prioritization Among Those

It is often the case that we are subject to various obligations that are not satisfiable at the same time. There are multiple ways of handling such situations. Most important obligations are either weighted more heavily or generated at a higher level. Suppose $O(obl1)$ to be the more important obligation and $O(obl2)$ the less important one. This can be formulated in the following way:

$$:\sim -O(obl1).[1:3]$$
$$:\sim -O(obl2).[1:2]$$
$$:- Do(obl1), Do(obl2).$$

or alternatively:

$$:\sim -O(obl1).[2:2]$$
$$:\sim -O(obl2).[1:2]$$
$$:- Do(obl1), Do(obl2).$$

Depending on the system one is trying to encode either approach may be preferable. For the second approach the weights need to be well chosen. Consider three obligations $obl1, obl2$ and $obl3$ such that $obl3$ is the most important and incompatible with either of the two. When simply choosing weights in ascending order this can be modelled by adding the following code to the common core:

$$:\sim -O(obl3).[3:2]$$
$$:\sim -O(obl2).[2:2]$$
$$:\sim -O(obl1).[1:2]$$
$$:- Do(obl1), Do(obl3).$$
$$:- Do(obl2), Do(obl3).$$

If one runs this code there would be two possible combinations of obligations given. Either $obl3$ is obligatory or $obl1$ and $obl2$ are obligatory, due to the weights of the violated weak constraints being the same in this case. Depending on the normative system that is being encoded this may be undesirable. So choosing the method for encoding conflicting obligations depends on whether there is a directly preferable obligation or fulfilling multiple obligations may be equally or more preferable.
Note that in our methodology all encoded obligations are comparable. Therefore, our methodology is limited to encoding only normative systems for which answer sets are always comparable. As mentioned earlier, [BDRS15] allows for encodings where obligations respectively answer sets may be incomparable.

4.4.7 Methodology

Note that an obligation may belong to multiple classes of the same time. In general, the encodings of such cases can be derived from the encodings of the classes the obligation belongs to. As an example, a contrary-to-duty obligation may have an exception (as can be seen in the methodology for encoding contrary-to-duty obligations) or consist of an obligation over a disjunction. This leads to the following general way of encoding obligations:

1. Determine the types of obligations and their importance.
2. Determine which actions are incompatible at the same time.
3. Encode the different kinds of obligations and their importance (through weight or level of the weak constraint) in the way described above. (Knowing which actions are conflicting makes it easier to determine the importance of the specific obligations.)
4. Encode the exclusion of previously determined incompatible combinations of actions.
5. Encode additional information, e.g., dependencies between actions, such as one action requiring another action, or what action to reason about. Examples of this will be given in the next section.

4.5 A Case Study

To test the method of encoding described above, we design a normative system that aims to incorporate all the intricacies mentioned in the previous section.

A topic that is currently of great interest is self-driving cars. Inspired by this topic is the following normative system that presents obligations that hold while driving:

1. It is obligatory to stop if the traffic light is red.
2. It is obligatory to not impede the flow of traffic (by stopping), unless it is to let a car merge.
3. It is obligatory to move out of the way when an ambulance approaches.
4. If you drive during winter it is obligatory to either have winter tires or all-season tires.
5. It is obligatory to not cause any damage.
6. It is obligatory to have your drivers license and vehicle registration with you, unless it was stolen and you have proof (of theft). (Only having one is punished the same as having none, as the police has to do the same administrative work.)
7. If one causes damage, it is obligatory to drive directly to the next police station to make a damage report.
8. It is obligatory to give first-aid, when seeing a medical emergency.

We encode the above normative system using our method. We will go through the methodology step by step.

Step 1:

We start by categorising the obligations.

The first obligation is a derived obligation. The obligation to stop is derived when the traffic light is red.

The second is an obligation with an exception which is to let a car merge. Note that a car wanting to merge does not necessitate the car stopping, but it does allow the car to stop should the agent want to.

The third obligation is once again an obligation that is derived similarly to the first one.

The fourth obligation is both a derived obligation and a disjunctive obligation. Should the antecedent be fulfilled one part of the obligation must be satisfied. Note that in this case it is not possible for both parts of the obligation to be fulfilled (as one cannot have winter tires and all-season tires at the same time).

The fifth obligation is a regular obligation, with no additional properties.

The sixth obligation consists of a conjunction of obligations that all need to be satisfied with an exception. This showcases an example of an obligation that belongs to multiple categories.

The seventh obligation is a CTD obligation, that is active when violating the fifth obligation.

The eighth obligation is once again a derived obligation.

Step 2:

Next, we look at what combination of obligations cannot be fulfilled at the same time. Although in this case the obligations are only incompatible in pairs theoretically it can be possible that a combination of more than two actions is incompatible although any two actions would be compatible:

- The first two obligations cannot be fulfilled at the same time, as a red traffic light would commit one to stopping although it is not to let a car merge. The second statement can be seen as non-contradictory by arguing that one does not impede the flow of traffic by stopping when the traffic light is red. However, for our encoding we will consider the two actions contradictory. We want the agent to derive the obligation to stop.
- The first and third obligation are incompatible, as moving out of the way requires movement that is obviously contradictory to stopping. Here the agent should move out of the way as letting the ambulance pass is of utmost importance.
- The second and eighth obligation are incompatible, as giving first aid requires stopping the car. Here the obligation to give first aid should be prioritised.
- For the same reason the third and eighth obligation are incompatible. In this case moving out of the way should be prioritised as the ambulance is more qualified to help in a medical emergency (as they have trained personnel and medical equipment).

- The seventh and eighth obligation are contradictory, as one cannot drive directly to the next police station and at the same time stop and give first aid. Once again, stopping to give first aid should be deduced by the agent.

As an intermediate step we now list the results from above in a short fashion. In the following list O_i refers to the i-th obligation. For example, O_1 refers to: "It is obligatory to stop if the traffic light is red." Summarizing the previous statements we obtain the following preferences, where $O_i \succ O_j$ means that O_i is preferred over O_j:

- $O_1 \succ O_2$
- $O_3 \succ O_1$
- $O_8 \succ O_2$
- $O_3 \succ O_8$
- $O_8 \succ O_7$

Step 3:

Using the above conflicts and priorities, we can derive the following weights and levels for the weak constraints corresponding to the obligations:

1. It is obligatory to stop if the traffic light is red. [1:3]
2. It is obligatory to not impede the flow of traffic (by stopping), unless it is to let a car merge. [1:2]
3. It is obligatory to move out of the way when an ambulance approaches. [1:4]
4. If you drive during winter it is obligatory to either have winter tires or all-season tires. [1:2]
5. It is obligatory to not cause any damage. [1:2]
6. It is obligatory to have your drivers license and vehicle registration with you, unless it was stolen and you have proof (of theft). [1:2]
7. If one causes damage, it is obligatory to drive directly to the next police station to make a damage report. [1:2]
8. It is obligatory to give first-aid, when seeing a medical emergency. [1:3]

Note that an obligations is always placed on the lowest level if it cannot be in conflict with another obligation.

Now we look at the predicates used in the encoding of the above system. Note that once again the common core is used and as such all predicates mentioned in Section 3.1 are used as well. The used predicates are:

- *Redlight* denotes that a red traffic light is active in front of the agent.
- *Winter* denotes the season being winter.
- *Theft* denotes that the agent is in possession of proof of theft of his drivers license and/or registration.
- *Licenses* is an auxiliary predicate used in the formulation of constraint 6. It is used in the way described in Subsection 4.4.3.

The constants used in the encoding are:

- *stop* is an action to be reasoned about. *Do(stop)* denotes the car stopping.
- *merge* is an event that can happen. *Happens(merge)* denotes that a car is trying to merge onto the lane that the agent is on.
- *emergency_vehicle* is an event that can happen. *Happens(emergency_vehicle)* denotes an ambulance (with active emergency lights) approaching the car.
- *move* is an action to be reasoned about. *Do(move)* denotes that the car needs to move out of the way.
- *equip_winter* is an action clarifying that the car is equipped with winter tires. Note that although it can be argued that it is not an action in the narrow sense (as having tires equipped is more of a state than an action) it will be considered an action in the scope of this example.
- Similarly, *equip_allseason* clarifies that the car is equipped with all-season tires.
- *damage* is an action to be reasoned about. *Do(damage)* denotes the agent causing damage.
- *carry_license* is an action to be reasoned about. *Do(carry_license)* corresponds to the agent having his drivers license with him.
- *carry_registration* is an action to be reasoned about. *Do(carry_registration)* corresponds to the agent having his registration with him.

- *drive_police* is an action to be reasoned about. *Do(drive_police)* denotes the agent driving to the next police station without any detours.
- *medical_emergency* is an event that can happen. *Happens(medical)* denotes a medical emergency happening in the vicinity of the agent.
- *give_first_aid* is an action to be reasoned about. *Do(give_first_aid)* denotes the agent giving first aid.

Finally, we can encode our example. We do this by adding the following lines to the common core.

First, we encode the obligations themselves:

$$
\begin{array}{l}
:\sim Redlight, -O(stop).[1:3] \\
:\sim not\ \ Happens(merge), -F(stop).[1:2] \\
:\sim Happens(emergency\_vehicle), -O(move).[1:4] \\
:\sim Winter, -O(equip\_allseason), -O(equip\_winter).[1:2] \\
:\sim -F(damage).[1:2] \\
Licenses :- O(carry\_license), O(carry\_registration). \\
:\sim not\ \ Licenses, not\ \ Theft.[1:2] \\
:\sim Happens(damage), -O(drive\_police).[1:2] \\
:\sim Happens(medical\_emergency), -O(give\_first\_aid).[1:3]
\end{array}
$$

The encoding of the obligations is done as described in Section 4.4. There are however two obligations that are combinations of different kinds of obligations.

The fourth obligation is a combination of a derived obligation and a disjunctive obligation. As such, we are able to encode it in the following way:

$$\sim Winter, -O(equip\_allseason), -O(equip\_winter).[1:2]$$

If *Winter* is not in the answer set, this weak constraint cannot be violated. Therefore, this weak constraint can only be violated when *Winter* is true. For answer sets where *Winter* is true this weak constraint is violated by the same answer sets that violate the following weak constraint:

$$\sim -O(equip\_allseason), -O(equip\_winter).[1:2]$$

This is the common way of encoding disjunctive obligations. Therefore, the weak constraint can be understood as deriving the disjunctive obligation only when winter is true,

thereby encoding the combination of a derived obligation with disjunction.

The sixth obligation is a mixture of an obligation over a conjunction of actions and an obligation with an exception. We once again use an auxiliary predicate to encode the conjunction of predicates as usual:

$$Licenses :- O(carry\_license), O(carry\_registration).$$

Using this the exception is then encoded the same way as usual:

$$:\sim not\ \ Licenses, not\ \ Theft.[1:2]$$

This captures that we either want the exception to be in the answer set or the obligations to have the license and the registration.

Step 4:

Having encoded the obligations themselves, next the conflicting actions which were determined in Step 2 are encoded:

$$\boxed{\begin{array}{l} :- Do(stop), Do(move). \\ :- Do(drive\_police), Do(give\_first\_aid). \end{array}}$$

Step 5:

Finally, the following information is added to the encoding:

$$\boxed{\begin{array}{l} :- Theft, Do(carry\_license), Do(carry\_registration). \\ Do(stop) :- Do(give\_first\_aid). \\ :- Do(first\_aid), not\ \ Happens(medical\_emergency). \\ act(stop). \\ act(move). \\ act(damage). \\ act(equip\_allseason). \\ act(equip\_winter). \\ act(carry\_license). \\ act(carry\_registration). \\ act(drive\_police). \\ act(give\_first\_aid). \end{array}}$$

The first constraint disallows a proof of theft from existing if the agent has both the license and the registration (if one or more are stolen he cannot be in possession of both). It is also clarified that giving first aid implies stopping the car (as else it would not be possible to give first aid). Furthermore, a constraint prohibits the agent from giving first aid without a medical emergency happening (as this is not possible). Finally, all the actions to be reasoned about are denoted as acts.

We now consider some examples of obligations which are derived in such cases.

Example 1

Assume that an agent is driving during winter after having its driver's license and registration stolen (and having the corresponding confirmation with him). The agent's car is not equipped with all-season tires. Upon driving, the agent comes upon a red light. This information will be denoted in an additional file in the following way:

$$\boxed{\begin{array}{l} Winter. \\ Theft. \\ -Do(equip\_allseason). \\ Redlight. \end{array}}$$

Two answer sets are generated for this. The difference between the two answer sets is simply whether the agent chooses to directly drive to the police station (as this is not forbidden). Both answer sets derive the same obligations. Note that only the derived obligations will be listed due to the large size of the answer sets:

$$F(damage), O(stop), O(equip\_winter)$$

The same two obligations and one prohibition are derived that would also be derived using common sense reasoning. The obligation to stop (due to the red light), the prohibition on damaging cars (that is always active) and the obligation to equip winter tires as the agent is not in possession of all-season tires.

Example 2

Assume that an agent is driving when witnessing a medical emergency. Furthermore, an ambulance is approaching with active emergency lighting and a vehicle is trying to

merge. This case can be encoded in an additional file in the following way:

$$\boxed{\begin{array}{l} Happens(medical\_emergency). \\ Happens(emergency\_vehicle). \\ Happens(merge). \end{array}}$$

Multiple answer sets are derived which differ on unimportant details such as whether the agent chooses to equip winter or all-season tires.

All answer sets however derive the same obligations:

$$F(damage), O(move), O(carry\_license), O(carry\_registration)$$

The obligation to move is derived as letting the ambulance pass is of higher importance than treating the medical emergency. The other obligations are obviously active as the exceptions are not given.

Example 3

Assume that an agent is driving when witnessing a medical emergency after having damaged another car. This test case is encoded in the following way:

$$\boxed{\begin{array}{l} Happens(medical\_emergency). \\ Happens(damage). \end{array}}$$

Once again multiple answer sets with minor differences are derived, as in the previous case. However, as expected all answer sets contain the same obligations:

$$F(damage), O(carry\_license), O(carry\_registration), O(give\_first\_aid)$$

As expected the agent is obligated to give first aid rather than driving directly to the police station.

This more complex example was also handled well by the generalised method of encoding described in this section. Using this method, the next section will work on encoding a normative supervisor for pacman as seen in [NBCG21].

CHAPTER 5

Encoding Ethical Pacman

Neufeld et al. [NBCG21] combined normative reasoning and reinforcement learning agents to accommodate complex situations, such as conflicting obligations or situations where no compliance is possible, trying not to lose the optimal patterns the agent has learned during training. They introduce a logic-based *normative supervisor* module. It functions by informing the reinforcement learning agent of compliant actions it could take. The agent then chooses from among the actions complying with the norms in force, and one of the least evil in case there are none.

The normative supervisor is encoded in [NBCG21] in defeasible deontic logic. Defeasible deontic logic is a deontic logic with defeasible rules. The latter are rules that specify typical correlations. An example would be that birds usually fly. This however would not hold if it is known that the bird is a penguin. These exceptions are encoded through so-called defeaters.

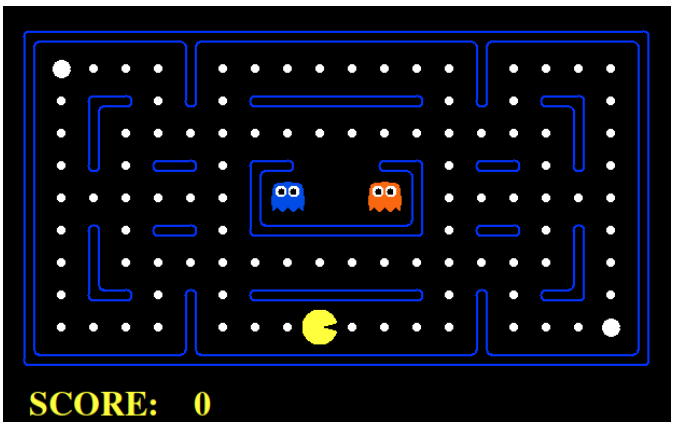
Neufeld et al. tested their framework on a reinforcement learning agent trained to play the game Pacman. This game was chosen as it simulates a closed environment with clearly defined (and simple) game mechanics. They added two norm bases in order to test the functionality of their framework.

Using a theorem prover for defeasible deontic logic (SPINdle), these normbases consisting of simple ethical constraints (explained in more detail in the next section) are then enforced upon the reinforcement learning agent.

This chapter explores an alternate implementation for the normative supervisor, using the DLV reasoner in the provided framework. We aim to show that the methods described in previous chapters suffice to encode the norm bases encoded in the theorem prover used in [NBCG21].

5.1 Pacman and Its "Ethical" Rules

Figure 5.1: Pacman



The video game on which the reinforcement agent is trained is the well known Pacman game. The starting point of the game can be seen in Figure 5.1. In this game the aim is for the player of the Pacman character to eat all the pellets placed on the maze that can be seen in Figure 5.1. There are two ghosts on the map which upon touching Pacman kill him. Usually Pacman and the ghosts move one step at a time. Should Pacman eat one of the larger pellets on the map, the ghosts enter a scared state allowing him to eat the ghosts as well. In this scared state, ghosts only move half a step at a time. A scared ghost is eaten by Pacman, as soon as they overlap in the graphical interface of the game. In other words, a scared ghost is eaten if the distance between the ghost and Pacman is less than 1 (so either 0.5 or 0) on both axes. Points in the game are given for pellets and ghosts eaten by Pacman before dying with points being deducted depending on how long the game lasted (the longer the game lasts the more points are deducted). Should Pacman collect all the pellets he wins. Note that finishing the game quicker is beneficial as less points are deducted. Two norm bases for this game were introduced simulating "ethical" constraints. Note that the names for the two norm bases ("Vegetarian" and "Vegan") may seem strange but we used the same names introduced in [NBCG21]. In order for the naming to be more intuitive, one may call the blue ghost "chicken" and the orange ghost "egg", as a vegetarian may eat eggs but not chicken and a vegan may not eat either.

Vegan

The vegan normbase forbids Pacman from eating any ghost. This can be written in the following way:

$$O(\neg eat(ghost))$$

or alternatively using the operator F for a prohibition:

$$F(eat(ghost))$$

Note that in the DLV encoding this normbase will not be so simple.

Vegetarian

The vegetarian normbase only prohibits Pacman from eating the blue ghost. (Pacman is allowed to eat the orange ghost.) This can once again be written in the following way:

$$O(\neg eat(blue\_ghost))$$

or alternatively using the operator F for a prohibition:

$$F(eat(blue\_ghost))$$

5.2 Technical Details

Neufeld provided the code for the framework presented in [NBCG21], as well as the documentation that was used in order to implement the DLV reasoner. The code was written in Java.

In order to embed DLV into Java, the Eclipse IDE plugin JDLV was used as a starting point [FLGR12]. This plugin was provided by DLV systems. As there was an error with the JDLV implementation, which we were not able to fix, we manually edited working code, that was delivered with JDLV as a test case, in order to generate working code for our example.

As JDLV requires an older version of the Eclipse IDE (Eclipse Juno) that is only compatible with an older version of Java (Java 7), the code provided by Neufeld was rewritten in order to omit Java 8 functionalities.

All the experiments and coding was done on a Lenovo Y50-70 PC running Ubuntu 22.04 LTS.

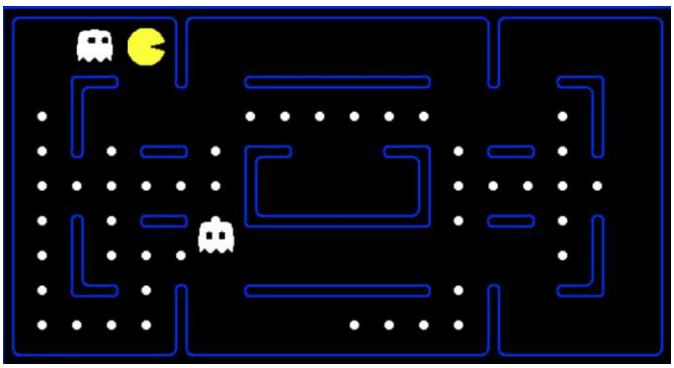
5.3 Encoding of the Norms

The way to encode the above norm bases is rather straightforward. We forbid the agent playing Pacman to move in a direction if the ghosts are frightened and moving into that direction could lead to a ghost being eaten. Furthermore, we forbid Pacman from stopping if this could lead to the ghost moving into Pacman (thereby leading to Pacman eating the ghost). Note that the encodings of the vegan and vegetarian normbase only differ in the way that the vegatarian normbase implements the weak constraints for the blue ghost, whereas the vegan version will implement the weak constraints for both ghosts.

Note that it is still possible for Pacman to eat a ghost. This could be the case if both a ghost and Pacman move towards a larger pellet from perpendicular directions. In that case Pacman will eat the pellet and immediately afterwards eat the ghost. As this could happen in [NBCG21], this will also be possible in this work. Furthermore, Pacman could be cornered between two frightened ghosts leaving the agent no choice but to eat one of the ghosts in the vegan norm base.

We start by considering the possible scenarios that could precede Pacman eating a ghost. These scenarios are the same for both norm bases. Since both the ghost and Pacman can move at most one field at a time, we can deduce that the Manhattan distance between Pacman and a frightened ghost can be at most two and at least one in the step preceding Pacman eating the ghost (as the location of the characters is given through integers). Let (x_1, y_1) be Pacman's coordinates and (x_2, y_2) the coordinates of a frightened ghost right before it is eaten. This gives us the following three possibilities for their relative locations:

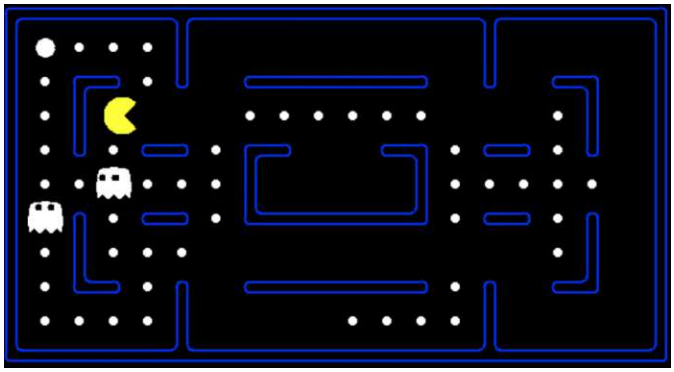
Figure 5.2: An example for case 1



1. Pacman and the frightened ghost are on the same path or offset by 0.5 on only on axis (meaning $x_1 = x_2 \pm 0.5$ or $y_1 = y_2 \pm 0.5$) and the distance between them is at most 1 on the other axis (meaning $|x_1 - x_2| \leq 1$ or $|y_1 - y_2| \leq 1$). An example of this case can be seen in Figure 5.2.

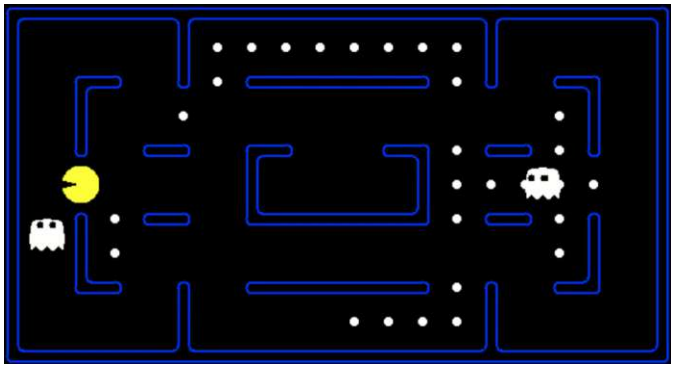
There are multiple options that can lead to the ghost being eaten in this situation. Either Pacman stops and the ghost moves into Pacmans direction or the ghost stops and Pacman moves into the ghosts direction or both move towards each other. In this case, forcing Pacman to move into a direction that is not the direction the ghost is in, suffices to ensure that Pacman cannot eat the ghost. This can be reformulated as stating that Pacman is prohibited from stopping or moving towards the ghost.

Figure 5.3: An example for case 2



2. Pacman and the frightened ghost are on the same path or offset by 0.5 on one axis (meaning $x_1 = x_2 \pm 0.5$ or $y_1 = y_2 \pm 0.5$) and the distance between them is at most 2 on the other axis (meaning $|x_1 - x_2| \leq 2$ or $|y_1 - y_2| \leq 2$). An example of this case can be seen in Figure 5.3.
   In this case, the ghost could be eaten if the ghost and Pacman both move towards each other. That ghost cannot be eaten in such a situation if Pacman does not move in the direction of the frightened ghost. (Stopping is a valid option in this case as long as the distance is more than 1, else the first case would hold.)

Figure 5.4: An example for case 3



3. Pacman and the ghost have a Manhattan distance of at most 2 and Pacman and the ghost are moving towards the same corner (in other words $|x_1 - x_2| \leq 1$ and $|y_1 - y_2| \leq 1$). An example of this case can be seen in Figure 5.4.
   In this case, the ghost could be eaten if the ghost and Pacman both move towards the same space. Therefore, prohibiting Pacman from moving towards the ghost would stop Pacman from eating the ghost in this situation.

Due to an error in the current implementation of JDLV some functionalities that are available in DLV did not work when using JDLV. The following section will start by explaining how the norms would be encoded if those functionalities worked in JDLV and the section afterwards will explain the workaround used in the actual program.

5.3.1 Theoretical Encoding

DLV is capable of handling basic arithmetic, using for example the predicates $+, -, *$. The predicate used in the theoretical encoding is $-$. $-(X, Y, Z)$ holds true if $Z = X - Y$. Furthermore, positive integers can be compared using the common comparison operators $<, <=, ==, >, >=$ [BFI+20].

The code gets updated after every move of the agent and gets passed the following predicates by the game:

- $diamond(X)$, where X is a direction (north, east, south or west). This predicate denotes that it is possible for Pacman to move into this direction. (Meaning there is no wall blocking him from moving in that direction.)
- $pacman(X, Y)$, where X and Y denote the location of pacman on the x-axis respectively the y-axis.
- $blueGhost(X, Y, Z)$, where X and Y denote the location of the blue ghost on the x-axis respectively the y-axis. Z is a boolean that denotes that the ghost is scared if $Z = 1$.
- $orangeGhost(X, Y, Z)$, where X, Y, Z have the same meanings as for $blueGhost$.
- $F(direction)$ will be added when it is impossible for Pacman to move into that direction (as could be the case when there is a wall in that direction). Note that F is the predicate we use for prohibtion. In the code this could be $F(east)$ as an example.

Using these predicates we use weak constraints to encode the norms forbidding Pacman from taking the given actions. We do this by encoding a weak constraint for each of the cases mentioned earlier. Note that DLV is only capable of working with positive

integer values. Therefore, we double the value of each coordinate. Then, Pacman always moves two coordinates and a scared ghost will move only one coordinate. In the following cases we will only show a weak constraint for one direction as an example, as the weak constraint is almost the same when the relative positions between Pacman and the ghost change.

1. In the first possible case, the distance between Pacman and the frightened ghost is at most 1 on one axis and at most 0.5 on the other. We therefore want to forbid Pacman from moving towards the ghost or stopping. (As the ghost could move into Pacman if Pacman stops.) We encode this by adding the following four weak constraints (for each direction the ghost could be in relative to Pacman and for each possible shift in the other axis). As an example, we show the case where the scared ghost is to the right of Pacman:

   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 1, -F(east).[1:4]$$
   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 1, -F(stop).[1:2]$$
   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 1, -F(east).[1:4]$$
   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 1, -F(stop).[1:2]$$

   In the case of the vegan normbase, the same rules need to be added for the orange ghost. The weight of the upper weak constraint is higher as moving towards the ghost is worse than stopping. The weights of the weak constraints are important as we do not want Pacman to not have any possible moves.

2. In the second case, Pacman and the frightened ghost are on the same path (meaning one of their coordinates are identical) and their distance is 2. We encode this by adding the following weak constraint (for each direction the ghost could be relative to Pacman). As an example, we show the case where the scared ghost is to the right of Pacman:

   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 4, -(D,B,G), G \leq 1, -F(east).[1:3]$$
   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 4, -(B,D,G), G \leq 1, -F(east).[1:3]$$

   In the case of the vegan normbase, the same rule needs to be added for the orange ghost. The weights of the weak constraints are chosen as stopping is preferred over moving towards the direction of the ghost when both options are not optimal.

3. In the third and final case, Pacman and the ghost have a Manhattan distance of 2 but Pacman and the ghost are not on the same path. (Intuitively, Pacman is around the corner of the ghost.) We encode this by adding the following two weak constraints (for each direction the ghost could be in relative to Pacman). As an example, we show the case where the scared ghost is above and to the right of Pacman:

   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 2, -F(east).[1:3]$$
   $$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 2, -F(north).[1:3]$$

   In the case of the vegan normbase, the same rule needs to be added for the orange ghost. The weights of the weak constraints are chosen as stopping is preferred over moving towards the direction of the ghost when both options are not optimal.

Finally, we also want to ensure that Pacman always has at least one valid move (stopping does count as a move), so we add the following rule:

$$:- F(north), F(east), F(south), F(west), F(stop).$$

Note that this is not a weak constraint, as it is not possible for Pacman to choose none of these options.
Using the discussed weak constraints and rules we can encode the vegetarian norm base by adding the following code to the common core, shown in Figure 5.5:

Figure 5.5: Encoding of the vegetarian norm base

$$
\begin{array}{l}
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 1, -F(east).[1:4] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 1, -F(east).[1:4] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(D,B,G), G \leq 1, -F(west).[1:4] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(B,D,G), G \leq 1, -F(west).[1:4] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(B,D,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(D,B,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(C,B,E), E \leq 2, -(D,A,G), G \leq 1, -F(north).[1:4] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(C,B,E), E \leq 2, -(A,D,G), G \leq 1, -F(north).[1:4] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(C,B,E), E \leq 2, -(A,D,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(C,B,E), E \leq 2, -(D,A,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(B,C,E), E \leq 2, -(D,A,G), G \leq 1, -F(south).[1:4] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(B,C,E), E \leq 2, -(A,D,G), G \leq 1, -F(south).[1:4] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(B,C,E), E \leq 2, -(A,D,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(B,C,E), E \leq 2, -(D,A,G), G \leq 1, -F(stop).[1:2] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 4, -(D,B,G), G \leq 1, -F(east).[1:3] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 4, -(B,D,G), G \leq 1, -F(east).[1:3] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 4, -(D,B,G), G \leq 1, -F(west).[1:3] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 4, -(B,D,G), G \leq 1, -F(west).[1:3] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(C,B,E), E \leq 4, -(A,D,G), G \leq 1, -F(north).[1:3] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(C,B,E), E \leq 4, -(D,A,G), G \leq 1, -F(north).[1:3] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(B,C,E), E \leq 4, -(A,D,G), G \leq 1, -F(south).[1:3] \\
:\sim pacman(A,B), blueGhost(D,C,1), -(B,C,E), E \leq 4, -(D,A,G), G \leq 1, -F(south).[1:3] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 2, -F(east).[1:3] \\
:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 2, -F(north).[1:3]
\end{array}
$$

$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 2, -F(east).[1:3]$
$:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 2, -F(south).[1:3]$
$:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(D,B,G), G \leq 2, -F(west).[1:3]$
$:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(D,B,G), G \leq 2, -F(north).[1:3]$
$:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(B,D,G), G \leq 2, -F(west).[1:3]$
$:\sim pacman(A,B), blueGhost(C,D,1), -(A,C,E), E \leq 2, -(B,D,G), G \leq 2, -F(south).[1:3]$
$:- F(north), F(east), F(south), F(west), F(stop).$
$act(north).\ act(east).\ act(south).\ act(west).\ act(stop).$

Note that the previously listed predicates detailing the locations, status of the characters, and the directions in which Pacman cannot move would also be contained in the file. As the code for the vegan norm base can be generated from the above code by adding the equivalent weak constraints relating to the orange ghost, it will not be explicitly written down. Alternatively, the coordinates of the ghosts can be given in the following way:

$$ghost(col, X, Y, scared)$$

In this case the color of the ghost would also be given as part of the ghost predicate.

5.3.2 Practical Encoding

The $-$ predicate did not work in JDLV, the java implementation of DLV we used to integrate DLV reasoning into the norm base. The exact reason for the predicate not working is unknown to us but our assumption is that it is a bug.

In order to substitute the $-$ predicate, we introduce three new predicates $tooclose(direction, boolean)$, $toooclose(direction, boolean)$ and $cornerclose(direction1, direction2, boolean)$. These predicates were passed along when a ghost was too close to Pacman. Each of these predicates was meant to substitute one case mentioned in the previous section. Note that the distances in the theoretical encoding are multiplied by 2.

1. $tooclose(direction, boolean)$ is active in the case where Pacman and a scared ghost have a distance of at most 1 on one axis and at most 0.5 on the other axis. The boolean is 1 if the scared ghost is the blue ghost and 0 otherwise. As an example, the following code in the theoretical encoding:

   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 1, -F(east).[1:4]$
   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 1, -F(east).[1:4]$
   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(B,D,G), G \leq 1, -F(stop).[1:2]$
   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 1, -F(stop).[1:2]$

   is replaced by:

   $$:\sim tooclose(east, 1), -F(east).[1:4]$$
   $$:\sim tooclose(X, 1), -F(stop).[1:2]$$

in the practical encoding. Note that the second weak constraint contains the variable X as it does not matter in what direction the ghost is too close. Stopping should not be allowed in such a situation.

2. $toooclose(direction, boolean)$ is active in the case where Pacman and a scared ghost have a distance of at most 2 on one axis and at most 0.5 on the other axis. Note that the name is the same as the name of the previous predicate, except with an extra o (due to the distance being 1 larger). The boolean is 1 if the scared ghost is the blue ghost and 0 otherwise. (Note that $toooclose$ is always active if $tooclose$ is active but not vice versa.) As an example, the following code in the theoretical encoding:

   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 4, -(D,B,G), G \leq 1, -F(east).[1:3]$
   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 4, -(B,D,G), G \leq 1, -F(east).[1:3]$

   is replaced by:

   $$:\sim toooclose(east,1), -F(east).[1:3]$$

   in the practical encoding.

3. $cornerclose(direction1, direction2, boolean)$ is active in the case where Pacman and the ghost have a Manhattan distance of 2 but Pacman and the ghost do not share a coordinate. As an example, the following code in the theoretical encoding:

   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 2, -F(east).[1:3]$
   $:\sim pacman(A,B), blueGhost(C,D,1), -(C,A,E), E \leq 2, -(D,B,G), G \leq 2, -F(north).[1:3]$

   is replaced by:

   $$:\sim cornerclose(north,east,1), -F(east).[1:3]$$
   $$:\sim cornerclose(north,east,1), -F(north).[1:3]$$

   in the practical encoding.

The code added to the common core in the new encoding for the vegan normbase then takes the following form in its entirety, shown in Figure 5.6:

Figure 5.6: Encoding of the vegan norm base

$:\sim tooclose(X, Y), -F(stop).[1:2]$
$:\sim tooclose(east, Y), -F(east).[1:4]$
$:\sim tooclose(west, Y), -F(west).[1:4]$
$:\sim tooclose(north, Y), -F(north).[1:4]$
$:\sim tooclose(south, Y), -F(south).[1:4]$
$:\sim toooclose(east, Y), -F(east).[1:3]$
$:\sim toooclose(west, Y), -F(west).[1:3]$
$:\sim toooclose(north, Y), -F(north).[1:3]$
$:\sim toooclose(south, Y), -F(south).[1:3]$
$:\sim cornerclose(north, east, Y), -F(east).[1:3]$
$:\sim cornerclose(north, east, Y), -F(north).[1:3]$
$:\sim cornerclose(south, east, Y), -F(east).[1:3]$
$:\sim cornerclose(south, east, Y), -F(south).[1:3]$
$:\sim cornerclose(north, west, Y), -F(west).[1:3]$
$:\sim cornerclose(north, west, Y), -F(north).[1:3]$
$:\sim cornerclose(south, west, Y), -F(west).[1:3]$
$:\sim cornerclose(south, west, Y), -F(south).[1:3]$
$:- F(north), F(east), F(south), F(west), F(stop).$
$act(north).$
$act(east).$
$act(south).$
$act(west).$
$act(stop).$

Note that in the weak constraints above, the boolean was replaced with a variable Y, as the value of the boolean (or in other words the ghost that is considered) does not matter to the agent. The practical encoding for the vegetarian normbase can be constructed by simply replacing the variable Y with the boolean 1 in the above encoding. Note that in the actual encoding the directions are written in quotation marks due to coding reasons. They are left out in the text for the sake of readability.

In order to shorten the code, one could introduce variables to replace the directions. The shortened code would then take the form shown in Figure 5.7:

Figure 5.7: Compact encoding of the vegan norm base

$:\sim tooclose(X, Y), -F(stop).[1:2]$
$:\sim tooclose(dir, Y), -F(dir).[1:4]$
$:\sim toooclose(dir, Y), -F(dir).[1:3]$
$:\sim cornerclose(dir1, dir2, Y), -F(dir1).[1:3]$
$:\sim cornerclose(dir1, dir2, Y), -F(dir2).[1:3]$
$:- F(north), F(east), F(south), F(west), F(stop).$
$act(north).$
$act(east).$
$act(south).$
$act(west).$
$act(stop).$

5.4 Results

For the comparison of our work to Neufeld et al.'s work [NBCG21] we used the same conditions as they did. The reinforcement learning agent was trained on 250 games and then the normative supervisor was tested using 1000 test games. The game always starts the same way, seen in Figure 5.1. Using our encodings for the norm bases, we got the following results:

Normbase	% Games won	Game score (Avg[Max])	Avg ghosts eaten (blue/orange)
Vegan	91.2	1217[1538]	0.013/0.018
Vegetarian	90.6	1366[1751]	0.001/0.788

Compared to the results given by Neufeld et al. in [NBCG21]:

Normbase	% Games won	Game score (Avg[Max])	Avg ghosts eaten (blue/orange)
Vegan	90.7	1209.86[1708]	0.023/0.02
Vegetarian	94	1413.8[1742]	0.01/0.79

For the vegan norm base our encoding outperformed Neufeld et al.'s encoding. Less ghosts of each color were eaten and the agent won more games using our encoding. The lower maximal game score can be explained by the fact that eating ghosts awards 200 points. The higher average game score in our encoding is most likely due to the higher number of games won, as losing a game always counts as zero points. Therefore, eating

more ghosts (which is the opposite of our goal) would lead to higher game scores. Our theory for the higher maximal game score for Neufeld et al.'s vegan norm base could be explained by there being a game where both ghosts were eaten by Pacman, while there was no such game for our encoding. So the norms aiming to prevent Pacman from eating the ghosts were successfully implemented.

For the vegetarian norm base there are two notable differences. First, the average number of blue ghosts eaten for our encoding is 1/10 of the average number of ghosts eaten in Neufeld et al.s encoding. Therefore, the goal of the vegetarian norm base was fulfilled. However, the percentage of games won was decently lower for our encoding. In our encoding Pacman prefers running into the non-scared orange ghost (thereby losing the game) over running into the scared blue ghost. This could lead to losing more games. Note that the lower average game score is probably a result of the higher number of losses (which award no points). All in all we consider the encoding of the vegetarian norm base a success as well. The maximal game scores are roughly the same for both encodings.

CHAPTER 6

Conclusion

In this thesis, we introduced a method for encoding deontic paradoxes and more generally normative systems in ASP using DLV as the system of choice.

We started by considering SDL, the first logic for reasoning about normative concepts such as obligations. Using some of the most famous deontic paradoxes to figure out why SDL is unable to capture the intricacies of common sense reasoning, we built the foundation for our encodings. These ground rules for reasoning about norms were encoded in the common core used in all our encodings for the deontic paradoxes. We were able to encode the paradoxes presented in this master thesis in a uniform and intuitive fashion using a common core among all the encodings. Weak constraints proved to be a great tool for encoding conflicting obligations as the level resp. the weights of the weak constraints can be used to encode the priority of the obligations.

The choice of paradoxes proved to be very important as omitting certain paradoxes lead to a less expressive methodology for encoding normative systems. As an example, without the addition of the *Fence Paradox*, CTD-obligations would have been active even when an exception to the obligation is given.

In order to generalise the encodings of the paradoxes, we described the different kinds of obligations and how to encode them. This general way of encoding and distinguishing the different obligations, seen in the considered paradoxes, and some simple rules sufficed to build our methodology for encoding normative systems in DLV.

We tested our methodology in multiple ways. Firstly, we built a normative system consisting of different obligations seen in the paradoxes. Using this normative system, we encoded some situations to serve as test cases in order to determine whether we were able

to deduce the correct obligations in given situations. Furthermore, we encoded a simple norm base in the game Pacman that was used to enforce rules on a reinforcement learning agent. Through these tests we were able to verify that our methodology is capable of encoding normative systems.

One of the strongest advantages that our approach provides is the simplicity of the encoding. In addition to using a commonly researched and well optimized programming approach with Answer Set Programming, there is a clearly defined common core as well as well defined ways for encoding different kinds of obligations. Due to the amount of research on Answer Set Programming the behavior of ASP programs is well studied and ASP solvers are optimized. The approach described for DLV in this work is also easily transferable to other ASP languages, such as ASP-Core-2 [CFG+20].

While our proposed method does have some positive characteristics, it does have some weaknesses as well.

The biggest weakness of our proposed method is that encoding complex normative systems can lead to extremely large (and confusing) files. This can be seen in Chapter 5 with the encoding of the vegan norm base for Pacman. Encoding an obligation as simple as "do not eat a ghost" led to a large number of weak constraints needing to be added. Note that Pacman as a game was chosen due to its simplicity and the available implementation using reinforcement learning. Although the number of weak constraints needed in order to encode these simple constraints for the two norm bases was large, our encoding was able to successfully enforce the desired constraints. Our encoding even outperformed the supervisor seen in [NBCG21] when considering ghosts eaten for both norm bases. For the vegan norm base, our encoding even led to a higher percentage of games won as well as a higher average score (which likely corresponds to the higher number of games won as a lost game is worth 0 points).

In general, obligations that require the agent to uphold certain conditions might be complicated to encode for more complex examples. In such cases taking particular actions might indirectly lead to failure of upholding the condition, thereby violating the obligation. For such normative systems using temporal logics approaches in addition to the DLV reasoner may be preferable. Encoding these normative systems also requires a large amount of knowledge about the underlying topic, as conflicts between actions (actions that cannot be taken at the same time) need to be encoded as well. Without the knowledge of movement speed of Pacman and the movement constraints placed on Pacman, respectively the ghosts it would not have been possible to encode the norm base in an effective way.

The simplicity of our methodology leads to another weakness however. In our methodology all encoded obligations are required to be comparable. As such our methodology is limited to encoding only normative systems for which possible ways of satisfying norms partially are comparable. While we did not run into problems with our methodology due to this, there may be cases where answer sets being incomparable at time may be necessary. Future work could look into other ASP solvers with more sophisticated methods for filtering out answer sets to model normative reasoning. The clasp extension asprin [BDRS15] or the DLV2 system that uses the WASP solver [ADMR20], could be used as examples.

All in all our approach lends itself to encode normative systems when the aim is to determine optimal ways to handle scenarios using agreed upon prioritization and weights of the obligations. Our approach could also be used in combination with other software which determines how to satisfy the given obligations. In the case of Pacman such a software might have interpreted how the obligation to not eat a ghost could be fulfilled. Such approaches were not considered in our work as it would have exceeded the scope of this thesis but remains for further research.

Future work could look into encoding other Deontic Logics, such as Dyadic Deontic Logic, to encode normative systems, as well as further applications of our method to encode normative systems.

CHAPTER 7

Appendix

Abbreviations

AI Artificial Intelligence 2

ASP Answer Set Programming 2

CTD Contrary-to-Duty (usually referring to a type of obligation) 29

DD Part of the axiomatization of SDL 19

DDL Defeasible Deontic Logic 2

DLV DataLog with Disjunction 3

JDLV Java Implementation of DLV 85

KD Part of the axiomatization of SDL 19

OD A theorem in SDL 20

ON A theorem in SDL 20

RMD A theorem in SDL 20

RND Part of the axiomatization of SDL 19

SDL Standard Deontic Logic 3

Paradoxes

Bibliography

[ABDL15] Natasha Alechina, Nils Bulling, Mehdi Dastani, and Brian Logan. Practical run-time norm enforcement with bounded lookahead. In Gerhard Weiss, Pinar Yolum, Rafael H. Bordini, and Edith Elkind, editors, *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 443–451. ACM, 2015.

[ADL18] Natasha Alechina, Mehdi Dastani, and Brian Logan. Norm specification and verification in multi-agent systems. *Journal of Applied Logics*, 5(2):457–490, April 2018.

[ADMR20] Mario Alviano, Carmine Dodaro, João Marques-Silva, and Francesco Ricca. Optimum stable model search: algorithms and implementation. *J. Log. Comput.*, 30(4):863–897, 2020.

[And58] Alan Ross Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, 67(265):100–103, 1958.

[Bar77] Jon Barwise. An introduction to first-order logic. In *Handbook of Mathematical Logic*, pages 5–46. Elsevier, 1977.

[BDHvdT02] Jan Broersen, Mehdi Dastani, Joris Hulstijn, and Leon van der Torre. Goal generation in the boid architecture. *Cognitive Science Quarterly*, 2:428–447, 01 2002.

[BDK13] Nils Bulling, Mehdi Dastani, and Max Knobbout. Monitoring norm violations in multi-agent systems. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 491–498. IFAAMAS, 2013.

[BDRS15] Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub. asprin: Customizing answer set preferences without a headache. In *AAAI*, pages 1467–1474. AAAI Press, 2015.

[BFI+20] Robert Bihlmeyer, Wolfgang Faber, Giuseppe Ielpa, Vincenzino Lio, and Gerald Pfeifer. Dlv user manual. https://www.dlvsystem.it/dlvsite/dlv-user-manual/, Apr 2020.

[Bic06] Cristina Bicchieri. *The Grammar of Society: The nature and dynamics of social norms*. Cambridge University Press, 01 2006.

[BKI19] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme*. Springer Fachmedien Wiesbaden, 6 edition, August 2019.

[BLR97] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Strong and weak constraints in disjunctive datalog. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97, Dagstuhl Castle, Germany, July 28-31, 1997, Proceedings*, volume 1265 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 1997.

[Bro13] John Broome. *Rationality Through Reasoning*. John Wiley & Sons, Ltd, September 2013.

[BvdT04] Guido Boella and Leendert W. N. van der Torre. Regulative and constitutive norms in normative multiagent systems. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, pages 255–266. AAAI Press, 2004.

[CFG+20] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309, 2020.

[FLGR12] Onofrio Febbraro, Nicola Leone, Giovanni Grasso, and Francesco Ricca. Jdlv user manual. https://www.dlvsystem.it/dlvsite/wp-content/uploads/2020/04/JDLV-Eclipse-Plugin-Manual.pdf, Apr 2012.

[GA12] Ricardo Gonçalves and José Júlio Alferes. An embedding of input-output logic in deontic logic programs. In Thomas Ågotnes, Jan M. Broersen, and Dag Elgesem, editors, *Deontic Logic in Computer Science - 11th International Conference, DEON 2012, Bergen, Norway, July 16-18, 2012. Proceedings*, volume 7393 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2012.

[GHP+13] Dov Gabbay, John Horty, Xavier Parent, Ron van der Meyden, and Leendert van der Torre, editors. *Handbook of Deontic Logic and Normative Systems*. College Publications, 2013.

[GHP+21] Dov Gabbay, John Horty, Xavier Parent, Ron van der Meyden, and Leendert van der Torre, editors. *Handbook of Deontic Logic and Normative Systems, Volume 2*. College Publications, 2021.

[GMD13] Laura Giordano, Alberto Martelli, and Daniele Theseider Dupré. Temporal deontic action logic for the verification of compliance to norms in ASP. In Enrico Francesconi and Bart Verheij, editors, *International Conference on Artificial Intelligence and Law, ICAIL '13, Rome, Italy, June 10-14, 2013*, pages 53–62. ACM, 2013.

[GORS13] Guido Governatori, Francesco Olivieri, Antonino Rotolo, and Simone Scannapieco. Computing strong and weak permissions in defeasible logic. *J. Philos. Log.*, 42(6):799–829, 2013.

[GR08] Guido Governatori and Antonino Rotolo. BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Auton. Agents Multi Agent Syst.*, 17(1):36–69, 2008.

[Han08] Jörg Hansen. *Imperatives and Deontic Logic – On the Semantic Foundations of Deontic Logic*. PhD thesis, University of Leipzig, 2008.

[Hor94] John F. Horty. Moral dilemmas and nonmonotonic logic. *J. Philos. Log.*, 23(1):35–65, 1994.

[Hor97] John F. Horty. Nonmonotonic foundations for deontic logic. In Donald Nute, editor, *Defeasible Deontic Logic*, pages 17–44. Springer Netherlands, Dordrecht, 1997.

[JC02] Andrew Jones and José Carmo. Deontic logic and contrary-to-duties. *Handbook of Philosophical Logic*, vol.8:p.265–364, 01 2002.

[JS92] Andrew J. I. Jones and Marek J. Sergot. Deontic logic in the representation of law: Towards a methodology. *Artif. Intell. Law*, 1(1):45–64, 1992.

[KS18] Robert A. Kowalski and Ken Satoh. Obligation as optimal goal satisfaction. *J. Philos. Log.*, 47(4):579–609, 2018.

[MNT11] Victor W. Marek, Ilkka Niemelä, and Miroslaw Truszczynski. Origins of answer-set programming - some background and two personal accounts. *CoRR*, abs/1108.3281, 2011.

[MvdT01] David Makinson and Leendert W. N. van der Torre. Constraints for input/output logics. *J. Philos. Log.*, 30(2):155–185, 2001.

[NBCG21] Emery A. Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. A normative supervisor for reinforcement learning agents. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction - CADE 28 - 28th*

International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of *Lecture Notes in Computer Science*, pages 565–576. Springer, 2021.

[PS96] Henry Prakken and Marek J. Sergot. Contrary-to-duty obligations. *Stud Logica*, 57(1):91–115, 1996.

[PT18] Xavier Parent and Leendert Van Der Torre. *Introduction to Deontic Logic and Normative Systems*. College Publications, December 2018.

[Rau10] Wolfgang Rautenberg. *A Concise Introduction to Mathematical Logic*. Springer New York, 2010.

[SBD+00] V.S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, and Robert Ross. *Heterogeneous Agent Systems*, chapter 6: Agent Programs, pages 115–170. MIT PR, June 2000.

[SSK+86] Marek J. Sergot, Fariba Sadri, Robert A. Kowalski, F. Kriwaczek, Peter Hammond, and H. T. Cory. The british nationality act as a logic program. *Commun. ACM*, 29(5):370–386, 1986.

[vBCF+22] K. van Berkel, A. Ciabattoni, E. Freschi, F. Gulisano, and M. Olszewski. Deontic paradoxes in mimamsa logics: there and back again. *Journal of Logic Language and Information*, 2022.

[vdT94] Leendert van der Torre. Violated obligations in a defeasible deontic logic. In Anthony G. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8-12, 1994*, pages 371–375. John Wiley and Sons, Chichester, 1994.