Representing Normative Reasoning in Answer Set Programming Using Weak Constraints

Christian Hatschka[1] [0000−0002−0881−8259], Agata Ciabattoni[1,3] [0000−0001−6947−8772], and Thomas Eiter[1,2] [0000−0001−6003−6345]

[1] Institute of Logic and Computation, TU Wien, Vienna, Austria
christian.t.hatschka@gmail.com
[2] Knowledge-Based Systems Group, TU Wien, Vienna, Austria
eiter@kr.tuwien.ac.at
[3] Theory and Logic Group, TU Wien, Vienna, Austria
agata@logic.at

Abstract. Deontic logics and normative reasoning keep gaining importance, as ethical AI and multi-agent systems are becoming increasingly more relevant these days, and tools that support decision making based on norms and regulations are needed. Challenges to this are illustrated by well known benchmark examples (referred to as deontic paradoxes) on which common deontic logics, e.g., Standard Deontic Logic, fail. In this work we encode a range of famous paradoxes in Answer Set Programming (ASP) using weak constraints. Their abstraction and generalization provides a plain methodology for encoding normative systems in this language. An experimental comparison to the normative supervisor by Neufeld et al. (CADE 2021) on "ethical" Pacman shows that the ASP encoding leads to comparable performance but ethically preferable results.

1 Introduction

Norms, which involve concepts such as obligation and prohibition, are an integral part of human society. They are enormously important in a variety of fields – from law and ethics to artificial intelligence (AI). In particular, imposing norms – be they ethical, legal or social – on AI systems is crucial, as these systems have become ubiquitous in our daily lives. Reasoning with and about them (normative reasoning) requires deontic logic, the branch of logic that deal with obligation, permission and related concepts. Normative reasoning comes with a variety of idiosyncratic challenges, which are often exemplified by benchmark examples (so called paradoxes). One of the challenges is reasoning about sub-ideal situations, such as contrary-to-duty (CTD) obligations, which are obligations only triggered by a violation. Other challenges in formalizing normative reasoning are associated, e.g., with defeasibility issues (norms having different priorities, exceptions, etc.): norms are indeed inherently violable, so systems of normative reasoning must be able to reason in the presence of violations. The first deontic system introduced – Standard Deontic Logic [vW51] – was failing most of the benchmark examples, (un)deriving formulas which are counterintuitive in a common-sense reading. This has motivated the introduction of a plethora of deontic logics, see, e.g. the Handbook volumes [GHP+13, GHP+21]. These logics have been investigated mainly in connection with philosophy and legal reasoning, and with the exception of Defeasible Deontic Logic [GORS13, GR08], they lack defeasibility and efficient provers. Defeasibility and efficient reasoning methods are instead offered by Answer Set Programming (ASP), which is one of the most successful paradigms of knowledge representation and reasoning for declarative problem solving. Indeed in a long and systematic effort of the knowledge representation community effective tools were developed that are capable of processing programs in ASP fast [MNT11, BET16]. Defeasibility is also not a problem when using ASP, due to its default-negation and weak constraints.

In this paper we introduce a method for encoding normative systems in Answer Set Programming (ASP) using weak constraints. We first encode desired basic properties of the deontic operators in a common core that will be used in all further encodings. The desired properties of these operators are then established by analyzing multiple well known deontic paradoxes (e.g., *Chisholm's Paradox*, the *Good Samaritan Paradox*,...). Our encoding is capable of handling the deontic paradoxes in a satisfactory manner. By abstracting and generalising the encodings of the specific paradoxes, we provide a simple methodology for encoding normative systems in ASP. Our methodology is also tested and in a case study described in [NBCG21], which involves a reinforcement learning agent playing a variant of the Pacman video game with additional "ethical" rules. The resulting encoding is compared with a theorem prover for Defeasible Deontic Logic (the normative supervisor in [NBCG21]), showing some benefiting behaviour.

This work is a short version of the Master Thesis [Hat22].

2 Preliminaries

We provide a basic overview on Answer Set Programming and Standard Deontic Logic. We assume familiarity with propositional and first-order classical logic.

2.1 Answer Set Programming

Since its inception, logic programming has found multiple practical uses, e.g., in Expert Systems [BKI19, chapter 1]. We recall below the definitions in [BKI19, chapter 9], where further information can be found.

In this work we consider extended logic programs with disjunctions. Extended logic programs can use both strong negation and default-negation in rules. The latter, 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) permits a realistic processing of knowledge, which is more akin to that of human reasoning.

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

$$H_1 \vee H_2 \vee \dots H_l \leftarrow A_1, \dots A_n, \textit{ not } B_1, \dots, \textit{ not } B_m. \quad (1)$$

where $H_1, \dots, H_l, A_1, \dots, A_n, B_1, \dots, B_m$ are literals in a first-order language. Informally, it can be read as: "If $A_1, \dots, A_n$ hold and none of $B_1, \dots, B_n$ are found to be true, at least one of $H_1, \ldots, H_l$ must be true". A logic program Π is a (finite) set of rules.

Informally speaking, such programs model a problem using logic. Answer sets are then given as solutions to this problem. An answer set of it is a minimal, consistent and closed set of ground literals S that satisfies all rules (1) s.t. if $\{A_1, \ldots, A_n\} \subseteq S$ and $\{B_1, \ldots, B_m\} \cap S = \emptyset$, then $\{H_1 \ldots, H_l\} \cap S \neq \emptyset$. Thus, answer sets contain no information that is not a fact or deduced using a rule, do not contain conflicting information, and for all rules if the body of the rule holds a part of the head is in the answer set. Furthermore S can be reconstructed from Π.

In order to model the defeasibility of obligations we use weak constraints, which are constraints that are only satisfied if possible. We use the form:

$$:\sim A_1, \ldots, A_n.\ [w : l]$$

as in DLV [BFI+20], where $A_1, \ldots, A_n$ are literals (that may be weakly negated) and w and l denote the weight resp. level of the weak constraint.

Weak constraints are used to filter out answer sets. Intuitively the remaining answer sets are those that have minimal weights of violated weak constraints with higher levels being more important.

2.2 Standard Deontic Logic and its paradoxes

Standard Deontic Logic (SDL) is the best known system of deontic logic. Introduced by von Wright in [vW51], SDL builds upon classical propositional logic and is part of the class of normal modal logics [CZ97]. SDL is a so-called monadic deontic logic, as the operators O (obligation), P (permission) and F (prohibition) are one-place operators, meaning they apply only to a single formula. We recall below the main concepts on SDL, see, e.g. [GHP+13, GHP+21, JC02] for more details.

Syntax of SDL
The following grammar generates its language ($\mathbb{AT}$ is the set of atomic propositions):

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

$O\varphi$, $P\varphi$ and $F\varphi$ are read as "it is obligatory that φ", "it is permissible that φ", and "it is forbidden that φ", respectively. These operators are inter-definable, e.g., $P\varphi := \neg O(\neg\varphi)$, and $F\varphi := O\neg\varphi$.

Axioms of SDL
A Hilbert system for SDL is obtained by adding the following axioms and rules to any axiomatization of classical propositional logic, where φ and ψ are arbitrary formulas:

$$\text{If } \varphi \text{ is a theorem, } O\varphi \text{ is a theorem} \tag{RND}$$

$$O(\varphi \to \psi) \to (O\varphi \to O\psi) \tag{KD}$$

$$O\varphi \to \neg O\neg\varphi \tag{DD}$$

"Standard" Semantics of SDL

The semantics of SDL is based on the well-known Kripke Semantics of modal logic, in which sentences can be interpreted with regard to possible worlds, see, e.g. [CZ97].

For this purpose a possible worlds model $M = \langle W, R, I \rangle$ 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 (satisfaction) of a formula p under M at a possible world/situation $u \in W$, written as $M, u \models p$ (the situation u in the model M satisfies p). $M, u \models p$ is defined recursively as follows:

- 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:

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

We note the following rules and formulas that hold in SDL, which we shall use in the sequel:

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

$$\neg O\bot \tag{OD}$$

Deontic Paradoxes and Their Classification

As a basis for our work, we recall below different deontic paradoxes and their classification. Deontic paradoxes play an important role in deontic logic and normative reasoning, and are usually in the form of (un)derivable formulas which are counter-intuitive in a common-sense reading. They serve as sanity checks for deontic logics and as driving force for defining new deontic systems. Although referred to in the literature as paradoxes, many of the considered problems are not paradoxes *per se*, but rather puzzles or dilemmas. They are examples for which SDL is unable to capture the nuances of normative reasoning expected in these scenarios.

There are many such paradoxes. We categorised them below according to the reason for their failure (see e.g. [JC02, GHP$^+$13]) and analyze one example for each class:

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*

Note that we added few paradoxes to the classification that were previously not considered: *Broome's Counterexample* [Bro13], the *Considerate Assassin Paradox* [PS96], *Asparagus Paradox* [vdT94, Hor97], *Fence Paradox* [PS96], *Alternative Service Paradox* [Hor94].

Deontic conditionals refer to obligations that arise situationally. Those conditionals (sometimes written as $O(A/B)$ (meaning "it is obligatory that A if B") have been introduced to cope with contrary-to-duty obligations, which are obligations arising due to another obligation not being fulfilled.

Paradoxes centering around 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.

As an example for paradoxes of this class we present *Ross's Paradox*. This paradox consists of the following two sentences:

It is obligatory that the letter is mailed. (R1)

It is obligatory that the letter is mailed or burned. (R2)

that can be formalised as:

$$O(m) \tag{R1}$$

$$O(m \vee b) \tag{R2}$$

Since $m \rightarrow (m \vee b)$ is a theorem, the second obligation follows from the first via RMD. A key property of obligations is the possibility of not being satisfied. This will later be seen in the contrary-to-duty obligations. It seems counter-intuitive 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].

Puzzles centering around DD and OD
Paradoxes that arise from DD and OD are centered around obligations which are unobeyable. As an example we discuss *Plato's Dilemma*:

It is obligatory that I meet my friend for dinner. (P1)

It is obligatory that I rush my child to the hospital. (P2)

In this scenario, a medical emergency has arisen, which necessitates immediate intervention. Clearly, it is not possible to satisfy both obligations at the same time. 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
These paradoxes center around obligations that hold only under certain circumstances. As an example for paradoxes from this class we present an interesting paradox that combines two different weaknesses of SDL, the so-called *Fence Paradox* [PS96]:

There must be no fence. (F1)

If there is a fence then it must be a white fence. (F2)

If the cottage is by the sea, there may be a fence. (F3)

Here (F2) serves as a contrary-to-duty obligation that is active when the obligation (F1) is violated. (F3) serves instead as an exception to the obligation generated by (F1). Note that with this interpretation, it does not necessitate a fence being white if the cottage is by the sea. Due to SDL having no means for expressing defeasibility, (F3) cannot be formalised. The contrary-to-duty obligation (F2) cannot be formalised as having a white fence implies having a fence, thereby deriving the obligation to have a fence. This would contradict $(F1)$, something that cannot be handled by SDL.

3 Encoding Paradoxes

We now encode the presented paradoxes. All encodings share the same common core, shown in Figure 1, that encodes properties of SDL. The following predicates are used in the encoding:

- $O(X)$ 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 definitely 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.

Fig. 1: The common core of our encodings

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

– $Happens(X)$ 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.

Intuitively, the common core guesses whether something is obligatory (1), forbidden (2) and whether the agent takes the action (6). The remaining rules then encode connections between predicates and exclude answer sets that we deem inconsistent, e.g., something being obligatory and forbidden or something being obligatory and an action not taking the action. The weak constraints (10) and (11) are used to exclude answer sets that derive obligations/prohibitions that have no reason for existing.

Soundness of the rules (1) to (9) is achieved, as inconsistent answer sets are excluded. (DD) for example is encoded through rule (5) that forbids an action from being both forbidden and obligatory. Similarly other rules prevent all other possible inconsistencies. Completeness on the other hand is given, as all answer sets that are considered consistent for an action are generated.

Note that an answer set represents an optimal way to handle given norms. Therefore actions are only determined as obligatory if they can and should be taken.

Ross's Paradox

We first encode *Ross's Paradox*. $(R2)$ would be derived in SDL, but not 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:

$$:\sim -O(mail).[1:2] \tag{12}$$
$$act(mail). \tag{13}$$
$$act(burn). \tag{14}$$

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

First, the obligation $(R1)$ is created using the weak constraint (12). The final two facts (13) and (14) declare *mail* and *burn* as actions to reason about.

Since *mail* is specified as an action, the logic program must guess it as obligatory or not obligatory, by the core encoding. 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.

This logic program yields two answer sets that both do not 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.

Plato's Dilemma

Since in general conflicting obligations may not be in direct but in indirect conflict, 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. 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 her 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 taking both actions. This can be encoded as follows:

$$:\sim -O(help), Happens(emergency).[1:3] \quad (20)$$
$$:\sim -O(meet).[1:2] \quad (21)$$
$$act(meet). \quad (22)$$
$$act(help). \quad (23)$$
$$:- Do(help), Do(meet). \quad (24)$$
$$Happens(emergency). \quad (25)$$

The encoding starts with a weak constraint (20) 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 $(P1)$ to help the child in case of an emergency, as here $emergency$ is an auxiliary predicate that denotes an emergency currently occurring. The weak constraint (21) simply encodes the obligation $(P2)$ to meet the friend for dinner. As the first weak constraint is at a higher level than the second, the program always ensures that the former is satisfied before the latter.

The constraint (24) encodes that it is not possible to both help the child and meet the friend. The remaining assertions simply state that an emergency is currently occurring and which actions to reason about. As desired only a single answer set is computed, as the program prioritises the obligation to help the child.

Fence Paradox
The *Fence Paradox* is an interesting paradox as it combines a CTD obligation with an exception to it. 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 betters the situation, the fence itself should still not be there [PS96]. A similar example would be if one were to forget to wish a friend a happy birthday. In such a case, one should still congratulate the friend a few days later, although congratulating on the actual birthday would have been better.
The important fact to consider is that should the cottage be by the sea, then (as obligation $(F1)$ is not active due to the exception in $(F3)$) the fence does not necessarily need to be white. This will be encoded by adding the following rules to the common core:

$$:\sim -F(have\_fence),\ \ not\ \ Location(sea).\ [1:2] \tag{30}$$
$$:\sim Do(have\_fence),\ \ -O(have\_white\_fence),\ \ not\ \ Location(sea).\ [1:2] \tag{31}$$
$$act(have\_fence). \tag{32}$$
$$act(have\_white\_fence). \tag{33}$$

An interesting part of this encoding is that the exception $(F3)$ is also part of the contrary-to-duty obligation $(F2)$. This needs to be done as the fence has to be white only when the cottage is not by the sea. Otherwise the obligation for the fence to be white would also be derived if the cottage was 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:

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

These facts specify that the cottage is by the sea, and furthermore that there is a fence. Then the resulting two answer sets both do not derive the obligation for the fence to be white. When testing other cases, the answer sets also represent the expected results.

4 Generalisation and Methodology

We start by classifying the obligations that appeared in the paradoxes excluding obligations without any special properties, e.g., the obligation to mail the letter in *Ross's Paradox*. The classes of obligations are:

– Conditional obligations
– Obligations over disjunctions
– Conjunctions of obligations that all need to be satisfied
– Obligations with exceptions
– Contrary-to-duty obligations

Note that prohibitions are counted as regular obligations, as they can be understood as an obligation not to do something.

In all encoded paradoxes, obligations of the same class were encoded in the same way. All obligations were encoded using weak constraints, in order to model their defeasibility. An obligation (to take an action a) that always holds is encoded in the following way:

$$:\sim -O(a).\ [w : l]$$

Here w and l are the weight respectively level of the weak constraint. Note that the weight and level of the weak constraint depend on the importance of the obligation and conflicting obligations. In most cases the weight is 1 and only the level is used to encode priorities among obligations. Conflicts between obligations are determined and more important obligations are then generated through weak constraints at a higher level. A more thorough explanation can be found in the Appendix in Section A.

By generalising the encodings of all considered paradoxes (shown in Section 2.2), we develop the following more general method of encoding normative systems in ASP using weak constraints:

1. For each of the norms determine what kind of obligation they represent, e.g., obligation with an exception or contrary-to-duty obligation.
2. Determine which actions are incompatible at the same time. (Knowing which actions are conflicting makes it easier to determine the importance of the specific obligations.)
3. Encode the different kinds of obligations and their importance. Here weights as priorities play an important role. The methods for encoding different kinds of obligations can be seen in the Appendix in Section A.
4. Encode the exclusion of combinations of actions found incompatible.
5. Encode additional information, e.g., dependencies between actions, such as one action requiring another action, or what actions to reason about.

To test this methodology, we use as a showcase the *Fence Paradox*. A further, more complex case study can be found in the Appendix (see Section B).

Step 1: We start by determining the types of obligations.

– “There must be no fence.” is an obligation with an exception (the exception is if the the cottage is by the sea).
– “If there is a fence, then it must be white.” is a CTD obligation.

The considered actions are:

– having a fence.

– having a white fence.

(For being strictly an action, we may view “having” as “upholding”.)
Step 2: In this case there are no incompatible actions, therefore we can skip this step.

Step 3: We can now encode the two obligations and their importance. As the CTD obligation and the obligation with exception are not in contradiction (as the CTD obligation is only active when the first obligation would be violated anyways), the two obligations can be placed at the same level. Encoding the two obligations we get:

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

Step 4: This step can once again be skipped.

Step 5: We now encode additional information:

$$\boxed{\begin{array}{l} act(have\_fence). \\ act(have\_white\_fence). \end{array}}$$

One can additionally add the information specifying that having a white fence implies having a fence in the following way:

$$Do(have\_fence) :- Do(have\_white\_fence)$$

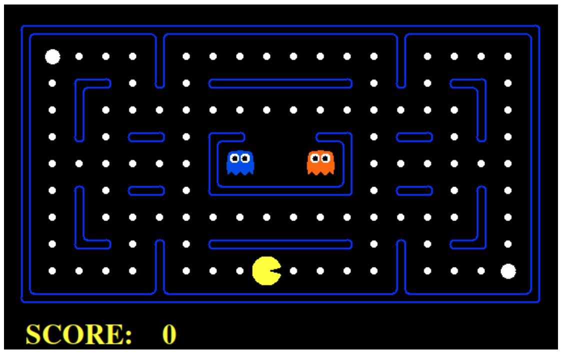
Notably, our methodology led to the encoding from above.

5 A Case Study: Pacman

Neufeld et al. [NBCG21] combined formal tools for normative reasoning with reinforcement learning (RL), with the aim of designing norm-sensitive autonomous agents. The authors developed a logic-based *normative supervisor* module which informs the reinforcement learning agent of compliant actions it could take. The agent then chooses an action complying with the norms in force in a given situation, and a least evil action in case there is no such action. Their approach allows to deal with complex normative systems, conflicting obligations or situations where no compliance is possible. Norms and the current state of the agent’s environment are encoded in defeasible deontic logic [GORS13,GR08], which is a deontic logic with defeasible rules that specify typical correlations, such as that birds usually fly, and the SPINdle theorem prover is used to check norms compliance. Exceptions are encoded through so-called defeaters, e.g., if the bird is a penguin.

Neufeld et al. tested their framework on a reinforcement learning agent trained to play “ethical versions” of the Pacman game. This game, which simulates a closed environment with clearly defined (and simple) game mechanics, was already used as case study in [NBM$^+$19]. The starting point of the Pacman game can be seen in Figure 2. The aim for Pacman is to eat all the pellets placed on the maze. There are two ghosts (orange and

Fig. 2: Pacman



blue) 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 Pacman 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 on the image. In other words, 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 eating all pellets, with points being deducted depending on how long the game has lasted (the longer the more). Should Pacman collect all the pellets, he wins. Finishing the game quicker is beneficial as less points are deducted.

Two norm bases for this game, "Vegan" and "Vegetarian", were analyzed in [NBCG21] (the vegan version already in [NBM$^+$19], using RL alone). Vegan Pacman is not allowed to eat any ghost, while Vegetarian Pacman can eat the orange ghost (like it would be tofu, while the blue ghost is chicken). These norm bases consisting of simple "ethical constraints" are enforced upon Pacman implemented as an RL agent, using the theorem prover SPINdle for defeasible deontic logic.

We consider here an alternative realization of the normative supervisor, based on our norms encoding from above and using the DLV reasoner in the provided framework.

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

$$O(\neg eat(ghost)) \qquad \text{respectively} \qquad F(eat(ghost)),$$

using the operator F for prohibition.

Vegetarian Pacman: 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:

$$O(\neg eat(blue\_ghost)) \qquad \text{respectively} \qquad F(eat(blue\_ghost)).$$

Note that the DLV encoding of these norms is not trivial, as it is not clear which actions lead to Pacman eating a ghost.

We encode the norm bases by forbidding the Pacman agent 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 vegetarian normbase implements the weak constraints for the blue ghost, whereas the vegan version will implement the weak constraints for both ghosts.

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.

The scenarios that can precede Pacman eating a ghost 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). This gives us the three possibilities for their relative locations. We encoded the norms by taking into account the locations of the ghosts relative to Pacman and then forbidding Pacman from making movements that could lead to Pacman eating the ghost. The full DLV encoding of the Pacman agent is given in [Hat22], and further information about it is given in the Appendix (see Section C).

Benchmark Results

For the comparison of our encoding to Neufeld et al.'s in [NBCG21], we used the same setting. The reinforcement learning agent was trained on 250 games and then the normative supervisor was tested using 1000 test games, where each of them was starting in the same state (see Figure 2).

The results reported by Neufeld et al. (using SPINdle for defeasible deontic logic) in [NBCG21] were:

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

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

We used our methodology to encode the norm bases from [NBCG21]. It outperformed the original supervisors 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). The higher maximum score seen in Neufeld et al.'s result is most likely due to a game where both ghosts were eaten in their tests. In the vegetarian norm base, the reason for the lower rate of wins and hence lower average score is most likely that under our encoding Pacman prefers losing a game over eating a ghost.

6 Related Work

Multiple approaches to handle normative systems have been proposed in the literature. Some of those related to the multi-agent systems community can be found, e.g., in [ADL18a]. We will discuss below the approaches most similar to our work.

One of the earliest works on encoding normative systems is [SSK$^+$86]. There Sergot et al. encoded the British Nationality Act using logic programming. Their goal was to show that logic programming was capable of representing the complexities of statutory law. As [SSK$^+$86] does not reason about obligations it does not aim to find optimal ways to act under given norms, but rather aims to determine whether the British nationality act applies to certain individuals.

[SBD$^+$00] introduces syntax and semantics for reasoning about obligations and prohibitions among agents. Although the authors refer to deontic logic, the proposed way of dealing with conflicting obligations is to satisfy a maximal subset of obligations, without considering possible preferences among obligations (as e.g., in Plato's dilemma).

Kowalski and Satoh [KS18] utilised abductive logic programs to encode the notion of obligation and many paradoxes. Rather than trying to derive all optimal ways of fulfilling given obligations, they focused on finding a best model that satisfies given goals (that must be satisfied). Furthermore, they only encoded a subset of the paradoxes we consider, and it remains to be seen whether their approach would work for the whole set.

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]. In their work, agents are capable of a more human kind of reasoning, e.g., to decide whether violating an obligation should the worth the penalty.

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], and there are sophisticated tools (for Linear Temporal Logic, e.g., [ABE$^+$18]) that effectively combine them with reinforcement learning. However, temporal logic cannot handle all the intricacies of normative reasoning, see the discussion in [ADL18b, NBC22].

A different ASP approach is given by deontic logic programs, as can be seen in [GA12]. The authors allow atoms in rules to take the form of complex SDL formulas and have found some application. They require an understanding of the embedded logic, whereas our presented methodology can be used without a deeper understanding of deontic logics.

7 Conclusion

Starting with the analysis of well known deontic paradoxes, we introduced a methodology to encode normative systems in ASP, using DLV as the system of choice. The selection of paradoxes proved to be very important, as omitting certain paradoxes leads to not addressing important features of normative systems. As an example, the analysis of the *Fence Paradox* allowed us to distinguish between Contrary-to-duty (CTD) obligations and exceptions, a well-known problem pointed out in [PS96] (CTD-obligations should not be active when an exception to the obligation is given).

The main advantages of our approach are the availability of optimized tools (ASP solvers) and the simplicity of the encoding. There is a clear cut common core that is supplemented with defined ways for encoding different kinds of obligations. The approach, described for DLV in this work, is also easily transferable to other ASP solvers, e.g. to clasp [BDRS15].

The main weakness of our method is that encoding complex normative systems can lead to extremely large programs (although this may be inevitable). Furthermore, maintenance obligations that require the agent to maintain certain conditions might be complicated to encode for complex examples. In such cases taking particular actions might indirectly lead to failure of maintenance. As an example for a maintenance obligation, consider: "See to it that the child stays dry." For such an obligation it does not suffice to not wet the child, but one must 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. For such normative systems, the use of approaches extending ASP with temporal logics (e.g., [GMD13]) may be preferable. Encoding these normative systems also requires knowledge about the underlying domain, as conflicts between actions (e.g., non-concurrency) need to be encoded as well.

Another weakness of our approach is that all encoded obligations are comparable by their associated weights and priorities. While we did not run into problems with our methodology due to this, there may be cases where optimal answer sets encoding solutions to paradoxes should be incomparable. 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, e.g. clingo [GKKS19] or dlvhex [EGI^{+}18]. In the case of Pacman, such a software might have to interpret how the obligation to not eat a ghost could be fulfilled. Another direction for future work is to look into encoding other Deontic Logics, e.g., with dyadic deontic operators.

Acknowledgment We thank the reviewers for their comments to improve the presentation of this paper.

References

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.

ABE$^+$18. Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proc. AAAI*, pages 2669–2678, 2018.

ADL18a. 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.

ADL18b. Natasha Alechina, Mehdi Dastani, and Brian Logan. Norm specification and verification in multi-agent systems. *Journal of Applied Logics*, 5(2):457, 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.

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.

BET16. Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming: An introduction to the special issue. *AI Mag.*, 37(3):5–6, 2016.

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.

BKI19. Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme*. Springer Fachmedien Wiesbaden, 6 edition, August 2019.

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.

CZ97. Alexander Chagrov and Michael Zakharyaschev. *Modal Logic*. Clarendon Press, 1997.

EGI$^+$18. Thomas Eiter, Stefano Germano, Giovambattista Ianni, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. The DLVHEX system. *Künstliche Intell.*, 32(2-3):187–189, 2018.

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.

GKKS19. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.

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.

Hat22. Christian Hatschka. Representing normative reasoning in answer set programming using weak constraints. Master's thesis, Technische Universität Wien, 2022.

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.

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.

NBC22. Emery Neufeld, Ezio Bartocci, and Agata Ciabattoni. On normative reinforcement learning via safe reinforcement learning. In *Proceedings of PRIMA 2022*, 2022.

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.

NBM$^+$19. Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R. Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. Teaching AI agents ethical values using reinforcement learning and policy orchestration. In *Proc of IJCAI: 28th International Joint Conference on Artificial Intelligence*, 2019.

PS96. Henry Prakken and Marek J. Sergot. Contrary-to-duty obligations. *Stud Logica*, 57(1):91–115, 1996.

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.

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.

vW51. Georg Henrik von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

A Appendix: Encoding Different Obligations

We start by presenting how to encode different kinds of obligations.

A.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.

A.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:

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

A.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.

A.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 modeled using auxiliary predicates. Using such an auxiliary predicate, called *Exception* in the next example, such an obligation can be encoded in the following way:

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

A.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:

$$\begin{gathered}:\sim -O(\mathit{obl}1).\ [1:2]\\ :\sim -Do(\mathit{obl}1), -O(\mathit{obl}2).\ [1:2]\end{gathered}$$

Note that $\mathit{obl}1$ and $\mathit{obl}2$ 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:

$$\begin{gathered}:\sim -O(\mathit{obl}1),\ \ \mathit{not}\ \ e.\ [1:2]\\ :\sim -Do(\mathit{obl}1), -O(\mathit{obl}2),\ \ \mathit{not}\ \ e.\ [1:2]\end{gathered}$$

A.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 $O(\mathit{obl}1)$ to be the more important obligation and $O(\mathit{obl}2)$ the less important one. This can be formulated in the following way:

$$\begin{gathered}:\sim -O(\mathit{obl}1).\ [1:3]\\ :\sim -O(\mathit{obl}2).\ [1:2]\\ :- Do(\mathit{obl}1), Do(\mathit{obl}2).\end{gathered}$$

or alternatively:

$$\begin{gathered}:\sim -O(\mathit{obl}1).\ [2:2]\\ :\sim -O(\mathit{obl}2).\ [1:2]\\ :- Do(\mathit{obl}1), Do(\mathit{obl}2).\end{gathered}$$

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 modeled by adding the following code to the common core:

$$
\begin{aligned}
&:\sim -O(obl3).\ [3:2]\\
&:\sim -O(obl2).\ [2:2]\\
&:\sim -O(obl1).\ [1:2]\\
&:- Do(obl1), Do(obl3).\\
&:- Do(obl2), Do(obl3).
\end{aligned}
$$

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.

B Appendix: An Additional Case Study

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:

O1 It is obligatory to stop if the traffic light is red.
O2 It is obligatory to not impede the traffic flow (by stopping), unless to let a car merge.
O3 It is obligatory to move out of the way when an ambulance approaches.
O4 If you drive during winter it is obligatory to either have winter or all-season tires.
O5 It is obligatory to not cause any damage.
O6 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.)
O7 If one causes damage, it is obligatory to drive directly to the next police station to make a damage report.
O8 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.

- O1 is a derived obligation: the obligation to stop is derived when the traffic light is red. Likewise, O3 and O8 are derived obligations.
- O2 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.
- O4 is both a derived and a disjunctive obligation. Should the antecedent be fulfilled, one part of the obligation must be satisfied. Note that not both parts of the obligation can be fulfilled, as one cannot have simultaneously winter tires and all-season tires.
- O5 is a regular obligation, with no additional properties.
- O6 consists of a conjunction of obligations with an exception, and thus belongs like O4 to multiple categories.
- O7 is a CTD obligation that is active when violating O5.

Step 2: Next, we look at pairs of obligations that can't be fulfilled simultaneously.

- O1 and O2 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. We can see O2 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.
- O1 and O3 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.
- O2 and O8 are incompatible, as giving first aid requires stopping the car. Here the obligation to give first aid should be prioritised.
- For the same reason O3 and O8 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).
- O7 and O8 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.

Incompatible combinations of more than two obligations include here always one of the pairs above (in general, the latter may not be always warranted).

Summarizing the statements above, we obtain the following preferences, where $O_i \succ O_j$ means that O_i is preferred over O_j:

$$O_1 \succ O_2, \quad O_3 \succ O_1, \quad O_8 \succ O_2, \quad O_3 \succ O_8, \quad O_8 \succ O_7.$$

Step 3: Using the above conflicts and priorities, we can derive the following weights and levels for the weak constraints corresponding to the obligations:

O1	O2	O3	O4	O5	O6	O7	O8
[1:3]	[1:2]	[1:4]	[1:2]	[1:2]	[1:2]	[1:2]	[1:3]

Note that an obligation 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. In addition to the predicates in the common core, we use the following predicates:

– *Redlight* means that a red traffic light is active in front of the agent.
– *Winter* means the season being winter.
– *Theft* means that the agent is in possession of proof of theft of his drivers license and/or registration.
– *Licenses* is an auxiliary predicate for formulation of O6, as described in Section A.3.

Furthermore, the following constants are used in the encoding:

– *merge*, *emergency_vehicle*, and *medical_emergency* are events that can happen. Specifically,
  • *Happens(merge)* means a car tries to merge into the lane that the agent is on.
  • *Happens(emergency_vehicle)* means an ambulance (with active emergency lights) approaches the car.
  • *Happens(medical)* means a medical emergency happens close to the agent.
– *stop*, *move*, *equip_winter*, *damage*, *carry_license*, *carry_registration*, *drive_police*, *give_first_aid* are actions to be reasoned about. Specifically,
  • *Do(stop)* means the car is stopped.
  • *Do(move)* means the car needs to move out of the way.
  • *Do(equip_winter)* resp. *Do(equip_allseason)* means the car is equipped with winter tires resp. all-season tires. While arguably tire equipment is more of a state than an action, we will use the action view for this example.
  • *Do(damage)* means the agent causes damage.
  • *Do(carry_license)* means the agent has his driver's license with him.
  • *Do(carry_registration)* means the agent has the car's registration with him.
  • *Do(drive_police)* means the agent drives straight to the next police station.
  • *Do(give_first_aid)* means the agent gives 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 as shown in Figure 3a, following our methodology. However, recall that two obligations are combinations of different kinds of obligations:

First, O4 is a combination of a derived obligation and a disjunctive obligation. As such, we are able to encode it in the following way:

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

This weak constraint can only be violated if *Winter* is true. For answer sets where *Winter* is true, it 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.

Fig. 3: Encoding for the driving scenario

$:\sim Redlight, -O(stop).$ [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]

(a) obligations

$:- Do(stop), Do(move).$
$:- Do(drive\_police), Do(give\_first\_aid).$

(b) conflicting actions

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

(c) action constraints and declarations

Second, O6 is a combination 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 and an exception on top as usual:

$$Licenses :- O(carry\_license), O(carry\_registration).$$
$$:\sim not\ \ Licenses, not\ \ Theft.\ [1:2]$$

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 from Step 2 are encoded, shown in Figure 3b.

Step 5: Finally, the rules and facts about actions in Figure 3c are added. The first constraint excludes a proof of theft 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. Furthermore, a constraint prohibits the agent from giving first aid without a medical emergency happening (as this is not possible). Finally, all actions to be reasoned about are declared 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:

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

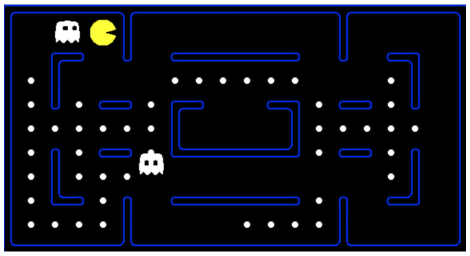
As expected the agent is obligated to give first aid rather than driving directly to the police station.

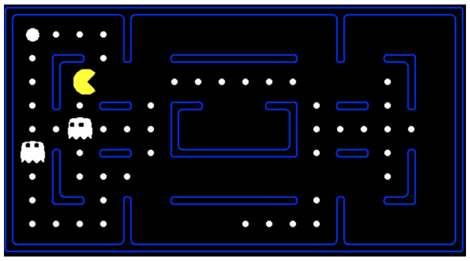
C Appendix: Encoding the normbases for Pacman

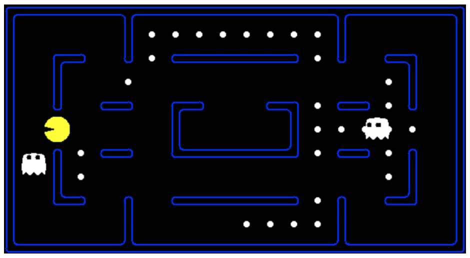
Let (x_1, y_1) be Pacman's coordinates and (x_2, y_2) the coordinates of a frightened ghost right before it is eaten. We start by looking at the three possible states that could precede Pacman eating a ghost:

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 4a.
   There are multiple options that can lead to the ghost being eaten in this situation. Either Pacman stops and the ghost moves into Pacman's 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.

Fig. 4: Pacman scenarios


(a) an example for case 1


(b) an example for case 2


(c) an example for case 3

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 4b. 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.)
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 4c.

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. We will only explain how the norms would be encoded if those functionalities worked.
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 prohibition. 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.

By abstracting the above information the vegan and vegetarian norm base can be encoded. The encodings can be found in [Hat22].