A preference-based approach to machine ethics for automated planning

Martin Jedwabny

▶ To cite this version:

Martin Jedwabny. A preference-based approach to machine ethics for automated planning. Computer Science [cs]. Universite de Montpellier, 2022. English. NNT : . tel-03923321v1

HAL Id: tel-03923321
https://hal.science/tel-03923321v1

Submitted on 4 Jan 2023 (v1), last revised 13 Apr 2023 (v3)

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale I2S

Unité de recherche LIRMM

A preference-based approach to machine ethics for automated planning

Présentée par Martin JEDWABNY
Le 2 décembre 2022

Sous la direction de Madalina CROITORU et Pierre BISQUERT

Devant le jury composé de

Madalina CROITORU, PU, Université de Montpellier, Montpellier, France	Directrice
Pierre BISQUERT, CR, INRAE, Montpellier, France	Co-encadrant
Jean-Gabriel GANASCIA, PU, Sorbonne Université, Paris, France	Rapporteur
Felipe MENEGUZZI, PU, University of Aberdeen, Aberdeen, Écosse	Rapporteur
Aurélie BEYNIER, PU, Sorbonne Université, Paris, France	Examinatrice
Anne LAURENT, PU, Université de Montpellier, Montpellier, France	Examinatrice

UNIVERSITÉ
DE MONTPELLIER

Abstract

Machine ethics is an uprising sub-field of artificial intelligence fueled by the interest and concerns about the deployment of automated agents in our everyday life. As these agents gain independence from human intervention and make decisions with possible impact on human welfare, real concerns are rising across domains.

Due to those reasons, various approaches have been proposed to imbue automated agents with ethical considerations. Several research currents have developed models stemming from psychology and philosophy in an effort to adapt decision-making algorithms to consider ethical values so that the impact of agents on people is bounded and guided by these notions.

Most of these approaches consist of either reasoning and applying a set of well-known ethical restrictions, also known as principles (top-down), or inferring them based on carefully crafted datasets through learning algorithms (bottom-up).

In this thesis, we look at the problem of implementing these ethical principles in the context of tasks involving sequences of interdependent decisions, i.e: automated planning. We show how certain notions can be modeled using preference-based frameworks, as in top-down approaches, and how these preferences can be inferred from a corpus of data like bottom-up methodologies, to develop a hybrid approach that can be applied to planning problems. An implementation for each facet of our approach is provided in order to test our ideas in practical scenarios.

Resumé

L'éthique des machines est un sous-domaine en plein essor de l'intelligence artificielle qui suscite intérêt et inquiétudes, en particulier en ce qui concerne le déploiement d'agents automatisés dans notre vie quotidienne. À mesure que ces agents gagnent en indépendance vis-à-vis de l'intervention humaine et prennent des décisions susceptibles d'avoir un impact sur le bien-être humain, de réelles inquiétudes appraissent dans plusieurs domaines.

Pour ces raisons, diverses approches ont été proposées pour apporter les agents automatisés de considérations éthiques. Plusieurs courants de recherche ont développé des modèles issus de la psychologie et de la philosophie dans le but d'adapter les algorithmes de prise de décision pour tenir compte des valeurs éthiques afin que l'impact des agents sur les personnes soit délimité et guidé par ces notions.

La plupart de ces approches consistent soit à raisonner et à appliquer un ensemble de restrictions éthiques bien connues, également appelées principes (top-down), soit à les inférer sur la base d'ensembles de données soigneusement élaborés grâce à des algorithmes d'apprentissage (bottom-up).

Dans cette thèse, nous examinons la mise en œuvre de ces principes éthiques dans le contexte de problèmes impliquant des séquences de décisions, c'est-à-dire : la planification automatique. Nous montrons comment certaines notions peuvent être modélisées à l'aide de cadres formels basés sur les préférences, comme dans les approches 'top-down', et comment ces préférences peuvent être déduites d'un corpus de données comme les méthodologies 'bottom-up', pour développer une approche hybride applicable à la planification automatique. Un logiciel pour chaque facette de notre approche est fournie afin de tester nos idées sur des scénarios pratiques.

Contents

List of Figures

List of Tables

1 Introduction

Recent years of progress in artificial intelligence (AI) research have resulted in the development and widespread use of artificial agents to aid people in daily activities and perform diverse tasks without human intervention. However, the systematic introduction of automated agents in domains where decisions can impact the well-being of people or society as a whole, such as autonomous vehicles, has been the subject of much public concern. Indeed, numerous authors [Tolmeijer et al., 2020] have pointed out the necessity of aligning machines with our ethical values and ensuring that an agent's reasoning will be understood by humans if we strive to build systems we can trust. To ensure this, it is paramount that we develop agents that can comprehend and apply ethical reasoning, a highly complex endeavor due to its reliance on societal views and conflicting judgments across cultures.

This thesis presents original research at the intersection of **machine ethics**, the sub-field of AI that studies the alignment of agents with our ethical values [Anderson and Anderson, 2007], and **automated planning**, one of the major AI sub-fields, which deals with reasoning about sequences of actions and anticipating their outcome to fulfill some predefined objectives [Ghallab et al., 2004]. Among the main difficulties in the automation of machine ethics, in this thesis, we focus on three: *(i) aligning machines to ethical values*, *(ii) maintaining the ability to reason in the presence of conflicting ethical values*, and *(iii) learning priorities between them.* As we mentioned, artificial agents that can coexist in our society should be equipped with some kind of ethical understanding, and for this purpose, much research (see Chapter 3) has been dedicated to allowing machines to discern right from wrong. Nevertheless, the characterization of the rightness of action might not be sufficient in the presence of ethical conflicts. Different kinds of conflicts may arise from several possible sources, such as the plurality of ethical principles people use to make decisions [Brundage, 2014], most notable in the presence of sacred values [Tetlock et al., 2000], and the cross-cultural variation of ethical preferences [Awad et al., 2018]. For this reason, we will set out to develop a framework that can model a variety of ethical principles and combine them through levels of priority. Furthermore, since this prioritization should be highly dependent on the domain at hand and the continuously evolving views of society on morality, we will need to provide a mechanism to learn the underlying hierarchies of ethical values from past instances of human behavior.

To alleviate these concerns, the contribution of this thesis will consist of the development of an abstract preference-based formalism for deterministic planning that takes into account several ethical values and priorities between them, and a mechanism to infer these priorities. The goal is to provide a flexible framework that allows the modelization of well-known ethical theories and their combination via preferences.

This chapter is structured as follows. In Section 1.1 we introduce the general context of the thesis. Then, in Section 1.2 we discuss the issue of conflicting ethical theories and how they can be handled within our framework, this allows us to present the research problem alongside our contributions in this regard in Section 1.3. Finally, we conclude this chapter by highlighting the structure of the thesis.

1.1 Machine ethics and dilemmas

In the context of AI, an *ethical machine* should take into account societal considerations of what is 'correct' ethical behavior, in conjunction with other operational constraints, to make its behavior more acceptable. Thus, it poses two essential problems: deciding what ethical behavior is in a certain context, and determining how a machine should combine this information with the rest in the reasoning process.

Part of the reason it is difficult to determine what constitutes an ethical decision is that human ethical reasoning remains a largely discussed and complex topic. The 'trolley problem' introduced by [Foot, 1967] is an example of such complexity. The simplest version of this dilemma, called the Bystander trolley problem, which is depicted in Figure 1.1, can be described as follows:

> *A runaway trolley is on the way to kill five people tied to its current track. You are standing far away and next to a lever, which if activated, would change the trolley's path to a side track in which only one person lies. You have only two options, either to pull the lever or do nothing.*

A simple-sounding problem statement like this and its many variations continue to stir debate amongst philosophers to this date [Bruers and Braeckman, 2014]. A situation like this is called a dilemma because no

[1] https://nymag.com/intelligencer/2016/08/trolley-problem-meme-tumblr-philosophy.html

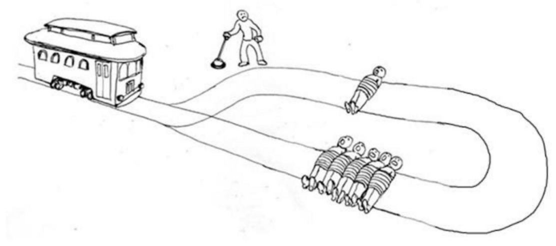


Figure 1.1: Trolley problem illustration[1].

option is perfectly ethical and one is forced to make a decision that would violate some ethical consideration. Instead of having a clear-cut answer, dilemmas serve to compare ethical principles, highlight faulty reasoning and refine them in the light of the problems they pose. Ethical dilemmas such as this highlight the kind of problems that arise when designing computational models for ethical principles and the need to establish priorities, which could depend on many societal factors and carry over to a system's designer.

From a philosophical standpoint, machine ethics can be seen as a form of applied ethics, i.e: the branch of moral philosophy concerned with practical real-life problems, for automated reasoning. Normative ethics, another of the three main branches along with applied ethics and meta-ethics, focuses on developing generalized principles to deduce what is right and wrong. Unsurprisingly, representing ethical principles inside the machine has proved to be a nontrivial task due to its reliance on human concepts and philosophical reasoning. Indeed, each principle demands analyzing choices from a particular perspective that is deeply tied to human values and judging them according to rules of conduct stemming from law, societal consensus, and our own personal experiences. Moreover, normative ethics can provide general guidelines on how to represent principles inside the machine, but they will always need to be adapted to its world representation and operational constraints. Previous research on machine ethics has begun implementing computational models to align machines with our ethical values [Tolmeijer et al., 2020] by adapting normative ethical principles to AI systems. However, ethical reasoning for automated systems can lead to different kinds of conflict. People seem to use varying ethical principles depending on the situation at hand [Brundage, 2014, Tetlock et al., 2000], or their culture [Awad et al., 2018], most notably in the presence of dilemmas. Furthermore, while some ethical theories are capable of comparing alternatives in such situations, many others reject every choice if none satisfies some of their precepts. For this reason, throughout this thesis, we will develop a general model for automated ethical reasoning that can deal with a plurality of ethical principles and values.

1.2 Research problem

Motivation The purpose behind this work relies on the current limitations of ethical reasoning for AI systems in the face of conflicting values and priorities. So far, the problem of designing an agent that is explicitly ethical, i.e: one that can reason and make decisions that depend on ethical considerations and possibly principles that are encoded in their system, has been addressed with three approaches (i) *top-down*: developing or adapting normative ethical principles to AI systems, (ii) *bottom-up*: inferring which choices people deem as ethical from a corpus of data, and (iii) *hybrid*: a combination of both.

Top-down methods are particularly useful when there is a considerable amount of structured domain knowledge and are typically designed using symbolic AI approaches. An advantage is that decisions can be justified using the underlying system and ethical considerations used to make decisions. On the other hand, they carry the typical limitations of the implemented theory because, in the effort of being general and characterizing the rightness of all actions under the same measure, they can hardly keep up with the types of exceptions people consider when making their decisions. That is, they are hard to apply to every situation without being inconsistent, or worse, rendering all decisions immoral. Additionally, they could require encoding various complex notions, such as possible consequences, causality and intentionality [Govindarajulu and Bringsjord, 2017], while there is no clear consensus about how to translate them into a computational model.

Then, bottom-up approaches build autonomous agents that can learn how to behave ethically through machine learning [Mitchell, 1997], the branch of AI that studies and develops systems that learn to perform tasks through past experience. The main difference with top-down approaches lies in letting the agent learn from a corpus of data, given by one or many information sources that indicate what is the appropriate ethical choice, instead of modeling a code of conduct. These methods are well-known to perform properly with big and diverse corpora of not perfectly consistent data, which is an advantage in ethically-nuanced domains. Indeed, these corpora typically suffer from inconsistencies between different information sources and even in the same source in very similar situations, due to the intricacies of ethical reasoning. Nonetheless, justifying that a decision is ethically correct using a bottom-up approach is more difficult due to the absence of an underlying ethical theory, which is a highly desirable quality in ethically-nuanced domains.

Hybrid approaches strive to solve the latter issue by proposing a middle ground between these two strategies: constructing a theory, guided by well-known principles and determining how to use it, for instance, by finding priorities between these principles, through learning algorithms. In summary, this approach offers the following advantages for machine ethics systems:

- By encoding ethical values and/or principles through a symbolic representation, it can provide a justification for its choices, and

- Because it learns from experience to apply ethical principles, it can adapt to inconsistencies of opinions from various stakeholders.

Although much research has been dedicated to top-down and bottom-up ethical systems, few frameworks have tried combining their ideas into a hybrid approach. Learning-based methods have repeatedly shown to be effective at replicating human behavior, but they seem to fall short in domains with ethical nuances due to their lack of reasoning transparency and understanding of ethics. On the other hand, top-down approaches to ethics will be inadequate whenever conflicting ethical views, principles, or exceptions take place. With this in mind, and to alleviate the aforementioned concerns, we will set out to develop a hybrid framework for ethical reasoning that retains the benefits of both top-down and bottom-up methods, so that it can deal with the types of conflicts we have identified.

Setting As discussed in the previous section, ethical reasoning is especially important in domains in which agents make decisions autonomously and continuously. Domains of this type will often require the agent to reason about the outcome of actions many steps in advance. Additionally, various ethical principles demand both the assessment of actions and consequences in the long run and the examination of certain interdependencies between them, such as causality [Berreby et al., 2018], to determine if a sequence of actions is the most ethically aligned to the target values.

For this reason, the setting in which we will center this thesis will be that of automated planning [Ghallab et al., 2004]. At a high level, planning is a very general and extensive field that allows the modeling of all sorts of problems that can be reduced to finding sequences of edges (denoting actions) in a graph where nodes represent sets of properties that characterize world states. AI planning as a setting is very extensive in the sense that it encompasses various subfields which require different representations, capabilities from the agent and algorithms to be solved. In this thesis, we will research ethics from a deterministic, single-agent context, also known as classical planning. The planning language we consider is PDDL [Fox and Long, 2003], a family of domain-independent languages based on first-order logic which is widely used in the context of deterministic planning.

While a great deal of literature has dealt with the problem of encoding ethical principles through AI planning, few hybrid approaches have been proposed that permit combining different principles. In conjunction with the generality of the planning setting, the maturity of the field as a whole, and the necessity to reason about evolving environments and the consequences of actions, we have chosen it as the appropriate setting for the development of our work.

In what follows, we will explain how this thesis addresses the research problem we formulated in this section by proposing a hybrid framework for ethical reasoning using a preference-based approach.

1.3 Research question and contributions

Given the increased autonomy and rapid deployment of automated systems, and the varied normative ethical theories describing potentially disparate points of view for the same dilemma, the problem of conflicting ethical assessments is likely to arise. Simultaneously, societal views on ethics have proven to be ever-evolving, and as such, developing ethical systems that can adapt to their perspectives of ethics is paramount. Therefore, the research question we want to answer in this thesis is:

Research Question

How can we align machines to our ethical standards when making decisions in the face of conflicting principles and values?

It has been shown that various ethical principles can be encoded in computational planning models. At the same time, a great deal of research has been developed to accommodate preferences in deterministic planning models, particularly for the International Planning Competitions (IPC). Our hypothesis is that a preference-based approach is a natural method to deal with the kinds of conflicts that stem from ethical reasoning. We argue that preferences can help combine the judgment of several ethical principles in a unified manner. Furthermore, little research has addressed using preferences for this purpose. Thus, our research problem can be reformulated into the following subset of more precise research questions:

Research Questions

- *How can we model and combine well-known ethical principles through preferences in a deterministic planning setting?*
- *Can we provide an efficient way of computing ethically optimal plans using our model?*
- *In which way can we elicit preferences from a corpus of data to determine priorities between ethical values inside our model?*

Let us now see how the contributions of the thesis address all three of the previously mentioned research questions.

1.3.1 Contribution 1: Modeling and combining ethical principles through preferences

Most of the existing top-down implementations have concentrated on determining which actions are right or wrong through ethical theories. However, it seems to be the case that depending on situations, people seem to apply different ethical theories or combinations of concepts from them, for instance, in the presence of sacred values [Tetlock et al., 2000]. Accordingly, we believe that an ethically-aligned agent should adapt to these different situations. Therefore, our first contribution will be to develop an extended mathematical formalization of PDDL that allows agents to represent ethical features and principles, and compare them through preferences. Choosing PDDL as our base language means that a great deal of research on planning will be easily adaptable to our work. By introducing a model for ethical preferences, our framework goes further than classifying actions as right or wrong. We argue that by working with preferences we avoid arriving at a situation in which every plan is rejected because they are not perfectly ethical. The benefit of ensuring that an ethically preferred plan is computed whenever a plan exists is that: (i) certain domains might arrive at a scenario that always demands the machine to do something, and (ii) the agent can always reject a plan after it has been computed and change its objective if the plan does not satisfy certain constraints.

Furthermore, we show the different benefits of this approach. Namely, we argue that separating the ethical aspects into different levels of priority is useful in ethical domains. We also show how various ethical theories may be adapted to our framework and can be combined by using different preferences.

1.3.2 Contribution 2: Computing ethically optimal plans

By addressing ethical reasoning in the planning setting, one of the main challenges we need to solve is finding plans. It has been shown that classical planning can be intractable even when considering severe restrictions [Bäckström and Nebel, 1995]. However, by developing numerous heuristic strategies and complex algorithms, research has built highly performant systems that can find plans for problems with enormous state spaces even when considering preferences [Gerevini et al., 2009].

For this reason, we will show how ethically optimal plans can be computed by transforming our model of ethical preferences into simple state utilities. By employing this translation procedure, problems encoded using our extended PDDL model can be solved using existing state-of-the-art planning technology. In order to do this, we introduce a *valuation* function, based on [Feldmann et al., 2006] that assigns a numerical value to plans.

Then, we implement our mathematical formalization in actual PDDL code as an extension of the programming language. Consequently, we develop two different implementations of the mentioned translation procedures we have made publicly available: (i) from ethical preferences into soft goals, i.e: final state utilities, and (ii) from ethical preferences into action costs. By implementing the translation routine into two different versions of PDDL, we maximize the scope of planners that can be used to solve planning problems with ethical preferences. And finally, we test the computational efficiency of our approach using various state-of-the-art planners and hard problems from the IPCs.

1.3.3 Contribution 3: Eliciting ethical preferences

Our ethical model for planning assigns features to plans, which are used to compare them on ethical terms through priority levels. Eliciting the opinion of non-experts can be problematic as people might be inconsistent with their choices according to certain ethical principles even when faced with barely dissimilar situations, due to the intricacies of ethical reasoning. Quite often, ethical domains require the opinion of experts in order to determine what is ethical in a domain. However, even experts regularly disagree on their criteria. This is why providing a mechanism that learns to reason ethically according to an elicitated corpus of opinions is essential.

As our final contribution, we will investigate how the ethical preferences we developed throughout this thesis can be learned from datasets. In doing so, we will shed light on how our framework can be used as a hybrid ethical system. We chose to perform this method using what is known as probabilistic logic, as it combines naturally with our logic-based formalization of PDDL and can provide a full trace of its reasoning through logic rules, allowing people to understand the agent's choices. With this in mind, we will describe an encoding using probabilistic logic, which can be used to learn the ethical preferences we introduce in this thesis. And lastly, we will showcase the practicality of our approach with a case study and provide preliminary experimental results.

1.4 Thesis Structure

The structure of this thesis is as follows:

Chapter 2. This chapter introduces necessary preliminaries related to logic and automated planning. We will start by describing a formalization of first-order logic programs and how queries can be computed. Then, we will introduce probabilistic first-order logic, which extends first-order logic with probabilistic annotations, and parameter learning, which will be used in Chapter 6 to infer preferences between ethical values. Lastly, we will provide a formalization for classical planning that will be used as a basis throughout this thesis.

Chapter 3. In this chapter we present the state-of-the-art of machine ethics implementations. We start by giving a comprehensive introduction to the history and motivations of the field. Then, we will describe a few useful taxonomies characterizing ethical agency. We will also glance over different sources of codes of conduct that have been implemented. And finally, we will examine various state-of-the-art implementations of machine ethics through the lens of the mentioned taxonomies.

Chapter 4. This chapter handles the first contribution of this thesis. We introduce an extension of PDDL to assign ethical features to plans, demonstrate how this framework captures different well-known ethical principles, and adapt a preference-based model to combine these principles using different levels of priorities under a unified framework. This chapter builds upon our work published in [Jedwabny et al., 2021a].

Chapter 5. In this chapter, we address our second contribution. We demonstrate how our ethical planning problems can be translated into utilities by using soft goals. Then, we provide an overview of the two implementations we developed for the ideas discussed in the previous section. We also describe various experiments we formulated to test the computational efficiency of our approach. As in the previous chapter, this also builds upon the work published in [Jedwabny et al., 2021a].

Chapter 6. Here, we tackle the third and last of our contributions. We provide a method to infer preferences between ethical values by learning a set of probabilistic annotations for a probabilistic logic program. We describe our implementation, which we adapted to be compatible with the planning model we developed in the previous chapters. This chapter continues some of our work published in [Jedwabny et al., 2021b].

Chapter 7. This chapter concludes, summarizes our contributions and presents several interesting future research problems based on possible extensions of this work.

Appendix A. We use this Appendix to briefly summarize some concepts related to computational complexity that we will use throughout the thesis.

Appendix B. In this second Appendix we list the PDDL code of the overarching example of Chapter 4 and the output of the translation routines we describe in Chapter 5.

Appendix C. This last Appendix contains the full implementation of the ethical rank learning problem and the example case study described in Chapter 6 using the Problog language.

2 Background notions

In this chapter, we lay down the fundamental notions concerning the fields of logic programming and automated planning, upon which the rest of this thesis will be built. As such, we will focus on introducing these subjects under the scope of our work, giving an overview of the literature surrounding them as far as our purposes require.

2.1 Logic programming

We will start this chapter by introducing some key concepts related to logic programming. Knowledge representation based on first-order logic programming is popular amongst planning literature and will serve as a basis for its formalization and the development of Chapters 4 and 5. Additionally, we will describe a model for probabilistic logic programming based on [De Raedt et al., 2007], which extends logic programming with probabilistic annotations and a method for learning some of these probabilities based on evidence, which will be used in Chapter 6.

2.1.1 First-order logic programming

First-order logic programming [Apt, 1990, Lloyd, 1994] is a declarative paradigm based on formal logic, that views computing as a procedure to find a proof for a logical theory, i.e: a set of logical sentences. A program encodes this logical theory and the proof is typically produced - if finding one is possible - using the resolution principle [Robinson, 1965], which provides a single rule of deductive inference that is sound and complete for proving statements constructed using the syntax of logic programs. The most widespread syntax for logic programming is known as Horn logic [Horn, 1951], a Turing complete [Tärnlund, 1977] restricted (in the form of rules it allows) version of first-order predicate logic. In this thesis, we will focus on Prolog [Clocksin and Mellish, 2003], a widely-known logic programming language that extends Horn logic, and employ its syntax instead of the one used classically in formal logic.

2.1.1.1 Syntax

We build upon a first-order logic language $\mathcal{L}$ composed of:

- *Variables* $\{X, Y, \ldots\} \in \mathcal{V}$,
- *Function* symbols $\{f, g, \ldots\} \in \mathcal{F}$, and
- *Predicate* symbols $\{p, q, \ldots\} \in \mathcal{P}$

Each predicate and function symbol comes with an arity which represents the number of arguments it takes and is denoted $arity(p) \in \mathbb{N}_0$ for predicates and $arity(f) \in \mathbb{N}_0$ for function symbols. We use a special set $\{a, b, \ldots\} \in \mathcal{C}$ called *constants* for function symbols of 0 arity.

A *term* t is a constant, a variable, or a functor, that is, a construct of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ has arity n and every t_i is a term. An *atom* is a construct $p(t_1, \ldots, t_n)$ composed of a predicate p of arity $n \in \mathbb{N}_0$, applied to terms $t_1, \ldots, t_n$. A *literal* is either an atom A, or its negation, denoted $\neg A$.

Definition 2.1 (First-order logic rule)**.** A *rule*, also called a definite clause, is a construct of the form:

```
H :- B$_1$,...,B$_n$.
```

$$\texttt{H :- B}_1\texttt{,...,B}_n\texttt{.}$$

Composed of a *head* atom H and a conjunction of *body* atoms $B_1, \ldots, B_n$.

A *fact* is a rule with an empty body, i.e: 'H :– .', which can be denoted simply as 'H.'. We say that a rule is *grounded* when it contains no variables. Finally, a logic *program* P is a set of rules.

2.1.1.2 Semantics

Having defined the syntax of programs, we turn our attention to their semantics. The set of all ground atoms that can be produced by using the predicates of $\mathcal{L}$, also called the *Herbrand base* is denoted as $\mathcal{B}$. Given a program P, we can associate the language $\mathcal{L}(P)$ and base $\mathcal{B}(P)$ using only the symbols appearing in P. For the time being, we will assume them to be identical to $\mathcal{L}$ and $\mathcal{B}$, respectively. An *interpretation* $I \subseteq \mathcal{B}$ is a subset of the Herbrand base of the program P (i.e. any set of ground atoms), while a *model* of P is an interpretation that satisfies every of its rules H :– B$_1$,...,B$_n$., i.e: it holds that $\forall \hat{X}.H \vee \neg(B_1 \wedge \cdots \wedge B_n)$ where $\hat{X}$ is the list of all the variables appearing in the rule.

Satisfying a rule implies that whenever the body holds, then the head does as well. Models inform us of what sets of ground atoms satisfy the rules of the program.

The following definitions help determine which ground atoms from the base hold in every model.

Definition 2.2 (Substitution)**.** A *substitution* $\theta = \{X_1/t_1, \ldots X_k/t_k\}$ is a mapping from variables to other terms. Applying a substitution θ to an atom A is denoted as $A\theta$ and it replaces the variables in the domain of θ with their corresponding terms.

Similarly, applying a substitution θ to a rule $r \in P$ replaces the variables of each of its atoms in the domain of θ with their corresponding terms and is denoted $r\theta$. In the case $A\theta$ or $r\theta$ contain no variables, we say that the operation is a *grounding* and we call $A\theta$ $(r\theta)$ a ground atom (rule). Given two atoms A and B, one substitution θ *unifies* them if and only if $A\theta = B\theta$.

Furthermore, the *grounding of logic program* P, denoted $\boldsymbol{ground}(P)$, is the set of all ground rules resulting from every possible grounding substitution of each rule $r \in P$.

Definition 2.3 (Immediate consequence operator)**.** If P is a ground logic program, the *immediate consequence operator* is defined as:

$$T_P(I) = \{\ \mathrm{H} : \mathrm{H} :\!-\ \mathrm{B}_1, \ldots, \mathrm{B}_n.\ \in P\ \wedge\ \forall i \in [1, n]\ B_i \subseteq I\}$$

$T_P(I)$ captures which atoms can be derived from I using the rules in P. In the general case, i.e: when P has rules that contain variables, we define $T_P(I) = T_{ground(P)}(I)$. Then, we can define a sequence $T_P^0 = \emptyset$, $T_P^{i+1} = T_P(T_P^i)$. Because the operator is monotone, due to the Knaster-Tarski theorem [Leśniak, 2012], it will have a fixpoint T_P^∞.

Having defined the building blocks of the semantics, we can now specify the consequence operator '$\models$' that indicates when a logic program entails a certain ground atom.

Definition 2.4 (Consequence)**.** A ground atom A is a *consequence* of the program P, noted $P \models A$ if and only if $A \in T_P^\infty$. Similarly, $P \models \neg A$ if and only if $A \notin T_P^\infty$.

As it turns out, this fixpoint is the same as the least Herbrand model [Van Emden and Kowalski, 1976], i.e: every model M of P satisfies $M \subseteq T_P^\infty$.

Given a logic program P and A a ground atom, the *query* problem is that of determining whether $P \models A$. This can be easily extended for the case in which A is a conjunction of ground atoms or non-ground atoms.

Example 2.5 (Ancestor)**.** Consider the following logic program P:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
```

```
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
father(a,b).
mother(c,b).
father(b,d).
mother(d,e).
father(f,e).
```

The rules model the *ancestor* relation between family members using the *parent* predicate in the first rule and the *ancestor* predicate in the second in a recursive manner. The third and fourth rules state that a *parent* is either a *father* or a *mother*, while the rest are ground facts, representing information about the family tree that is already known.

To determine which atoms belong to T_P^∞, we should ground P according to the previous definitions. This grounding is essentially the result of replacing separately in each rule the variables X, Y, Z with every possible ground term in $\mathcal{L}$, which in this case would be a, b, c, d, e, or f.

Then, we have that:

$$\begin{aligned}
T_P^0 &= \emptyset \\
T_P^1 &= \{\texttt{father(a,b), mother(c,b), father(b,d),} \\
&\quad \texttt{mother(d,e), father(f,e)}\} \\
T_P^2 &= T_P^1 \cup \{\texttt{parent(a,b), parent(c,b), parent(b,d),} \\
&\quad \texttt{parent(d,e), parent(f,e)}\} \\
T_P^3 &= T_P^2 \cup \{\texttt{ancestor(a,b), ancestor(c,b), ancestor(b,d),} \\
&\quad \texttt{ancestor(d,e), ancestor(f,e)}\} \\
T_P^3 &= T_P^2 \cup \{\texttt{ancestor(a,d), ancestor(c,d), ancestor(b,e)}\} \\
T_P^4 &= T_P^3 \cup \{\texttt{ancestor(a,e), ancestor(c,e)}\}
\end{aligned}$$

It is simple enough to check that $T_P^4 = T_P(T_P^4) = \ldots = T_P^\infty$. Thus, by iterating the consequence operator, we generate the tree of ancestors denoted by the program P.

Of course, generating T_P^∞ is computationally expensive and not the only method to determine if $P \models A$. Many strategies and algorithms have been developed in the literature, most notably following the resolution principle [Robinson, 1965]. The most popular variation of the resolution principle is called *selective linear definite* (SLD) resolution [Kowalski and Kuehner, 1971] and is the one used by Prolog. Here, we will not present SLD resolution, however, as it is not essential for the development of this thesis and it has been largely covered in first-order logic programming literature.

2.1.1.3 Complexity

For logic programs, the time complexity of querying is given in terms of the number of operations it takes to determine if $D \cup P \models A$ for a set of ground atoms D, called the input database, a program P and another set of atoms A, the subject of the query. We refer the reader to Appendix A for general complexity notions and previous literature [Dantsin et al., 2001] for an extensive presentation of the computational cost of Datalog and general logic programs. Briefly summarizing the results, for Datalog:

- The complexity of checking $D \cup P \models A$ for a fixed Datalog program P and variable input database D and ground atoms A, called the *data complexity*, is PTime-complete.
- In the case D is fixed, while P and A are not, known as the *program complexity*, it is EXPTime-complete.
- Lastly, when all D, P and A are not known as the *combined complexity*, it is EXPTime-complete.

Having presented Prolog-based logic programming, we can now introduce probabilistic logic programming, an extension that allows handling noisy information in the form of probabilistic annotations that can be assigned to facts and rules.

2.1.2 Probabilistic logic programming

In recent years, numerous extensions of logic programming have been devised to capture probabilistic logic. These extensions generally allow reasoning about uncertain information by attaching probability annotations to facts and rules. While this approach allows reasoning in noisy domains with the power of logical reasoning, as we will see, it also typically suffers from additional computational costs to account for the possibility that certain pieces of information might or might not be true. Some examples of such systems include PRISM [Sato and Kameya, 2001], Markov logic networks [Richardson and Domingos, 2006], Bayesian logic networks [Kersting and Raedt, 2001], Probabilistic Horn abduction [Poole, 1993] and Problog [De Raedt et al., 2007]. For a more in-depth description of various frameworks and their semantics, we refer the reader to [De Raedt and Kimmig, 2015, Riguzzi and Swift, 2018].

Here, we will focus on ProbLog [De Raedt and Kimmig, 2015], a probabilistic first-order logic language that extends the notions presented before with simple probabilistic annotations. We chose this language due to its simplicity of representation, the fact that its semantics are well established in the probabilistic logics community and because the implementation behind it is relatively mature and easily accessible.

2.1.2.1 Syntax

Essentially, Problog was developed as a probabilistic extension of Prolog, mentioned in Section 2.1. As such, Problog also uses definite clauses, which we simply called rules, but extends them with *probabilitic annotations*. These annotations account for the (un)certainty of the information and permit reasoning in noisy domains.

Definition 2.6 (Probabilistic logic rule)**.** A *probabilistic logic rule*, or *Problog rule*, is a construct of the form $p_i :: r_i$ where $p_i \in [0, 1]$ denotes a probability and r_i is a (first-order) logic rule. In other words:

```
p_i :: H :- B_1,...,B_n.
```

Which is composed of a probability $p_i \in [0, 1]$, a *head* atom H and a *body* $B_1, \dots, B_n$ as defined in Section 2.1.

If the rule contains no body atoms, i.e: $n = 0$, we will call it a *probabilitic fact* and write it as ‘p$_i$ :: H.’. Also, in the case the rule is not probabilistic, i.e: $p_i = 1$, we can also denote it as a normal Prolog rule without an annotation: ‘H :– B$_1$,...,B$_n$.’.

Then, a *ProbLog program* $P = \{p_1 :: r_1, \dots, p_n :: r_n\}$ consists of a finite set of probabilistic rules.

2.1.2.2 Semantics

As we briefly mentioned before, the semantics of Problog programs is based on well-known semantics for probabilistic logics, which is known as distribution semantics [Sato, 1995]. It is also the semantics used for other systems [Poole, 1993, Poole, 1997, Sato and Kameya, 1997]. Instead of checking whether a logic program P has as a consequence a set of ground atoms A, denoted $P \models A$, as we saw before in the context of first-order logic programming, this setting measures a probability $Pr(P \models A) \in [0, 1]$ of the set of atoms being a consequence of the program, to take into account the possibility of noisy information in the form of probabilistic annotations.

According to these semantics, a Problog program $P = \{p_1 :: r_1, \dots, p_n :: r_n\}$ defines a probability distribution [Devore, 2011] over the groundings of rules in P without probabilities. In other words, let $\theta_{i,1}, \dots, \theta_{i,m_i}$ be the finite[1] set of groundings for each rule $p_i :: r_i \in P$, then P defines a probability distribution over the *possible worlds* $L \subseteq L_P$ of P, where the possible worlds are defined as $L_P = \{\theta_{1,1}r_1, \dots, \theta_{1,m_1}r_1, \dots, \theta_{n,1}r_n, \dots, \theta_{n,m_n}r_n\}$. That is, the possible worlds are the subsets of all the groundings of the original program P without probabilistic annotations. Then, P defines a probability distribution over the possible worlds $L \subseteq L_P$ as follows.

Definition 2.7 (Probability of a possible world)**.** Let $L \subseteq L_P$ be a subset of the possible worlds of P, then the probability of L given P is:

$$Pr(L \mid P) = \prod_{r_i \in L} p_i \prod_{r_i \in L_P \setminus L} (1 - p_i)$$

[1]We will assume the groundings are finite and correspond to the terms that appear on the program P.

Essentially, $Pr(L \mid P)$ measures the probability of non-probabilistic rules in L being true according to the probabilistic annotations in P.

Moreover, ProbLog defines the *success probability* of a ground query A (i.e. finite conjunction of ground atoms) as the overall probability that a random subset $L \subseteq L_P$ has as a consequence A[2]:

Definition 2.8 (Success probability of a query)**.** Given a Problog program P and a query A, the success probability of A according to P is:

$$Pr_s(P \models A) = \sum_{\substack{L \subseteq L_P \\ L \models A}} Pr(L \mid P)$$

Naively, the success probability of a query A for P can be calculated by considering every subset L of L_P, checking whether $L \models A$ and summing the probabilities of those ground programs according to $Pr(L \mid P)$. Current Problog implementations [De Raedt et al., 2007] avoid doing this as much as possible by using different optimization techniques, for instance, by using binary decision diagrams [Bryant, 1986].

While in the case of non-probabilistic logic programs it suffices to check that A is a consequence of a single grounding of the program P, the probabilistic setting forces to take into account all possible combinations of rules, calculating their probability and summing the ones that have the query as a consequence.

Example 2.9 (Knows/Seen)**.** Consider the Problog program:

$$P = \{1.0 :: r_1, 1.0 :: r_2, 0.8 :: r_3, 0.5 :: r_4, 0.5 :: r_5, 0.5 :: r_6, 0.5 :: r_7\}$$

And let:

r_1 = `knows(X,Y) :- hasSeen(X,Y).`

r_2 = `knows(X,Y) :- hasSeen(X,Z), knows(Z,Y), transitive().`

r_3 = `transitive().`

r_4 = `hasSeen(a,b).`

r_5 = `hasSeen(b,c).`

r_6 = `hasSeen(b,d).`

r_7 = `hasSeen(c,d).`

Briefly, the program is composed of two strict rules r_1, r_2 and five probabilistic facts $r_3, \ldots, r_7$ stating the uncertainty of the $knows(X, Y)$ being transitive through r_3 and of different people a, b, c, d having seen each other with $r_4, \ldots, r_7$. We can ask whether the person a knows d with a

[2]Meaning, the probability that some combination of strict rules of P has A as a consequence, where the probability of each combination (i.e: the subsets of strict rules) is given by multiplying the probability p_i of each rule r_i in the set and also the inverse of those rules not in the set $(1 - p_i)$.

query (ground atom) $A = knows(a, d)$, which can be derived using the following ground sets of rules, which correspond to the possible worlds that can deduce A:

$$\begin{aligned}
L_1 &= \{r_1, r_2, r_3, r_4, r_5, r_7\},\\
L_2 &= \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\},\\
L_3 &= \{r_1, r_2, r_3, r_4, r_6\},\\
L_4 &= \{r_1, r_2, r_3, r_4, r_5, r_6\},\\
L_5 &= \{r_1, r_2, r_3, r_4, r_6, r_7\}.
\end{aligned}$$

Then, the probability of success of A can be calculated as follows:

$$\begin{aligned}
Pr_s(P \models A) &= \sum_{\substack{L \subseteq L_P\\ L \models A}} Pr(L \mid P)\\
&= \sum_{i \in [1,5]} Pr(L_i \mid P)\\
&= \sum_{i \in [1,5]} 0.05\\
&= 0.25
\end{aligned}$$

Here, the probability of each subset of strict rules L_i is given by the probability of rules r_1 (1), r_2 (1) and r_3 (0.8) which are in every set, multiplied by the probability of the rest of the rules being or not being in the set. Because the probability of those other rules being or not being in the set are the same (0.5), $Pr(L_i \mid P) = 1.0{*}1.0{*}0.8{*}0.5^4 = 0.05$. And thus, as we have shown $Pr_s(P \models A) = 0.25$.

2.1.2.3 Annotated disjunctions

A simple extension that will prove useful in later chapters is that of allowing a certain kind of disjunction in Problog rules. Annotated disjunctions extend Problog programs with expressions of the form:

$$\mathtt{p}_1 \ \mathtt{::}\ \mathtt{H}_1 \ \mathtt{;}\ \ldots\ \mathtt{;}\ \mathtt{p}_m \ \mathtt{::}\ \mathtt{H}_m \ \mathtt{:-}\ \mathtt{B}_1,\ldots,\mathtt{B}_n.$$

Where each $p_i \in [0, 1]$ is a probabilistic annotation, $\sum_{i=1}^{n} p_i \leq 1$, each H_j is an atom representing different heads of the rule and $B_1, \ldots, B_n$ is the body of the rule just like in a classic Problog program. We assume that when using this kind of rule: (i) all variables in the head also appear in the body, and (ii) no head atom H_j can be unified with another H_k.

The semantics of annotated disjunctions is that whenever the body $B_1, \ldots, B_n$ of the rule is evaluated to be true, then at most one of the head atoms H_j holds as well. This means that in the context of Definition 2.7, which characterizes the probability of a possible world (set of rules without annotations) $Pr(L|P)$, $\mathrm{p}_1 :: \mathrm{H}_1 ; \ldots ; \mathrm{p}_m :: \mathrm{H}_m :\!- \mathrm{B}_1,\ldots,\mathrm{B}_n.$ denotes that the probability of the (non-disjunctive) rule $\mathrm{H}_i :\!- \mathrm{B}_1,\ldots,\mathrm{B}_n.$ being present in a possible world is p_i for each $i \in [1, m]$, the probability of more than one such rule being present in a possible world is none, and that the probability of no such rule being part of a possible world for any $i \in [1, m]$ is $1 - \sum_{i=1}^{m} p_i$, which corresponds to the probability of no atom H_j holding when the body $B_1, \ldots, B_n$ does.

Example 2.10. Consider a Problog program P composed only of the following rule:

```
0.8 :: a() ; 0.2 :: b() :- c().
```

It states that whenever the atom $c()$ holds, then there is a 0.8 probability of $a()$ holding and 0.2 of $b()$ holding and that in each possible world one or the other is present, but in no case should both $a()$ and $b()$ hold together. In other words, given the queries $Q_1 = \{a()\}$, $Q_2 = \{b()\}$ and $Q_3 = \{a(), b()\}$, then the probabilities of success of the queries are $Pr(Q_1|P) = 0.8$, $Pr(Q_2|P) = 0.2$ and $Pr(Q_3|P) = 0$.

It is also important to note that Problog programs with annotated disjunctions can be translated into equivalent programs without them, as demonstrated in [Gutmann, 2011], so they can actually be considered syntactic sugar. Here, we will not delve into the details of this transformation and refer the reader to [Gutmann, 2011] for more information about it and the proof of correctness.

2.1.2.4 Complexity

It is clear the querying problem is computationally expensive for probabilistic logic programs and that naive computation is infeasible for all but the smallest programs. This is why more optimized procedures have been developed. [De Raedt et al., 2007] use an approximation procedure based on previous work [Poole, 1993] in order to compute these probabilities using binary decision diagrams [Bryant, 1986]. For an in-depth complexity analysis of probabilistic logic programming semantics we refer the reader to [Riguzzi and Swift, 2018] and in the particular case of Problog, to [Kimmig et al., 2011, De Raedt et al., 2007, De Raedt and Kimmig, 2015].

2.1.3 Parameter learning from evidence

So far, we have seen how Problog defines the probability of success of a query given a probabilistic logic program. However, the language can also be utilized for learning probabilistic annotations for facts if the right program and evidential interpretations (i.e: truth-value assignments for ground atoms) are given, by utilizing procedures such as the one presented in [Gutmann et al., 2011]. In this section, we will overview this procedure and the representation of the parameter learning problem, which will be used later in Chapter 6.

An *evidential interpretation* of a Problog program P assigns truth values to (some of) its ground atoms. More formally, an interpretation is represented as a pair $I = (I^+, I^-)$ composed of two disjoint sets of ground atoms from P where I^+ are the atoms considered true and the ones in I^-, false. We will use evidential interpretations to represent knowledge in the form of a combination of true and false ground atoms that hold together according to an information source. They differ from the interpretations described in Section 2.1.1.2 in that, by considering both true and false sets of ground atoms I^+, I^- the truth value of the rest of the ground atoms is considered unknown, instead of false like in classic interpretations. From a machine learning perspective, evidential interpretations will describe the examples of a dataset.

Before defining the parameter learning problem, we need to define the probability of an evidential interpretation being entailed by a Problog program. Given an evidential interpretation I and a Problog program P, $Pr(I)$ denotes the probability of all the elements in I^+ being entailed by P and none of the ones in I^-:

Definition 2.11 (Probability of an evidential interpretation)**.** Given a Problog program P and an evidential interpretation $I = (I^+, I^-)$, then the probability of I according to P is:

$$Pr(I|P) = \Big(\prod_{A \in I^+} Pr(A|P)\Big) * \Big(\prod_{A \in I^-} (1 - Pr(A|P))\Big)$$

Where each ground atom $A \in I^+ \cup I^-$ can be seen as a query and $Pr(A|P)$ denotes the success probability of the query.

Now, we are ready to define the problem of learning parameters from evidential interpretations, based on [Gutmann et al., 2011]. Essentially, the parameter learning problem consists of finding the probabilistic annotations $p_1, \ldots, p_n$ that maximize the probability of the evidential interpretations holding, as follows:

Definition 2.12 (Parameter learning problem)**.** A *parameter learning problem* is characterized by:

- A **parametrized** Problog program $P(p_1, \ldots, p_n) = P_{fixed} \cup P_{param}$ composed of:
  - A fixed set of Problog rules P_{fixed}, and
  - A set of Problog rules P_{param} with n **unknown** probabilistic annotations $p_1, \ldots, p_n$.
- A multiset of evidential interpretations $Is = \{I_1, \ldots, I_k\}$.

Given the above, find the optimal parameters $\hat{p_1}, \ldots, \hat{p_n}$ such that:

$$(\hat{p_1}, \ldots, \hat{p_n}) = \underset{(p_1, \ldots, p_n) \in [0,1]^n}{\operatorname{argmax}} Pr(Is|P(p_1, \ldots, p_n))$$

$$= \underset{(p_1,\ldots,p_n)\in[0,1]^n}{\text{argmax}} \prod_{i=1,\ldots,k} Pr(I_i|P(p_1,\ldots,p_n))$$

Where $Pr(I_i|P(p_1,\ldots,p_n))$ is the probability of the evidential interpretation I_i given the Problog program $P(p_1,\ldots,p_n)$ with some chosen parameters.

That is, the evidential interpretations serve as the dataset of examples (or evidence) that the program has to resemble as close as possible. If no combination of parameters can entail any of the evidential interpretations, for example, if the rules in P are not enough to entail any of the ground atoms in any evidential interpretation I_i, any parameters will suffice. On the other hand, if this is not true, then a solution will tune the parameters accordingly.

Let us exemplify how this technique works.

Example 2.13 (Connectivity)**.** Consider the following parameter learning problem inspired by graph theory, that describes a predicate *connected*, which represents the transitive closure of the relation modeled by the predicate *edge*:

- A **parametrized** Problog program:

$$P(p_1,\ldots,p_n) = P_{fixed} \cup P_{param}$$

  Where:

$$\begin{aligned} P_{fixed} =& \{\texttt{1.0::connected(X,Y):-edge(X,Y).} \\ & \texttt{1.0::connected(X,Y):-edge(X,Z),connected(Z,Y).}\} \\ P_{param} =& \{p_1\texttt{::edge(a,b).} \\ & p_2\texttt{::edge(b,a).} \\ & p_3\texttt{::edge(b,c).} \\ & p_4\texttt{::edge(c,b).} \\ & p_5\texttt{::edge(a,c).} \\ & p_6\texttt{::edge(c,a).}\} \end{aligned}$$

- A multiset of evidential interpretations $Is = \{I_1, I_2\}$, where:

$$\begin{aligned} I_1^+ =& \{\texttt{connected(a,b),connected(b,a)}\} \\ I_1^- =& \{\texttt{connected(b,c),connected(c,b),} \\ & \texttt{connected(a,c),connected(c,a)}\} \\ I_2^+ =& \{\texttt{connected(a,b),connected(b,c)} \\ & \texttt{connected(a,c)}\} \\ I_2^- =& \{\texttt{connected(b,a),connected(c,b),} \end{aligned}$$

```
connected(c,a)}
```

Simply, we have strict rules describing the *connected* predicate but it is unknown which edges exist in the graph connecting the nodes a, b, c. The evidential interpretations in Is inform different points of view, I_1 states that only a and b are connected in both directions, and I_2 states that a is connected to b, which is connected to c.

With analysis, we can find the solution for this parameter learning problem:

$$(p_1, p_2, p_3, p_4, p_5, p_6) = (1, 0.5, 0.5, 0, 0, 0)$$

The explanation behind $p_1 = 1$ is that in both interpretations the edge between a and b must exists because the only other way in which these nodes can be connected would be if a is connected to c and then c to b using some combination of edges, but this does not hold for I_1 or I_2. Then, $p_2 = 0.5$ because only in I_1 b is connected to a and the only way this can happen is if there is an edge between them. A similar explanation can be made for $p_3 = 0.5$ and I_2. And lastly, no other edge can be present because no set of edges that includes `edge(c,b)`, `edge(a,c)`, or `edge(c,a)` satisfies I_1 or I_2.

For the purposes of this thesis, we will not delve into the details of a solver. A description of a parameter learning solver can be found in [Gutmann et al., 2011] and an implementation is publicly available online under the Problog library[3]. Regardless of the implementation, the parameter learning setting can present enormous computational running time costs if the Problog program is sufficiently complex, as parameter learning will depend on the cost of querying. However, the currently available implementation can handle large amounts of ground atoms, in the scale of the thousands, for relatively simple Problog programs, as the WebKB dataset [Craven and Slattery, 2001].

2.2 Planning

Automated planning is one of the oldest AI problems and one of its major fields. In simple terms, it deals with reasoning about sequences of actions and their impact on the surrounding world to fulfill a request. Its overarching goal is not only to develop entities that exhibit intelligent behavior but also to understand what constitutes smart planning and intelligence as a whole.

Generally speaking, planning is an explicit deliberation process that chooses and organizes actions by anticipating their outcomes, aiming at achieving some predefined objectives [Ghallab et al., 2004]. In other words, planning involves reasoning about the outcomes of actions, how they relate to one another and their impact on the state of the world, be it in real life

[3]https://github.com/ML-KULeuven/problog

or a virtual setting. The automated agent perceives a state of the world, or more simply *state*, either as previously acquired information, sensors or other external sources. This states are represented using properties, which are frequently called *fluents*, and hold a certain *value*. An automated agent can execute *actions*, which can change the value of fluents under certain conditions. These changes represent the *effects* of the action in the state of the world. Given a request, i.e: a combination of fluents that we want to hold in the world, we can compute the world states as they will be when we execute some actions and check whether this request will be achieved, by anticipating the outcomes. Literature refers to these requests as *goals*. Thus, planning revolves around finding actions that when performed in an *initial* state, will ultimately affect the state of the world, in such a way that the resulting state will satisfy the predefined goal.

As such, planning is a very general problem setting that requires looking ahead in terms of world states and depending on the complexity of these world states and the actions, it can make anticipating these world states a computationally intractable problem. Let us note that planning is not required in cases where the goal can be fulfilled by predefined actions immediately. For example, in the case of simple systems (such as an elevator), where all possible requests are known in advance, it suffices to pre-program a set of actions for each of these requests. However, in cases where one faces new situations with many different possible fluent combinations for goals, changing world states, and in which goals take multiple inter-dependent actions to be fulfilled, planning is required. This type of problem is even more common in multi-agent settings, where coordination between the actions of multiple agents is essential to figure out what each agent has to do and in which order. It can also be the case where planning is performed in an environment that poses a high risk or cost to choosing the right or wrong actions, in this case, not any sequence of actions might suffice, but one might want to find an optimal plan according to some metrics.

2.2.1 History and context

With regard to artificial intelligence, planning deals with the computational study of the deliberation process just described. It was first studied in the context of Shakey [Nilsson, 1984], a project from the Stanford Research Institute (SRI) in the 1960s that looked into different ways of providing robots the ability to reason about their environment, analyze requests and break them down into sequences of actions to perform. The Shakey robot was one of the first-ever robots that implemented what we now know as an AI planner to organize and select a sequence of actions to fulfill requests. It gathered information about its environment using an antenna for a radio link, sonar range finders, a television camera, onboard processors, and collision detection sensors. The robot made sense of the world through a model composed of some rooms connected by corridors, with doors and light switches that the robot could interact with. Typical requests consisted of going to certain areas of some room, or moving objects around, which it was able to do with the help of a set of wheels and a motor. The project itself culminated with the development of the STRIPS planning language and many significant results for AI as a whole, such as the A* search algorithm [Hart et al., 1968].

Since its inception, the field has grown considerably into various branches. One of the most general divisions is that of domain-specific vs domain-independent planning. Domain-specific planning is dedicated to solving problems using highly specific representations and algorithms for their problems, that cannot be always applied to other settings. Some examples of this can be found in path and motion planning, i.e: the process of constructing a path and determining the actions to traverse from a starting point to an endpoint given a map of the agent's surroundings. On the other hand, domain-independent planning uses generic representations and techniques to solve generic planning problems. The advantage of this approach is that planning algorithms can be immediately adapted across domains every time a new problem is encountered. However, this requires planning without any domain background knowledge, which can be inefficient, but it also leads to a better understanding of the planning problem itself. Although only domain-independent techniques can always be used in domain-specific problems, the two methods are complementary. Domain-specific planning is useful when efficiency is critical, whereas domain-independent planning is useful when planning for many different contexts without previous knowledge.

Moreover, the assumptions the agent takes into account when defining a model of its environment and its own capabilities largely affect the planning algorithms that can be used for that specific setting. Many of them involve *state properties*, also known as *fluents*, i.e: the set of values that define a state. Some of these assumptions are the following:

- **State observability**: can all properties of the state be perceived by the agent (full observability) or only some of them (partial observability)?
- **Fluent value domains**: are the values that fluents can take discrete or continuous?
- **Action determinism**: are the effects of actions certain and known in advance (deterministic), or not (non-deterministic)? And if not, are the effects governed by some probability distribution?
- **Action duration**: are the effects of actions immediate, or are they dependent on some time unit?
- **Action concurrency**: can many actions be taken at the same time?
- **Number of agents**: is it only one agent able to perform an action (single agent) or many of them (multi-agent)?
- **Exogenous actions**: can certain property values change without the agent performing any action?

- **Initial/goal state(s)**: is the initial/goal state known in advance fully, or partially? Is the initial/goal state unique or a set of states?
- **Action cost/utility**: do actions have an associated cost or utility? If so, which values can these costs/utilities have, and by which unit are they measured?
- **Re-planning**: is the plan executed as-is, or are they any circumstances (e.g.: execution failures) in which the agent can compute a whole new plan in light of new knowledge?

All of these assumptions affect not only the planning process of the agent, but also the information it needs to process, the capabilities necessary to acquire it, and ultimately, the complexity of computing a plan. For instance, if the setting is one of partial observability, the agent may never know whether the goal state has been reached or not. Likewise, if the actions are not deterministic, the agent will not know which effects were effective until it performs its actions, and so it may have to foresee all the possible branching effects of its actions unless, of course, it is possible to re-plan. In the case actions are given duration, the agent will be forced to keep track of their effects and the moment in which they become effective. And if many agents can perform actions in the same task, coordination and communication between them will likely be necessary to achieve the goal. If we consider action costs, not every sequence of actions leading to the goal will be equivalent, and so we might be interested in finding an optimal plan, i.e: one with minimal cost.

In short, these assumptions characterize distinct sub-fields of planning, as they each require different algorithms and capabilities from the agent.

For the remainder of this thesis, we will focus on what is known as classical planning, which will be the subject of the following section.

2.2.2 Classical planning

Taking perhaps the simplest assumptions from those mentioned before leads to what is known as classical planning. This type of planning was the one considered first by the STRIPS [Fikes and Nilsson, 1971] project. Although the problem definition might seem simple, it developed many powerful ideas and formed the basis for developing many non-classical planning techniques. This setting considers problems in which:

- All states are **fully observable**, i.e: all of their properties are known at each point in time.
- The values that fluents can take are **discrete**.
- The effects of actions are **deterministic**, i.e: certain and known in advance.
- The effects of actions are **immediate**.
- Only a **single action** can be taken at the same time.

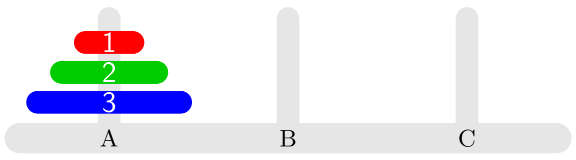


Figure 2.1: Towers of Hanoi example.

- There is a **single agent** able to perform actions.
- There are **no exogenous actions**, i.e: property values can only change as a result of an action performed by the agent.
- The initial state is **unique**. Both the initial and goal states are **fully known in advance**.
- Actions have **no associated cost**.
- At least in the most basic form, there is **no re-planning**, i.e: all actions in the plan are supposed to be executed after the planning process.

To illustrate, one can think of the following famous puzzle.

Example 2.14 (Towers of Hanoi)**.** There are three pegs (A, B and C) and three disks (1, 2 and 3) of increasing width sitting on top of peg A. The problem consists of moving all the disks to C. Only the top disk from a peg may be moved to another peg and a smaller disk can never be placed underneath a bigger one.

Considering the assumptions above, this puzzle can be seen as a classical planning problem. Informally, a state of the world is represented by the position of the disks on the pegs and we can observe them at all times. The position of the disks (which will be the fluents of the problem), namely, in which peg they are located and on top of which other disks, are all discrete properties. Moreover, moving disks from one peg to another are deterministic, immediate and non-concurrent actions, performed by a single agent (the player) and have no associated cost.

Nevertheless, to show that the puzzle is indeed a classical planning problem, we will need a formal description. AI planning sub-fields use specific languages to model planning problems. They serve both as a basis for algorithms and as a characterization of what problem domains can be captured by the sub-field. As with all planning formalisms, the representation at hand serves as an imperfect approximation of problems one would encounter in the real world. However, by incorporating restrictive assumptions, one can gain both computational capabilities and representational simplicity, which will allow us to focus on reasoning layers on top of the actual planning domain in later chapters.

For the rest of this thesis, we will use a language based on the Planning Domain Definition Language (PDDL) along the lines of [Russell, 2010].

Although PDDL has been described before, most literature resorts to semi-formal definitions when it comes down to the logic-based representation of fluents and actions. Here, we have defined all constructs using the definitions and concepts of Section 2.1, for improved consistency and use in later chapters.

As mentioned before, states are defined using fluents. A fluent denotes a property of a state that can be affected by an agent executing actions.

We represent fluents with ground atoms without function symbols other than constants, as described in Section 2.1, coming from a predefined language $\mathcal{L}$. For the rest of this thesis, we will suppose that the predicate symbols $\mathcal{P}$ and constants $\mathcal{C}$ of this language are finite and restricted to the symbols mentioned in the planning problem specification, which we will define in what follows.

Definition 2.15 (Fluent)**.** A *fluent* is a ground atom $p_f(c_1, \ldots, c_k)$, where p_f is a predicate and $c_1, \ldots, c_k$ are constants from $\mathcal{L}$. F is the predefined set of possible fluents, i.e: $f \in F$.

A world state perceived by the agent, or simply state, is characterized by a set of fluents in F that hold at the current point in time, which can also be interpreted as a conjunction. The close-world assumption is taken into account, i.e: any fluent that is not part of a state is considered to be false.

Definition 2.16 (State)**.** Given a set of fluents F, a *state* $s \subseteq F$ is a subset of those fluents. We denote the set of all possible states S, which corresponds to the powerset of fluents 2^F.

Having defined the elements that represent states of the world, we can now give a proper definition for operators and actions, which trigger changes between said states. An operator is a lifted representation of an action, meaning that it uses first-order logic variables to represent a set of actions, which are its ground instances.

Definition 2.17 (Operator)**.** An *operator*, sometimes called action schema, is a construct of the form:

$$o = \langle Name(o), Pre(o), Eff(o) \rangle$$

Consisting of:

- An atom $Name(o) = p(X_1, \ldots, X_n)$ with predicate p from $\mathcal{L}$ denoting the *name* of the operator and $X_1, \ldots, X_n$ a list of all the variables that can be used in the operator,

- A set of literals $Pre(o)$ with variables contained in $\{X_1, \ldots, X_n\}$, called the *preconditions*, which define the conditions in which it is possible to execute an operator, and

- A set of *effects* $Eff(o) = \{Eff_1(o), \dots, Eff_{n_o}(o)\}$ where $n_o \in \mathbb{N}$ and for every $i \in [1, n_o]$:

  $$Eff_i(o) = \forall(Y_1, \dots, Y_{m_i})\ Cond_i(o) \Rightarrow Post_i(o)$$

  where:

  - $Y_1, \dots, Y_{m_i}$ are distinct variables, disjoint from $X_1, \dots, X_n$,
  - $Cond_i(o)$ and $Post_i(o)$ are sets of fluent literals with variables contained in $\{X_1, \dots, X_n\}$ or $\{Y_1, \dots, Y_{m_i}\}$, denoting that whenever the *conditions* $Cond_i(o)\theta$ hold in a state for any grounding θ of $\{Y_1, \dots, Y_{m_i}\}$, then the *postcondition* $Post_i(o)\theta$ represents the updates that will be applied as a result of executing the operator.

  In the special case that $Eff_i(o)$ contains no variables ($m_i = 0$), we can represent it without the quantifier, i.e: $Eff_i(o) = Post_i(o)$.

Both the literals in the preconditions and effects are restricted to be composed of the same predicate symbols used in F, and their arguments to be constants or variables in $\{X_1, \dots, X_n\}$ (and $\{Y_1, \dots, Y_{m_i}\}$ in the case of effects). The list of variables $X_1, \dots, X_n$ is meant to be interpreted as universally quantified, meaning that the content of o holds for any substitution of the variables to constants. We denote O the predefined set of all operators.

An expression of the type $\forall(Y_1, \dots, Y_{m_i})\ Cond_i(o) \Rightarrow Post_i(o)$ is what is typically called a conditional effect in the context of PDDL [Fox and Long, 2003] and they are useful for designing more compact representations of planning problems.

Then, when a substitution θ grounds all the variables of an operator o, we call the result of the grounding, an action.

Definition 2.18 (Action)**.** An *action* is a ground instance of an operator. Given an operator $o = \langle Name(o), Pre(o), Eff(o)\rangle$ with variables $\{X_1, \dots, X_n\}$ and a grounding substitution θ of the variables, we call

$$a = \langle Name(o)\theta, Pre(o)\theta, Eff(o)\theta\rangle$$

an action, where θ substitutes the atom in the name and literals in the preconditions o, and

$$Eff(o)\theta = \bigcup_{i=1,\dots,n_o} Eff_i(o)\theta$$

$$Eff(o)_i\theta = \bigcup_{\theta' \text{ grounding of } \{Y_1,\dots,Y_{m_i}\}} (Cond_i(o)\theta)\theta' \Rightarrow (Post_i(o)\theta)\theta'$$

Notice that although θ grounds the parameters of the action, without grounding $Y_1, \dots, Y_{m_i}$ as well with θ', the resulting expression would not be composed of ground fluent literals. By grounding all the variables in the universal quantifier preceding $Cond_i(o)$, we ensure that $Eff(o)_i\theta$ and by extension, $Eff(o)\theta$ will be composed of ground $(Cond_i(o)\theta)\theta' \Rightarrow (Post_i(o)\theta)\theta'$ expressions without variables or quantifiers.

We denote $A(o)$ the set of all actions resulting from grounding o with constants from $\mathcal{C}$ as defined before.

We will also use $Name(a), Pre(a), Eff(a), Eff_i(a), Cond_i(a), Post_i(a)$ for actions in the same way as operators.

Because all the variables in the operator are parameterized in the name and the universal quantifiers of its effects, grounding these variables also does so for the preconditions and effects. Therefore, both the literals in the preconditions and effects of an action are composed of fluents or their negation.

The semantics of actions in a state is defined in terms of fluents they add or remove from a state. An action is applicable in a state whenever all the positive fluent atoms in its preconditions are included in the state and non of the negative ones are. Moreover, whenever an applicable action is executed in a state, another state is produced in which all the positive literals in the effects are added and the negatives are removed. All the other fluents not mentioned in the effects are preserved. This is important in terms of planning as it explains how the model solves the frame problem [Shanahan, 2016], i.e: ensuring fluents in a state do not change arbitrarily without an action being executed.

Definition 2.19 (Applicability and successor state)**.** An action a is *applicable* in state s, denoted $s \models a$ if and only if $Pre(a) \cap F \subseteq s$ and $\{f : \neg f \in (Pre(a) - F)\} \cap s = \emptyset$.

If $s \models a$, the *successor state* of executing the action is:

$$\begin{aligned} Succ(a,s) = \big(s - \{f \in F : \exists Eff_i(a) \text{ s.t. } s \models Cond_i(a) \text{ and } \neg f \in Post_i(a)\}\big) \\ \cup \{f \in F : \exists Eff_i(a) \text{ s.t. } s \models Cond_i(a) \text{ and } f \in Post_i(a)\} \end{aligned}$$

Furthermore, this can be extended to sequences of actions.

A sequence of actions $\pi = [a_0, a_1, \dots, a_n]$ with $n \geq 0$ and $a_0, a_1, \dots, a_n \in A$ is *applicable* in state s if all of their actions are, i.e: a_0 is applicable in s and for every $i \in [1, n]$ it holds that a_i is applicable in $Succ(a_{i-1}, \dots Succ(a_0, s))$.

Given that all actions are applicable in their respective states, the state resulting from executing a list of actions is defined as:

$$Succ(\pi, s) = Succ(a_n, Succ(\dots, Succ(a_1, Succ(a_0, s))))$$

We have now defined all that is necessary to formalize classical planning problems. This is typically done in two steps: by defining planning domains, which capture first-order lifted information (i.e: with variables) about a problem and actual planning problems, which are their ground instances along with an initial state and ending (also called goal) states.

Definition 2.20 (Classical planning domain)**.** A *classical planning domain* is a triple $\mathcal{D} = \langle \mathcal{L}, F, O \rangle$ composed of a first-order logical language $\mathcal{L}$, fluents F over $\mathcal{L}$ and operators O over $\mathcal{L}$ and F.

Then, as we mentioned before, a planning problem is a ground instance of a planning domain along with an initial and goal state. An initial state represents the current state of the world before the intervention of an agent executing actions, while a goal state is a set of literals that denote the ending conditions after the execution of a sequence of actions under which one can say that the problem was successfully solved.

Definition 2.21 (Classical planning problem)**.** A *classical planning problem* is a tuple $T = \langle \mathcal{D}, s_0, g \rangle$ that describes all the relevant information that characterizes the states of the domain, the actions, the initial state and the final conditions a plan has to reach. More precisely:

- $\mathcal{D} = \langle \mathcal{L}, F, O \rangle$ is a planning domain,
- s_0 is a set of fluents called the *initial state*, and
- g is a set of literals called the *goal*.

We denote $A(T) = \bigcup_{o \in O} A(o)$, or simply A when the context is free from multiple planning problems, the set of all actions in T. In other words A contains all ground instances of O with substitutions to constants from $\mathcal{L}$.

Let us illustrate these concepts with the following example.

Example 2.22 (Towers of Hanoi continued)**.** Following up on the example, we can represent the formalized planning task T as follows:

- $$\begin{aligned} F =& \{clear(X) : X \in \{d_1, d_2, d_3, p_A, p_B, p_C\}\} \cup \\ & \{on(X, Y) : X \in \{d_1, d_2, d_3\} \wedge Y \in \{d_1, d_2, d_3, p_A, p_B, p_C\}\} \cup \\ & \{canStack(X, Y) : X \in \{d_1, d_2, d_3\} \wedge Y \in \{d_1, d_2, d_3, p_A, p_B, p_C\}\} \end{aligned}$$
- $$\begin{aligned} s_0 =& \{clear(d_1), clear(p_B), clear(p_C)\} \cup \\ & \{on(d_1, d_2), on(d_2, d_3), on(d_3, p_A)\} \cup \\ & \{canStack(d_1, p_A), canStack(d_2, p_A), canStack(d_3, p_A) \\ & canStack(d_1, p_B), canStack(d_2, p_B), canStack(d_3, p_B) \\ & canStack(d_1, p_C), canStack(d_2, p_C), canStack(d_3, p_C) \\ & canStack(d_1, d_2), canStack(d_2, d_3), canStack(d_1, d_3))\} \end{aligned}$$.
- $g = \{on(d_1, d_2), on(d_2, d_3), on(d_3, p_C)\}$.
- O is the set of operators of the form:

$$a = \langle move(D, X, Y), Pre(a), Eff(a) \rangle$$

where $Pre(a) = \{canStack(D, Y), on(D, X), clear(D), clear(Y)\}$ and $Eff(a) = \{clear(X), on(D, Y), \neg clear(Y)\}$.

The constants available in the problem d_1, d_2, d_3 refer to the disks, while p_A, p_B, p_C denote each of the three pegs.

Intuitively, $clear(X)$ denotes a disk or peg that does not have a disk on top, $on(X, Y)$ refers to whether the disk or peg X is placed on top of Y and $canStack(X, Y)$ means that disk X can be potentially stacked over the disk or peg Y. This can hold in two cases: a disk can be stacked on a bigger disk (recall d_1 is smaller than d_2 and d_2 is smaller than d_3) and any disk can be stacked on a peg.

Then, an action $move(D, X, Y)$ amounts to moving a disk D that is placed on top of either a peg or another disk, which we will refer to as X, to another peg or disk Y, given that both are clear (they do not have a disk on top) and that X can be placed over Y.

Notice that the three effects of $move(D, X, Y)$ have no condition or extra variables apart from D, X, Y, and thus we can ignore the universal quantifier in each effect.

As explained earlier, performing actions triggers transitions between states. A planning problem is solved by finding a list of actions that arrives at a goal state by successively applying all its actions from the initial state. This list of actions is called a plan.

Definition 2.23 (Plan)**.** A *plan* for a planning problem T is a list of actions $\pi = [a_0, a_1, \ldots, a_n]$ with $n \geq 0$ and $a_0, a_1, \ldots, a_n \in A$ that is applicable in s_0 and satisfies the goal conditions, i.e: $\pi \models g$ if and only if it holds that $Succ(\pi, s_0) \subseteq \{f \in F : f \in g\}$ and $Succ(\pi, s_0) \cap \{f \in F : \neg f \in g\} = \emptyset$.

Following this definition:

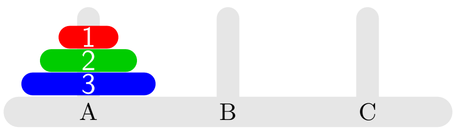
Example 2.24 (Towers of Hanoi continued)**.** A plan for our running example is given by:

$$\begin{aligned}\pi_{hanoi} =&[move(d_1, p_A, p_C), move(d_2, p_A, p_B), move(d_1, p_C, p_B),\\ &move(d_3, p_A, p_C), move(d_1, p_B, p_A), move(d_2, p_B, p_C), move(d_1, p_A, p_C)]\end{aligned}$$

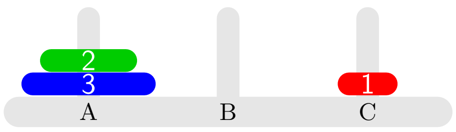
And the final state is:

$$\begin{aligned}s_7 = \{&clear(p_A), clear(p_B), clear(d_1)\\ &on(d_1, d_2), on(d_2, d_3), on(d_3, p_C)\}\end{aligned}$$

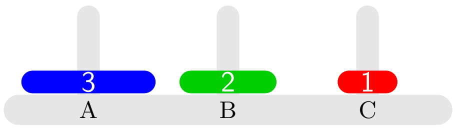
We can see the successive states traversed by the plan in Figure 2.2. It is simple to see $\pi_{hanoi} \models g$ but that the plan is not unique, i.e: there are many ways to reach the final state, for example, by moving disk



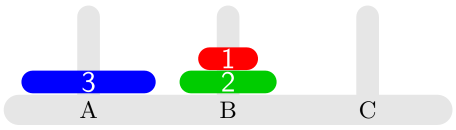
(a) Initial state s_0



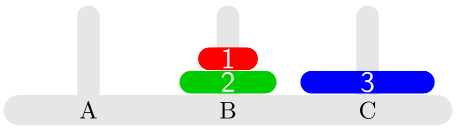
(b) $s_1 = Succ(move(d_1, p_A, p_C), s_0)$



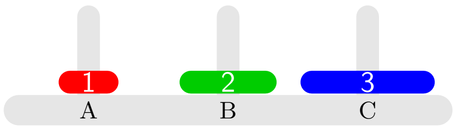
(c) $s_2 = Succ(move(d_2, p_A, p_B), s_1)$



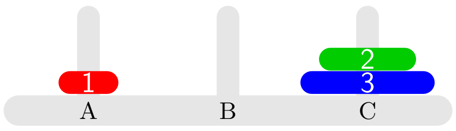
(d) $s_3 = Succ(move(d_1, p_C, p_B), s_2)$



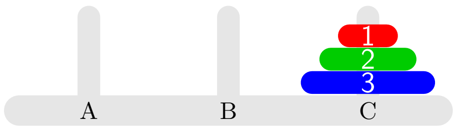
(e) $s_4 = Succ(move(d_3, p_A, p_C), s_3)$



(f) $s_5 = Succ(move(d_1, p_B, p_A), s_4)$



(g) $s_6 = Succ(move(d_2, p_B, p_C), s_5)$



(h) $s_7 = Succ(move(d_1, p_A, p_C), s_6)$

Figure 2.2: Towers of hanoi state transitions for plan π_{hanoi}.

d_1 back and forth between pegs and then performing the rest of the actions in π_{hanoi}, amongst other possibilities.

In most problems, many different plans can be generated to reach the goal. However, not all plans might be as desirable. For instance, in the example before, many useless intermediate actions could be performed (moving a disk back and forth). Furthermore, performing actions can be time-consuming or carry costs. This is why in the following section, we will show how the previous planning problems can be extended with numerical utilities.

2.2.3 Planning with utilities

So far, we have defined classical planning problems that consist of finding sequences of actions that transition from the initial state to another that satisfies the goal conditions. However, a simple, yet powerful extension that can be used to capture a whole new set of problems is that of utilities, in particular, action costs and soft goals.

Representing costs and utilities allows to model problems that take into account resources of many different kinds (e.g: power consumption, financial cost, etc.) associated with performing actions.

Definition 2.25 (Action cost)**.** An *action cost* function $c : A \mapsto \mathbb{R}_0^+$ maps the actions of a planning problem T into non-negative reals.

Normally in planning, we will be interested in action cost functions that are decidable and can be evaluated in, at the most, polynomial time with respect to the size of its input, namely the atoms in the action and state taken as input, in order to not produce an unnecessary overhead in the computation of plans.

Soft goals allow defining preferences between goal states by assigning numerical values to them. That is, they allow to compare plans in terms of their ending state but do not define which sequences of actions constitute a valid plan, as goals normally do.

Definition 2.26 (Soft goal)**.** A *soft goal* function $u : S \mapsto \mathbb{R}_0^+$ maps the states $s \in S$ of a planning problem T into non-negative reals.

Depending on the planning model that implements this theoretical framework, soft goals may be represented in many different ways, for example, by using logic formulas over the fluents and assigning utilities to the goal states that satisfy them [Gerevini et al., 2009]. However, for the remainder of this thesis, we will consider only soft goal utilities for single states.

Definition 2.27 (Utility planning problem)**.** A *utility planning problem* is a tuple $T = \langle \mathcal{D}, s_0, g, c, u \rangle$, where $\langle \mathcal{D}, s_0, g \rangle$ is a classical planning problem, c is an action cost function over A the actions in $\mathcal{D}$ and u is a soft goal cost function over F the fluents in $\mathcal{D}$.

Given a utility planning problem $T = \langle \mathcal{D}, s_0, g, c, u \rangle$, the plans for T are the same as those in the classical planning problem $\langle F, s_0, g, O \rangle$, as the semantics of their actions remain the same. The only difference comes from comparing the utility of different plans. The utility of a plan π corresponds to the costs of the actions in it and the final state.

Definition 2.28 (Plan utility)**.** Given a plan π for $T = \langle \mathcal{D}, s_0, g, c, u \rangle$, its utility is defined as:

$$u(\pi) = u(Succ(\pi, s_0)) - \sum_{a \in \pi} c(a)$$

Then, this utility establishes the notion of optimality through a comparison between plans.

Definition 2.29 (Optimal plan)**.** An *optimal* plan π for $T = \langle \mathcal{D}, s_0, g, c, u \rangle$

is one for which no other plan π' has a higher utility, i.e:

$$\forall \pi' \text{ plan for } T, \text{ it holds that } u(\pi) \geq u(\pi').$$

Let us illustrate these definitions with an example. The previous example of the 'Towers of Hanoi' does not demand any apparent utilities. Instead, we will be using a typical domain inspired by planning literature called 'Logistics'.

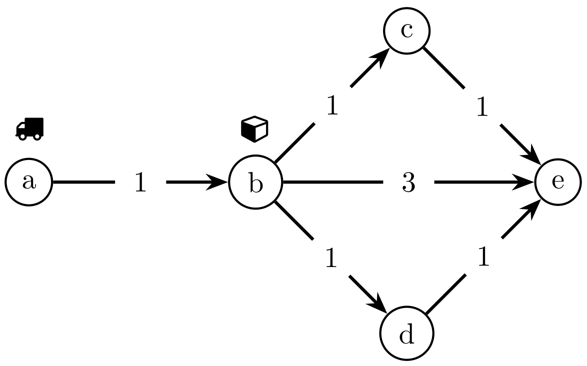


Figure 2.3: Logistics planning task.

Example 2.30 (Logistics)**.** A truck is tasked to carry a package from location a to location e. There are five different locations a, b, c, d and e which are connected as depicted in Figure 2.3, as well as the respective costs of traversing from one location to another. To carry a package from one location to another, the package can be *loaded* into and *unloaded* from the truck. At the start, the truck is unloaded and located in a, while the package is in location b. Lastly, it is considered preferable but not necessary if the package is not loaded in the truck in the final state.

We can represent this problem with $T = \langle \mathcal{D} = (\mathcal{L}, F, O), s_0, g, c, u \rangle$ as follows:

- $F = \{at(X, Y) : X \in \{truck, package\} \wedge Y \in \{a, b, c, d, e\}\} \cup$
  $\{isLoaded(truck)\} \cup$
  $\{connected(X, Y) : X, Y \in \{a, b, c, d, e\}\}$

- $O = \{o_1 = \langle load(X),$
  $\{at(truck, X), at(package, X), \neg isLoaded(truck)\},$
  $\{isLoaded(truck)\}\rangle,$
  $o_2 = \langle unload(X),$
  $\{at(truck, X), isLoaded(truck)\},$
  $\{\neg isLoaded(truck)\}\rangle,$
  $o_3 = \langle moveLoaded(X, Y),$
  $\{at(truck, X), connected(X, Y), isLoaded(truck)\},$
  $\{\neg at(truck, X), at(truck, Y), at(package, Y)\}\rangle,$
  $o_4 = \langle moveUnloaded(X, Y),$
  $\{at(truck, X), connected(X, Y), \neg isLoaded(truck)\},$
  $\{\neg at(truck, X), at(truck, Y)\}\rangle\}$
- $s_0 = \{at(truck, a), at(package, b)\} \cup$
  $\{connected(X, Y) : (X, Y) \in \{(a, b), (b, c), (b, d), (b, e), (c, e), (d, e)\}\}$
- $g = \{at(package, e)\}$.
- $c(load(X)) = 0$,
  $c(unload(X)) = 0$,
  $c(moveLoaded(X, Y)) = 1$ if and only if $(X, Y) \neq (b, e)$,
  $c(moveLoaded(X, Y)) = 3$ if and only if $(X, Y) = (b, e)$,
  $c(moveUnloaded(X, Y)) = 1$ if and only if $(X, Y) \neq (b, e)$, and
  $c(moveUnloaded(X, Y)) = 3$ if and only if $(X, Y) = (b, e)$.
- $u(s) = 10$ if and only if $isLoaded(truck) \notin s$, and
  $u(s) = 0$ if and only if $isLoaded(truck) \in s$.

Considering the following three plans:

- $\pi_1 = [moveUnloaded(a, b), load(b), moveLoaded(b, e)]$,
- $\pi_1 = [moveUnloaded(a, b), load(b), moveLoaded(b, e), unload(e)]$, and
- $\pi_3 = [moveUnloaded(a, b), load(b), moveLoaded(b, c),$
  $moveLoaded(c, e), unload(e)]$

We can calculate the utilities with the formulas given above. Then, we have that $u(\pi_1) = -4$, $u(\pi_2) = 6$ and $u(\pi_3) = 7$. Thus, out of the three plans, π_3 is considered preferred.

Although utilities like the ones described above are a well-established model to represent preferences in deterministic planning problems, other mechanisms can be used, for instance, to represent qualitative preferences on trajectory properties, represented using logic formulas and evaluated with respect to sub-sequences of actions in plans. Extensive work on automated planners designed for this model can be found in the procedures of the fifth international planning competition (IPC5) [Gerevini et al., 2009]. Also, existing research has developed planners for classical planning problems where action costs can be state-dependent [Ivankovic et al., 2014, Geißer et al., 2015].

It is also worth mentioning that the action costs we consider can be compiled away into soft goals by transforming the planning problem through the procedure described in [Keyder and Geffner, 2009]. On the other hand, action costs provide a useful technique to model many problems and can provide a more compact representation in many cases.

2.2.4 Computation

It is well known that even classical planning problems present hard complexity issues when searching for plans. Determining if a plan exists or finding it is PSPACE-hard [Bäckström and Nebel, 1995], as well as finding an optimal plan if costs and utilities are considered [Helmert, 2003], although much harder in practice.

This is why numerous techniques have been developed in order to allow planners to find any plan or optimal plans with reasonable time and space constraints for a wide variety of problems.

Many of the techniques for classical and utility planning rely on grounding the problem, i.e: computing all the possible states and actions from the operators. By doing this, one can construct a graph where the nodes are the possible states and edges denote actions. We call this graph the state transition system.

Definition 2.31 (State transition system)**.** The embedded *state transition system* of the planning problem $T = \langle \mathcal{D}, s_0, g, u, c \rangle$ is a weighted labeled directed graph $STS(T) = \langle S, E \rangle$, where:

- The set of vertices S corresponds to the set of all possible states, i.e: all the possible combinations of fluents, in T.
- There is a labeled weighted edge $e = \langle s, c(a), name(a), s' \rangle \in E$ if and only if there exists an action $a \in A(T)$ such that a is applicable in s and $s' = Succ(a, s)$. Here, s represents the starting node and s' its successor, while $name(a)$ is a ground name predicate of the underlying action and $c(a)$ its cost.

In the case of classical planning problems, it suffices to consider the same graph without weights, or with all weights in the edges set to zero.

To calculate the utility of a plan, it suffices to sum the weights of the edges traversed (which correspond to the actions of the plan) and subtract this number from the utility of the final state.

Let us illustrate an example of such a graph in what follows:

Example 2.32 (Logistics continued)**.** Given the planning problem T described before, $STS(T)$ is as described in Figure 2.4 where we use the shorthand notation $s : \{a,b,c,d\} \times \{a,b,c,d\} \times \{0,1\} \mapsto S$ to represent the nodes of the graph (which correspond to the states of T) in a compact manner:

$$\begin{aligned} s(X,Y,Z) = &\{at(truck,X), at(package,Y)\} \cup \\ &\{isLoaded(truck) \iff Z = 1\} \cup \\ &\{connected(X,Y) : (X,Y) \in \{(a,b),(b,c),(b,d),(b,e),(c,e),(d,e)\}\} \end{aligned}$$

For instance, $s(a,b,0)$ would refer to the initial state s_0, in which the package is in location b, while the truck is in a and is not loaded (represented by $Z = 0$).

Note that we omit the states of the planning task that are unreachable from the initial state and we display the cost and the name of the underlying action of each edge. Furthermore, we have marked the initial state $s(a,b,0)$ and the two reachable goal states $s(e,e,0)$ and $s(e,e,1)$ by using thicker boxes.

By using this representation, plans can be retrieved by computing paths from the initial to the goals states. Therefore, planning is reduced to a search problem, aside from the generation of the state transition graph.

However, most times searching the entire space in such a way is infeasible, as the amount of states is exponential in the size of the fluent set. This is why, many algorithms take into account heuristics [Geffner and Haslum, 2000]. A *heuristic function* $h : S \mapsto \mathbb{R}$ maps states to a numeric value that estimates the cost of getting from the initial state to the other state in the argument of the function. We say that a heuristic function is *admissible* if and only if it does not overestimate the minimum cost of any state, i.e: for every state $s \in S$ it holds that $0 \leq h(s) \leq h^*(s) = min\{cost(\pi) : Succ(\pi, s_0) \models g\}$.

Various heuristics have been studied for planning. Most of them are based on what is known as *relaxations* [Hoffmann, 2003], which consists of constructing simpler versions of the problem, for example, by not considering negative literals in action effects, known as delete-relaxations, or eliminating all the fluents using some predicate symbol, which is known as abstraction [Sacerdoti, 1974]. Another well-known strategy is that of landmark computation [Karpas and Domshlak, 2009], which amounts to finding intermediate (artificial) goals that a plan must reach in order to later find the final states that satisfy the real goal.

Some of the families of algorithms that have been developed for computing plans in classical and utility planning problems efficiently can be classified as forward-state space search, backward state-space search and plan-space search.

A *forward state-space search* algorithm consists of searching forward from the initial state to the goal states. It works by maintaining a list of candidate nodes, starting with only the initial state, along with other useful information (like the heuristic value) and the cost required to reach each of the nodes if utilities are considered. Then at each step, it proceeds by taking the head of the list, generating their adjacent nodes, pruning those

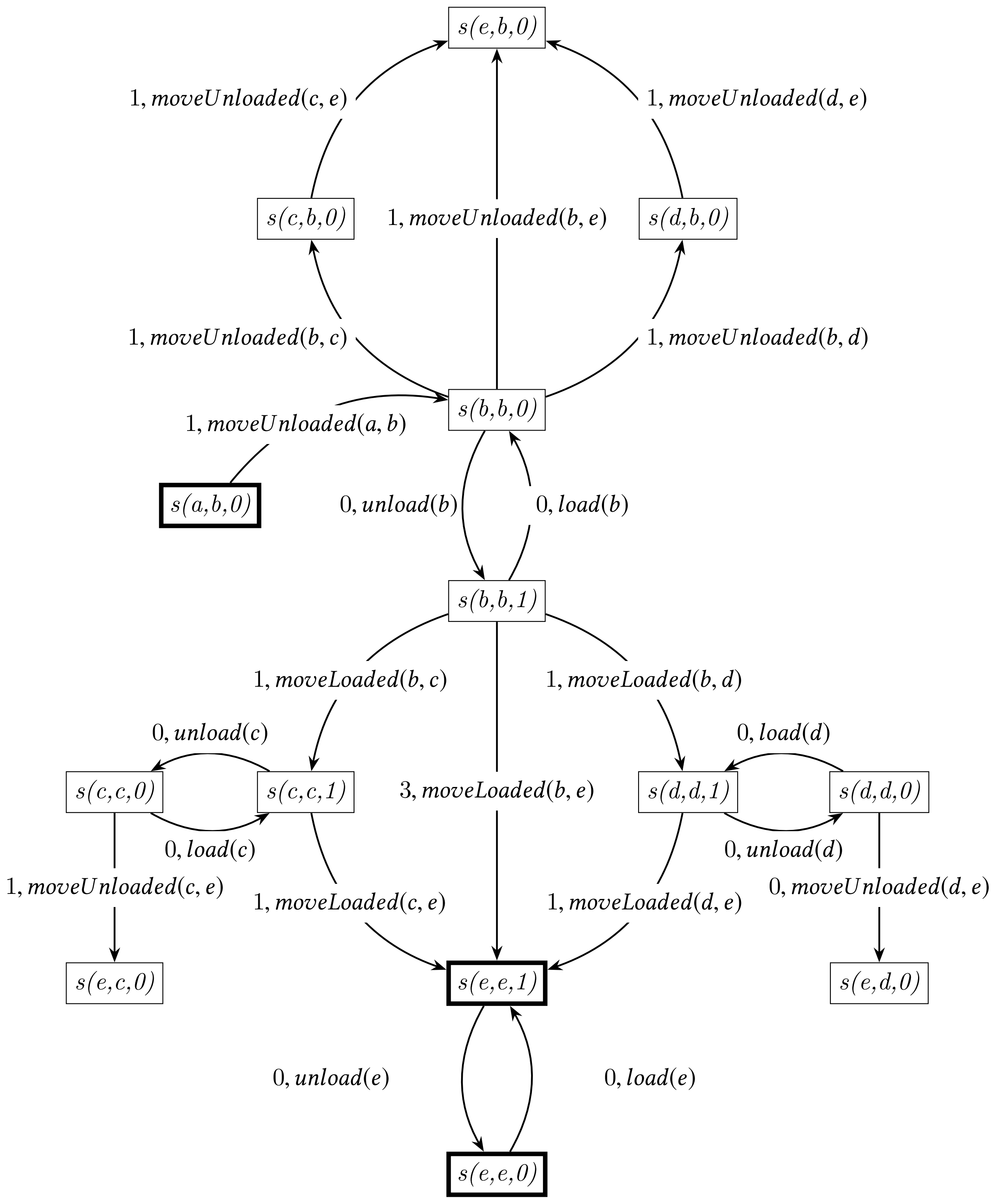


Figure 2.4: State transition system for the 'Logistics' example.

that are unpromising and adding the rest of them to the list. Finally, the algorithm stops when the required plan is computed, either by reaching any goal or reaching one with minimum cost in the case of utility planning.

The way in which nodes are selected from the list, how their children are computed and which of them are pruned change from one algorithm to another. Some of the strategies known in the literature include [Ghallab et al., 2004]: breadth-first search, depth-first search, hill climbing (greedy) search, uniform cost search, A* search, depth-first branch and bound search and iterative deepening.

As opposed to the last approach, *backward state-space search* starts from the goals states and traverses back to the initial state to find a plan. In this case, the computed plan is given by the inverse of the traversal. Many of the heuristics that work for forward-search also apply to backward search, with the caveat that the heuristic cost they calculate is the one necessary to reach the current state from the initial state, instead of being the one needed to reach a goal state from the current.

Plan-space search, in contrast, works by reformulating planning problems as constraint satisfaction problems and then using constraint satisfaction techniques to generate the plans.

For more information about the algorithms, we refer the reader to [Ghallab et al., 2004].

2.2.5 Implementations

Extensive research on automated planners designed for the planning problems modeled in this chapter can be found in the procedures of the various international planning competitions (IPC).

Naturally, the state-of-the-art implementations mentioned in this section present very complex and highly optimized algorithms that combine and optimize many of the ideas we have described so far in the previous section, along with other ideas and heuristics. Because describing all of these concepts would be largely out of the scope of this thesis, we will only mention some of these systems and a reference to their author's work.

The IPC serves as a reference for comparing algorithm implementations and techniques for PDDL problems. Various tracks of these competitions benchmark state-of-the-art planners for different types of planning problems. In particular, the *Satisficing* track compares planners for classical problems, while the *Optimal* track also takes into account action costs. In the 2008 edition, a *Net benefit* track was added, which allowed both action costs and soft goals. Later, this and other kinds of preferences were added to the Optimal track. Moreover, in the 2011 edition, the Optimal and Net benefit tracks were canceled because only one planner was submitted. Some of the winning planners for these two tracks are listed in Table 2.1.

Year	Reference	Track	Selected planners (winners, or novel)
2004	[Hoffmann and Edelkamp, 2005]	Optimal	SATPlan [Kautz et al., 2006]
		Satisficing	SGPlan [Chen et al., 2004]
			Fast Downward [Helmert, 2006]
2006	[Gerevini et al., 2009]	Optimal	SATPlan
			MIPS [Edelkamp and Helmert, 2001]
		Satisficing	SGPlan
2008	[Helmert et al., 2008]	Net benefit	Gamer [Edelkamp and Kissmann, 2009]
		Optimal	Gamer
		Satisficing	LAMA [Richter et al., 2011]
2011	[López et al., 2015]	Optimal	FDSS [Helmert et al., 2011]
		Satisficing	LAMA
2014	[Vallati et al., 2018]	Optimal	SymBA* [Torralba et al., 2014]
		Satisficing	IBaCoP2 [Cenamor et al., 2014]
2018	[Torralba and Pommerening, 2018]	Optimal	Delfi1 [Katz et al., 2018]
		Satisficing	FDSS 2018 [Seipp and Röger, 2018]

Table 2.1: Best scoring planners of the last IPCs.

3 Machine ethics

This chapter will serve to introduce the field and concepts of Machine ethics and review the state-of-the-art implementations. First, an overview of the history and context of the subject, its inception and motivations will be presented. A comprehensive description of the different taxonomies characterizing ethical agents will be elaborated, along with the distinct characteristics of each type. We will glance over different sources of codes of conduct that have been implemented and inspired different views on the field, and briefly mention some literature in which they were used. And finally, we will examine various state-of-the-art implementations of machine ethics.

3.1 History and context

Artificial Intelligence (AI) is the subfield of computer science that studies the development of intelligent computational agents [Russell, 2010], i.e: that can perform tasks that typically require human intelligence. In this field, automated agents are computational systems that can make decisions on their own without human intervention.

Automated systems were once developed for specific tasks and many times operated by professionally trained users. Currently, it is becoming more common for automated agents to work with minimal or no human oversight in contexts where their actions can have an impact on people. In addition, a user might not know that an interaction with an automated agent is taking place, or might not be aware of what parts or behaviors of a system have been automated. In consequence, the field of AI ethics was developed to prevent automated agents from having a negative impact on society when they are deployed.

There are two main ways that have been developed to alleviate this kind of concern. The first is by enabling or improving human oversight of automated agents by enhancing the transparency, algorithmic fairness, data privacy and reliability of their underlying decision-making systems. This is tightly linked to the field of explainable artificial intelligence (XAI) [Xu et al., 2019], which strives to develop models that increase the understanding of human users toward machines, notably when dealing with black-box systems [Adadi and Berrada, 2018]. The second approach, which is the one that concerns this thesis the most, is called machine ethics [Anderson and Anderson, 2007] and consists in making sure that machines behave ethically by enforcing certain behaviors while not requiring human oversight.

Machine ethics was conceived to study how to *automate moral reasoning.* Many of the definitions and notions regarding machine ethics were first discussed and coined during the 2005 AAAI Symposium [Anderson and Anderson, 2007].

Ethics and morality. Typically, the terms *ethical* and *moral* behavior are used interchangeably and are tightly associated with the idea of distinguishing between right and wrong, concerning some code of conduct put forward to describe a rational, or good-willed person [Gert and Gert, 2020]. However, different sources sometimes make a distinction [Harper, 2009]. For instance, some communities consider morality to be personal and that ethics should refer to the views of society as a whole. Yet, when describing the behavior of a machine, this distinction might not be necessary, as a machine does not distinguish between its values and those of society like a person would. This interesting question will nevertheless not be our concern in this thesis.

In the context of AI, an *ethical machine* is one that has been programmed to follow some given rules of conduct, or considerations, of what *correct* moral behavior is, according to society, in addition to other operational constraints, to make its behavior more acceptable. As such, the essential problems it poses for decision-making systems are (i) deciding what correct moral behavior is for the machine, and (ii) how to make the machine combine this information with the rest of the operational constraints in the decision-making process.

Some sources distinguish the concept of moral machines (capable of moral reasoning) from *ethically aligned machines* (discussed in Section 3.2), which in the words of the IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems [Shahriari and Shahriari, 2017], describes machines that function in an ethically desirable way, or at least ethically acceptable.

As we explained in the Introduction (see Chapter 1), determining what makes a decision or action correct can be a very complex problem if the machine ought to mimic human ethical reasoning. Ethical dilemmas, i.e:

problems in which someone is forced to choose between multiple unethical alternatives, serve to exemplify why it may be so difficult. Recalling Chapter 1, the Bystander trolley problem [Foot, 1967], which is the original of its many variations, can be described as follows:

> *A runaway trolley is on the way to kill five people tied to its current track. You are standing far away and next to a lever, which if activated, would change the trolley's path to a side track in which only one person lies. You have only two options, either to pull the lever or do nothing.*

Even to this date ethical dilemmas such as this and its different variations remain a topic of debate amongst philosophers [Bruers and Braeckman, 2014] and provide arguments to compare ethical theories, refine them, and demonstrate defective reasoning.

A seminal work sparking an interest in machine ethics and the overall application of ethics in AI was the 'Moral Machine experiment' [Awad et al., 2018]. There, the authors studied a crowdsourced corpus of information about many variations of the trolley problem in the context of automated driving agents, as depicted in Figure 3.1 and gathered over 40 million decisions from people of various countries and cultures. In doing so, they uncovered many nuances to the elicited societal choices, attributed to the cross-cultural variation of moral preferences, concluding that cultural traits and institutions have a great impact on what people think is moral or not.

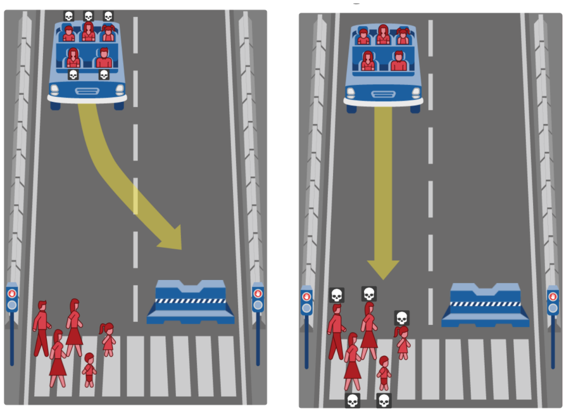


Figure 3.1: Moral Machine [Awad et al., 2018] example case.

This in turn has further shown just how complex automating moral reasoning might be and the kind of problems that can carry over to machines. If moral reasoning cannot tell us what decision is moral for such dilemmas, we cannot expect ethical theories to be applied to automated agents without additional reasoning.

While it is possible to link the trolley problem to the decisions an autonomous system such as a driver-less vehicle might be faced with (a discussion that has been the subject of many popular news articles [1] [2] [3]), the problem is that dilemmas such as this demand that we as a society take a stance about how we want machines to solve them, which again, depends on many cultural and societal factors.

Some bodies of research have since criticized the ideas behind the Moral Machine's mechanism of aggregating opinions for ethical decision-making, expressing concern about the incapability to explain decisions that have been inferred [Etienne, 2021] (as will be discussed in Section 3.4.2), and the danger of making policies for autonomous vehicles without sufficient structural context about each particular situation [Jaques, 2019].

It is expected that different cultures and institutions will have inconsistent priorities when faced with moral choices. A moral machine will typically be faced with such conflicts and be deployed in contexts where multiple sources dictate what counts as moral, e.g: stakeholders, users, experts, the law, and general societal rules.

Motivations. As for the motivation behind the study of this field, [Cave et al., 2018] propose four main benefits of giving machines some ethical understanding:

1. **Improving individual machine decisions**: this is the most typical setting of machine ethics and involves granting the machine ethical considerations in critical domains. In other words, domains in which there are ethical considerations and a possible negative impact on people's well-being, or uncertainty about the consequences of decisions. The main risks that might prevent a machine from taking ethical decisions in this setting are model inaccuracies, limited computational capacity and failure to capture or represent the severity of ethical considerations.

2. **Improving how machines justify decisions within a moral system**: because many people are doubtful about letting automated systems take over human decisions (for instance, in the case of autonomous driving vehicles), granting machines the ability to explain the ethical reasons behind their decisions can help them be considered more trustworthy. The biggest challenge, in this case, is to make the explanations sufficient, convincing and understandable to humans.

3. **Improving human moral decisions**: because human understanding and reasoning are not perfect, ethical machines could help people by acting as a decision-support system and pinpointing certain aspects between alternatives they do not take into account.

[1]https://www.nature.com/articles/d41586-018-07135-0

[2]https://www.newyorker.com/science/elements/a-study-on-driverless-car-ethics-offers-a-troubling-look-into-our-values

[3]https://www.computer.org/publications/tech-news/trends/ethics-safety-and-software-behind-self-driving-cars-in-the-aftermath-of-first-pedestrian-killed/

4. **Improving how people justify decisions within a moral system**: machine ethics could help develop improved ethical theories and conciliate views on moral dilemmas by forcing us to formulate these theories in a computable format. By doing this, ethicists would have new tools to verify the implications of moral theories automatically and therefore improve human moral understanding and decision-making. Concretely, by producing moral decision theories that mimic human behavior, underlying principles, biases and priorities behind human decisions can be uncovered, as well as inconsistencies when applying the same principles to other scenarios.

From a less practical perspective, some researchers [Anderson et al., 2005b, Berreby et al., 2015, Tolmeijer et al., 2020] have also suggested that studying machine ethics could lead to a better understanding of ethics itself by discovering which of their ideas are even applicable to artificial agents, how they can be encoded using formal frameworks and uncover inherent problems in some theories through automated reasoning.

Application domains. So far, machine ethical reasoning has been studied in a wide variety of application domains. One of the most popular domains in current literature is that of autonomous vehicles [Awad et al., 2018, Anderson and Anderson, 2018]. Closely related but with an even more critical set of risks is that of unmanned aircraft, which was studied in [Dennis et al., 2016]. Then [Anderson et al., 2006] studies the implementation of ethical reasoning in the context of healthcare robots by studying the task of administering medicine to patients through an automated agent and developing a system that uses supervised machine learning to create rules that discern whether the duty to protect the patient's health is more important than their autonomy. Furthermore, [Lindner and Bentzen, 2017] presents a robotic agent that is capable of solving ethical dilemmas through reasoning and displaying agreement or disagreement through words and gestures.

Future challenges. There are numerous challenges for the field machine ethics coming from both technical and non-technical dimensions [Anderson et al., 2006, Moor, 2011, Tolmeijer et al., 2020, Brundage, 2014]. From the perspective of ethics, [Moor, 2011] mentions the problem of overcoming our limited understanding of what an effective ethical theory for a machine could be and that domains like ethics are heavily nuanced by human notions because there is no agreement so far on how an ethical machine should act in every application domain. This is why the study of machine ethics and the development of both new theories for computational agents and reasoning mechanisms themselves are of utmost importance. Likewise, [Brundage, 2014] explains the difficulties posed by the plurality of ethical values people use to make decisions, their possible conflicts and how this might carry over to machines. The article also explains how, due to the computational limitations of computational agents, the complexity of the world and the lack of sufficient information, agents might not even be able to compute ethical courses of action in some scenarios. Similarly, [Malle and Scheutz, 2019] discusses the importance of developing new learning algorithms that can adapt general ethical norms to specific contexts due to the complexity and particularities of human norms, and creating new moral vocabularies to express moral judgments, explain decisions, or apologize in the case it is necessary. Furthermore, [Tolmeijer et al., 2020] stresses the importance of developing systems that can take into account and combine multiple ethical reasoning mechanisms, both from moral theories and practical domain-specific codes of conduct. In addition, the authors emphasize the need for domain-specific benchmarks, i.e: datasets with all the relevant cases of unethical behavior, based on input from domain experts, that permit to compare implementations between one another and verify their ethical alignment. They also highlight the need for algorithmic transparency, code usability and code availability to compare implementations easily. Moreover, [Baum, 2020] mentions three further challenges from a more technical viewpoint: determining the sources of opinion to be considered when eliciting moral considerations, (ii) how to understand/measure their views on these considerations, and (iii) which method to use to aggregate the opinions of different people. In short, the field of machine ethics is one that carries challenges coming from multiple sources such as modeling and reasoning with intrinsically human concepts, computational complexity issues and from a more technical perspective, the availability of implementations and benchmarks that can help verify and compare systems.

Having discussed the motivations and challenges of the field, we will now present some useful concepts and taxonomies that will help us understand and classify the different kinds of systems that have been developed for ethical machines.

3.2 Taxonomies

Provided that information sources about ethical behavior in a certain context are available, one question remaining is what makes the behavior of a machine moral, or rather, how can it act in such a way that people deem it as moral. This, in turn, will depend on many factors, such as autonomy, and societal impact.

The level of autonomy of systems can be divided between those which are operated by humans (controlled), those that perform fixed functions demanded by humans without intervention (automatic) and finally those that can make decisions on their own (autonomous). Depending on this level of autonomy, different considerations and levels of morality should be expected from a moral machine [Wallach and Allen, 2008]. Alternatively, a moral machine in some cases can help humans make decisions. That is, applying automated moral reasoning for decision support rather than decision-making.

Furthermore, Moor [Moor, 2011] recognizes four types of ethical agents, depending on the impact its decisions have on people and their awareness of their ethical considerations:

1. **Ethical-impact agents**: those which are deployed in contexts where ethical nuances are present and their behavior can affect people. For instance, cameras or traffic lights affect people's privacy and safety, respectively, but are not programmed to follow any moral behavior.

2. **Implicit ethical agents**: those that are programmed by a person to behave more ethically or avoid unethical conduct, but do not possess any real notion of ethical considerations. An example of such a system would be the automatic doors of a tram or an elevator, which sense if a person is standing next to the door, and would delay its operation until the person moves. Such a system would be considered to act ethically just by functioning correctly, regardless of whether or not it has an understanding of ethical considerations.

3. **Explicit ethical agents**: those that can reason and make decisions that depend on ethical considerations and possibly follow principles that are encoded in one way or another in their representation. This is the kind of agent we will be interested in this thesis and the one that the field of machine ethics aims to develop.

4. **Full ethical agents**: those that make and can justify their decisions over ethical considerations at the same level a person would. It is debatable whether a machine could show such a level of reasoning, given that a person's ethical understanding depends on various human notions that are far from the capabilities of current machines, such as consciousness, intentionality, and free will.

Both implicit and explicit ethical agents can be considered ethically aligned agents, as we mentioned in the previous section. In the case of this thesis, we will focus on explicit ethical autonomous agents, that is, agents that can perform tasks without human intervention and take ethical considerations into their decision-making process to prevent unethical behavior.

Moreover, systems implementing explicit ethical agency can either take ethical considerations at the same level as the other decision-making properties (as in [Anderson et al., 2005b, Lindner et al., 2017]) or do so in a different reasoning layer. This last approach is the one taken by *ethical governors* [Arkin et al., 2009]. Practically, it consists of adding an extra reasoning layer (possibly but not necessarily as an external agent) to automated agents to determine which of their actions might infringe on ethical values. Then, when such a situation becomes available, agents can select the most ethical path of action concerning their ethical reasoning layer, built on top of their existing decision-making layer. Some examples of this approach include the work of [Arkin et al., 2009], which embeds this extra layer within the agent and rejects action plans that violate ethical constraints. [Dennis et al., 2016] propose a framework for BDI agents that instead selects action plans which minimize ethical constraint violations.

Many different kinds of approaches have been developed for AI ethics implementations. Perhaps one of the most fundamental divisions in the current literature is that of [Allen et al., 2005], which separates approaches into:

1. **Top-down**: methods that explicitly seek to model ethical rules or implement ethical theories,
2. **Bottom-up**: methods that teach the machine to learn what is ethical from past experiences through learning, and
3. **Hybrid**: methods that combine both approaches.

We will cover a comprehensive amount of literature for each of these approaches in Section 3.4.

Top-down methods normally implement existing codes of conduct, for instance, coming from the law, rules of thumb, or moral theories, i.e: the branch of philosophy that studies the rightness of actions. They are particularly useful when there is a big amount of information about the domain and are typically designed using symbolic AI approaches, such as rule-based systems. An advantage, in this case, is that decisions can be justified using the underlying system and ethical considerations used to make decisions.

On the other hand, they carry the typical limitations of the implemented theory. In the case of ethical theories or the law, it is problematic to hold the decisions to some noble standard that is hard to apply to every situation without being inconsistent, or worse, rendering all decisions immoral. Additionally, they require the machine to be aware of both the ethical considerations and how to translate and use them in their theory. In some cases, this can include understanding various complex notions, such as possible consequences, causality and intentionality [Berreby et al., 2018].

Bottom-up approaches consist of building autonomous agents that can learn how to behave ethically by example. Similar to top-down approaches, it is necessary to provide the agent with information about the scenario and ethical considerations in the domain in which it operates. The difference lies in not modeling a code of conduct, but rather letting the agent learn from a corpus of data, given by one or many information sources, showing what is the appropriate ethical choice in many scenarios. The goal is to mimic human behavior with less structured information. Such learning approaches are typically implemented using statistics-based techniques coming from the field of machine learning.

These methods are well-known to perform properly with big and diverse corpora of inconsistent data, and as we shall see, domains with ethical nuances suffer many times from inconsistencies between different information sources and even in the same source in very similar situations, due to the intricacies of ethical reasoning. Yet, one significant difficulty is the elicitations of a corpus. Some approaches request the opinion of experts (ethicists), which can be a domain-dependent, slow and expensive solution, while others in the style of the moral machine, have resorted to a crowd-sourcing approach, i.e: consulting a large number of people, typically online, for payment or as volunteers. Lastly, verifying and explaining that a decision is ethically correct using a bottom-up approach is more difficult without an underlying ethical theory, which are highly desirable qualities in ethically-nuanced domains.

3.3 Sources of codes of conduct

As mentioned in the previous sections, many of the theories and concepts behind machine ethics implementations stem from other fields. This is particularly true for top-down approaches, which apply codes of conduct to constrain the behavior of a decision-making agent. We call codes of conduct the theories, mostly thought for human subjects, about which acts are deemed right or wrong. For the particular case of machine ethics, these theories often stem from moral philosophy, the law, or other theories, for instance, directly adapted from existing AI strategies.

Moral philosophy [Attfield, 2012] is the sub-field of philosophy that studies what makes actions right or wrong. Its theories provide different concepts and evaluation methods to come up with answers, or at least points of view, in situations where human values are essential. This field is often divided into three main branches: *meta-ethics*, *normative ethics* and *applied ethics*. Meta-ethics studies the meaning of the essential terms surrounding ethics, such as what justice, right, wrong, morality and truth are. Normative ethics focuses on developing and analyzing theories that can deduce what is right and wrong. Many of these theories, as we shall see, have been already studied in the context of machine ethics as frameworks for aligning machines into ethical values. Finally, applied ethics is the branch concerned with the application of moral concepts to practical real-life problems. This is highly related to the field of machine ethics, which deals with practical problems by computational reasoning.

In what follows, we will describe the main normative ethical theories and other approaches that have been used in machine ethics literature. Most of these theories have been studied and developed over the course of hundreds if not thousands of years, this is why, a complete account of all the philosophical subtleties and historical precisions surrounding them is avoided in favor of a more compact, yet sufficient presentation of the main ideas, based on recent surveys and articles.

3.3.1 Consequentialist ethics

Perhaps the most popular theory amongst the literature in machine ethics literature is consequentialism (see [Sinnott-Armstrong, 2021] for a recent overview). Some of its most notable proponents were the philosophers Jeremy Bentham (1748-1832), John Stuart Mill (1806-1873) and Henry Sidgwick (1838-1900). In this theory, actions are evaluated only upon their consequences irrespective of all other factors, i.e: it implies that the end justifies the means. The precise method to determine the rightness of action varies between branches of consequentialist ethics [Haines, 2006].

Some of the most prominent contrast points are:

- how to determine which consequences are good or bad,

- how are consequences of different actions compared,
- from which perspective the consequences are evaluated (oneself, other individuals, or both), and
- who judges the consequences.

The interpretation of these aspects characterizes the different branches of consequentialist ethics. One of the most typical elements of contrast between theories is that of the perspective of consequence. While *utilitarianism* [Mill and Bentham, 1987] judges an action right if it maximizes the well-being of every party involved (oneself and other people), *egoism* only takes into account the happiness of oneself and *altruism* considers every else but oneself.

Another point of contention is that of determining through which moral factor to measure the rightness of consequences. Classic consequentialist theories take into account a broad view of these factors. In contrast, *hedonism*, also called hedonistic act-consequentialism, regards pleasure (and conversely, pain) as the most important moral pursuit of individuals in a decision, disregarding other factors such as freedom and justice.

Another distinction can be made between *act-consequentialism*, which considers an action to be right when it results in more good than with any other alternative, and *rule-consequentialism*, perhaps an attempt to reconcile utilitarianism with deontological theories (which we will describe in the next section), that states that an action is right if it adopts rules whose adoption would bring about the best consequences.

Of course, many other theories, assumptions and combinations of the mentioned concepts exist and denote alternative views on consequentialist ethics.

If we consider the trolley problem seen before, consequentialism typically chooses to sacrifice one person to save the other five, as that would produce the most amount of well-being if all lives are held into equal consideration. As another example, many people would agree that lying is inherently bad, but that provided that lying could save lives, then the end could justify the means. This is precisely the stance of consequentialist theories, as it only values the consequences to determine the rightness of actions.

In the case of machine ethics, this theory can prove one of the most straightforward to implement [Anderson et al., 2005a, Winfield et al., 2014, Lindner et al., 2017, Lindner et al., 2019, Berreby et al., 2017, Bonnemains et al., 2016], because decision-making systems naturally model the consequences of actions. On the other hand, some of the problems of applying this theory come down to measuring the goodness of consequences on ethical terms, the fact that some consequences might be incomparable, how to measure the well-being brought about by consequences, and how machines consider different people's well-being.

3.3.2 Deontological ethics

Deontological ethics (see [Alexander and Moore, 2021] for a recent overview) is a normative ethical theory that assesses the rightness of actions according to duties and obligations that should always be followed. As opposed to consequentialism, deontological ethics considers that the consequences of an action do not that make it right or wrong, but rather the duties of action that must be followed. Typical examples of obligations are ‘not to kill’ and ‘not to lie’. Its ideas can be traced back to divine command theory [Quinn, 2013] (following the will of God) and are famously known by Immanuel Kant’s (1724-1804) notion of *categorical imperative*. According to him, a person should ‘act only according to the maxim (known as maxim-of-will) by which (s)he can also will that it would become a universal law’. In other words, to act in such a way that one would accept anyone else to act in such a way themselves.

Many branches of deontological ethics exist, but they all retain the view of judging the rightness of one action with respect to certain *norms* (i.e: rules of conduct), that convey human duties and obligations. A distinction for deontological theories can be made between *agent-centered*, which focuses on the permissions and obligations of the person acting, and *patient-centered*, i.e: those whose rules of conduct center around the rights of the people affected by the action, e.g: the right of other people to not be used as a means to an end.

Apart from the classical interpretation, *contractarian ethics* [Sayre-McCord, 2013] interprets deontological norms to be derived from social contracts and mutual agreements. A social contract in this sense would be a pre-defined agreement among a set of individuals about the norms and requirements that their actions should follow. Furthermore, *divine command theory* [Quinn, 2013] is a type of deontological theory that states that the rightness of actions is to be determined by the will of God and norms, obligations and permissions are commanded by God.

The most classical interpretations of deontological ethical theories typically consider turning the lever in the trolley problem to be inadmissible, as killing will always be a prohibition. A more detailed interpretation, however, could provide norms that take additional considerations for particular cases like this one, in which all choices end with the death of someone. It is easy, nonetheless, to see how providing norms for every possible situation in this manner would become cumbersome and possibly lead to exceptional cases with undesirable results.

The main benefit of deontological theories is their consistency, i.e: the norms are established in advance, can be applied in any situation and serve as a justification for all actions. On the other hand, many deem it too inflexible and argue that by sticking to norms, consequences and the immediate welfare of people are ignored. Also, as mentioned before, providing norms for every possible situation is a difficult task in itself. Along with consequentialist ethics, it is one of the most popular theories for AI systems [Ganascia, 2007, Lindner et al., 2019, Noothigattu et al., 2018, Bringsjord and Taylor, 2012].

3.3.3 Virtue ethics

Virtue ethics (see [Hursthouse and Pettigrove, 2018] for a recent overview) is a theory that, in contrast to deontological and consequentialist theories, places all the importance on the characterization of the person acting. Its ideas can be traced back to Aristotle (384 BCE-322 BCE). Instead of emphasizing consequences and rules, virtue ethics judges an action to be right if and only if the person acting is expressing good will and moral values. When faced with a decision, the basic question it uses to determine rightness is: ‘would this action be what a virtuous person would do?’. These virtues should contribute to social welfare and harmony. Some examples include generosity, courage and moderation. Of course, different groups of people would hold different virtues in higher or lower regard depending on their culture, e.g: while the ancient Greeks would hold pride as an important virtue, medieval Christians preferred humility instead.

Many bottom-up machine ethics approaches, which teach a computational agent to behave ethically through many examples of human decisions can be seen as implementing virtue ethics. However, it can also be argued that because such mechanisms do not have a direct understanding of virtues, they do not fit in this category exactly. Some attempts to materialize the ideas of pure virtue ethics can be seen in [Howard and Muntean, 2017, Gamez et al., 2020, Hegde et al., 2020, Cointe et al., 2016, Thornton et al., 2016].

3.3.4 Prima facie duties

Prima facie duties [Ross and Ross, 2002] are what is known as a *pluralist* moral theory. They were conceived by William David Ross (1877-1971). According to this theory, there are a given set of moral duties/values that should be considered:

- Fidelity: strive to keep promises and be honest.
- Reparation: make amends when we have wronged someone.
- Gratitude: repay others when they perform actions that benefit us.
- Non-maleficence: refrain from harming others in any way.
- Beneficence: improve other people’s health and well-being.
- Self-improvement: improve our own health and well-being.
- Justice: be fair and try to distribute benefits and burdens evenly.

In his work, Ross suggests that this list is by no means complete and that other duties could be added in certain situations.

Furthermore, the rightness of actions should be compared with respect to these duties. This can be seen as a particularist (i.e: case-by-case) view on deontological ethics and can also be linked to virtue theory, as these duties also identify desirable values about who acts. Of course, not every duty is as important as the others in every situation, for example, non-maleficence is almost always to be prioritized, as harming other people should be avoided in all decisions. Moreover, prima facie duties force actions all other things being equal, i.e: if an action supports a more important duty or the same duties that another action plus some, then that action is deemed right.

Prima facie duties in the context of machine ethics have been studied, among other literature, in [Anderson et al., 2005a, Anderson et al., 2005b, Anderson and Anderson, 2018].

3.3.5 Doctrine of double effect

Aside from the main moral theories, another interesting concept for the implementation of ethical machines is the doctrine of double effect (see [McIntyre, 2019] for a recent overview). It is understood to have been introduced by Thomas Aquina (1225-1274), an Italian priest. Essentially, this doctrine captures the conditions under which it can be considered morally acceptable to bring upon negative consequences as a side effect of an action that has overruling good consequences. Although there are many interpretations for it, an action is considered ethical if some variant of the following conditions is met:

1. the action in itself must be good or indifferent,
2. the agent must only intend the good effects and if there are any bad ones, s/he would rather be in a situation where the action would not be needed,
3. the bad effects do not cause the good effects by themselves, rather they are both produced by the action independently, and
4. there must be a proportionally important or desirable reason to permit the bad effects in light of the good ones.

That is, the doctrine of double effect combines some of the ideas of consequentialist ethics, as it considers the effects/consequences of an action, with the causality between the effects and the intentions of the agent performing the action.

While on one hand, this theory is useful to distinguish and prevent cases in which a person is used as a means to an end (due to the third point), it increases the modeling complexity of a domain in the context of machine ethics, as a system has two additional properties to capture: causality and intentions.

Some of the known work on machine ethics implementing this theory include [Bonnemains et al., 2016, Govindarajulu and Bringsjord, 2017, Lindner et al., 2019], described in Section 3.4.

3.3.6 Other theories

Another two mechanisms that can be directly adapted to ethical reasoning in the context of AI are the *do-no-harm* and *Pareto* principles [Lindner et al., 2017, Lindner et al., 2019]. Both principles can be seen as consequentialist as they only consider the outcomes of an action to determine their permissibility. Briefly, the do-no-harm principle states that an action is permissible if none of its consequences are bad. Instead, the Pareto principle is used to compare actions and characterizes an action as morally as preferred as another when the positive consequences of the first are a superset of the ones of the second and the negative consequences of the first are a subset of those of the second.

Notably, some literature [Anderson, 2008, Vanderelst and Winfield, 2018] has also examined modeling the three laws of robotics by [Asimov, 1950], namely: (i) do not harm people or allow harm through inaction, (ii) obey orders from humans except in the case they conflict with the first law, and (iii) protect the machine's wellbeing as long as this does not interfere with the previous two laws. Both the author of the original work and further research has later criticized different aspects of these laws, particularly the fact that absolute-law-based frameworks which ignore how the laws are satisfied can have undesirable results e.g: a robot preventing people to leave a house to prevent any harm.

3.4 Implementations

While selecting what theory to use for AI systems is a problem in itself, research has already begun to explore how the ideas of these fields could be used in AI and the possible problems it might face.

In what follows, we will describe the state-of-the-art research that has been developed for the implementation of ethical machines. For clarity, we have separated the literature between top-down, bottom-up and hybrid systems. Moreover, each of these sections has been further divided into decision-making and planning systems. Although planning will be our main focus, most concepts and ideas from decision-making systems remain relevant for our work.

Regarding existing surveys, [Charisi et al., 2017] discuss the main challenges of implementing machine ethics, [Brundage, 2014] pinpoint the practical and theoretical limitations automated agents might face when implementing ethical theories and [Yu et al., 2018, Dennis and Fisher, 2018, Tolmeijer et al., 2020] provide extensive descriptions of existing implementations and future challenges with a focus on many useful taxonomies.

Before we delve into the state-of-the-art systems, let us discuss some of the concepts and research currents that can help distinguish these implementations from one another.

While some of these systems implement ethics using values and consequences, some works [Govindarajulu and Bringsjord, 2017, Lindner et al., 2019, Bonnemains et al., 2016] have explored additional notions that are essential to some moral theories and could prove to be useful for machine ethics systems, such as:

- **Causality**: assessing which actions cause certain effects immediately or in the long run,

- **Intentionality**: determining which of the consequences of an action were the actual goal of the acting agent,
- **Proportionality**: taking into account which negative effects are proportional to positive effects, which may or may not justify producing both in certain scenarios, and
- **Side-effects vs. means to an end**: identifying whether realizing some bad effect causes a desirable effect (i.e: a means to an end), or if rather the bad effect is a mere side-effect.

As an example, one may think about a situation in which two robots are moving through a platform and one of them is about to fall through a hole. If the first agent collides with the second, this would prevent it from falling. Here, the agent colliding with the other would cause the second to avoid falling through the hole, the intention of the first would be to save the other agent rather than to collide, the consequence of colliding would be proportional to that of saving the other agent, and colliding would be the means to the end of saving the second agent.

Causality for decision-making systems has been modeled using modal logics [Bonnemains et al., 2016], or specific frameworks, such as causality networks [Lindner et al., 2017]. In the case of planning frameworks, causality normally poses many questions regarding its interpretation because agents perform many successive actions, and thus the effects of certain actions can have repercussions on the available set of future actions and the ending world state. A study on different interpretations of causality in planning can be found in [Berreby et al., 2018]. Furthermore, differentiating side-effects from a means to an end often requires modeling causality between effects by definition, and has been used in practice to describe some moral theories, such as the doctrine of double effect, that apply these notions to have a more detailed account of the reasoning behind decisions [Govindarajulu and Bringsjord, 2017, Lindner et al., 2019, Bonnemains et al., 2016]. Moreover, deciding how to model intentions and proportionality is not always straightforward. At least, all of the system's goals are forcefully intended outcomes, however, the rest of the effects produced by the actions of an agent can be intended or not. This is why annotating intended outcomes [Govindarajulu and Bringsjord, 2017] and possibly other reasoning mechanisms are required to model intentionality. To describe proportionality, some works have resorted to specific constructs [Bonnemains et al., 2016], or by directly comparing the utility of good and bad consequences [Govindarajulu and Bringsjord, 2017].

3.4.1 Top-down ethical systems

As we explained in Section 3.2, top-down systems address the risks posed by the presence of autonomous agents in everyday life by designing machines equipped with ethical understanding, i.e: knowledge about values and considerations that can enable them to separate right from wrong in the particular scenarios they are being deployed.

Decision-making As an early work, we can find in [Anderson et al., 2005a] a top-down implementation of hedonistic act-utilitarianism (see Section 3.3.1), that the authors called JEREMY. In their work, they describe a decision-making system in which the users can input the consequences for various actions along with their perceived pleasure, the likelihood of occurrence and the people affected. In turn, the system quantifies the overall utility (pleasure) of each possible action and determines the best out of them. While their system acted as a starting point to test different theories, their limited reasoning capabilities and high dependency on detailed input data (such as the perceived pleasure and likelihood of consequences) make it impractical for real-world application.

Then in [Lindner et al., 2017] the authors presented a system called HERA, which implements various ethical theories and is available online[4]. This system is capable of reasoning using different ethical theories and combining their capabilities. Some of the represented theories include the doctrine of double effect, utilitarian consequentialism and an adaptation of the Pareto principle. The underlying formalization and these ethical theories are represented using a version of causal networks [Halpern, 2016], which is capable of modeling intentionality and causality. Their system provides both a framework to test ethical theories through various scenarios and a standardized format for representing situations through causal networks in a simple manner. This system is also integrated into the architecture of the robotic agent IMMANUEL [Lindner and Bentzen, 2017]. As shown in this work, this robot is capable of receiving ethical dilemmas and human judgments and displaying either agreement or disagreement through speech and facial expressions, along with its motives.

Another early work is that of the SIROCCO system [McLaren, 2003], which presents a formal model, called the 'Ethics Transcription Language' (ETL) in order to define ethical principles, along with a symbolic representation-based mechanism to retrieve the relevant principles for a decision in different situations. Developing standardized languages for ethical domains and principles is indeed a necessary step forward for the field to this day to compare implementations. On the other hand, because both the ethical principles and situations need to be encoded using the same strict representation, it could be hard eliciting all the ethical principles for a particular domain in practical scenarios.

Notably, a great deal of machine ethics systems have been designed for the Belief-Desire-Intention (BDI) [Rao et al., 1995] model. The BDI model is very well known in the autonomous agent community. Autonomous agents implementing this framework maintain an account of the three aforementioned elements: a representation of the knowledge about the current situation in the surrounding world (beliefs), the state of affairs it wants to hold in this world (desires), and the actions that the agent has committed itself to do in order to achieve the goal (intentions). In comparison to the other mentioned approaches, BDI agents perform reasoning at a high level of abstraction. The actions required to achieve a goal for a BDI agent are

[4]https://github.com/existenzquantor/ethics

typically obtained via an external module, e.g: a planning library. That is, instead of computing which actions can be performed to reach their desires, a BDI agent obtains various alternatives and selects one of them.

In the context of machine ethics, some literature has already explored the idea of using BDI agents to compare multiple plans on ethical terms after they have been computed [Bremner et al., 2019, Cranefield et al., 2017, Ganascia, 2015, Honarvar and Ghasem-Aghaee, 2009]. The ETHAN system [Dennis et al., 2016] presented a mechanism to order plans on ethical terms after they were computed externally and annotated with ethical considerations. The situations revolved around unmanned aircraft systems and the corresponding annotations concerned possible collisions and violations of the Rules of the Air. As another case of study, in [Cointe et al., 2016] the authors introduce an abstract framework for ethical reasoning designed for BDI agents, present a set of constructs to capture several ethical concepts and compare plans on ethical terms, and show how to extend these notions to a multi-agent setting.

The benefit of the BDI approach is that it is ideal for implementing ethical governors by comparing these courses of action on ethical terms, resulting in an appropriate setting for ethical reasoning. On the other hand, because these courses of action are not generated directly by an ethical reasoning mechanism, the system can lose finer-grained control of its actions. If the action/plan generating module fails to provide the most ethical course of action, the BDI agent will only be able to compare weaker alternatives.

In one of the few adaptations of divine-command ethics (see Section 3.3.2), [Bringsjord and Taylor, 2012] proposes a logic system where human inputs are seen as divine commands by lethal autonomous robots to deduce which actions should be permissible in the military domain.

Another logic system that has been used to compute ethically-aware decisions is that of non-monotonic logics [McDermott and Doyle, 1980]. [Ganascia, 2007] shows how to encode ethical decision theories inspired by Aristotelian ethics [Broadie and Rowe, 2002], the deontological maxim-of-will (see Section 3.3.2) and Constant's 'Principle of Politics' [Constant, 2013] using answer set programming [Lifschitz, 1999] as a deduction mechanism.

Also using a logic-based model, [Bonnemains et al., 2016] formulates a decision-making framework that models causality and proportionality through special logic-based and preference-based constructs and implements the moral theories of consequentialism, deontological ethics and the doctrine of double effect.

Regarding morality as preferences, [Awad et al., 2020] demonstrates a procedure to infer cases in which people would find it acceptable to break societal rules in exceptional cases using CP-nets [Boutilier et al., 1999] as their preferences model. Also, in [Loreggia et al., 2018], a measure of distance between CP-nets and an approximation algorithm are described, which the authors use to compare how close the preferences of a system are with different ethical principles, also encoded using CP-nets.

Planning In order to represent problems involving planning, various frameworks have been used, each with its benefits and capabilities. Some of the classical planning languages that have been used for machine ethics implementations include PDDL [Fox and Long, 2003], STRIPS [Fikes and Nilsson, 1971], SAS+ [Bäckström and Nebel, 1995] and the event calculus [Shanahan, 1999]. In Section 2.2.2, we have presented a model for capturing classical planning problems with the PDDL language. STRIPS, as we mentioned, was the language developed for the first planning system, and can be considered a more restricted version of PDDL. SAS+ extends STRIPS with multi-valued fluents (i.e: states are represented by a set of value assignments to their properties, instead of a set of properties alone) and conditional effects, which restrict some of the effects of actions with additional conditions. Most versions of PDDL also support these two features through extensions [Fox and Long, 2003]. Moreover, the event calculus is a first-order logic-based calculus for representing planning domains. It represents states, actions and their effects using first-order logic rules, axioms and special predicates in such a way that verifying which fluents hold in a particular state can be reduced to logical consequence queries. Concretely, the event calculus uses explicit timepoints (a special kind of constant) to represent the successive states reached after performing actions. In comparison to the other mentioned models, relying on first-order logic results in a more formal representation and makes the system easy to verify. On the other hand, reducing planning to logic queries is almost always less efficient than the highly optimized algorithms used for PDDL-like languages.

Another research current that can be tightly linked to machine ethics, is that of *deontic logic*. Deontic logic [Von Wright, 1951] is a formal system that extends classical logic, either propositional or first-order, with a set of special (modal) symbols to represent concepts such as permissibility, necessity, obligation and prohibition. At its core, it is a formalization that is inspired by the ideas surrounding deontological ethics and permits reasoning about information to infer what actions or states of affairs should be permitted or not.

Using the event calculus, [Hashmi et al., 2014] shows how to model obligations and prohibitions, two concepts central to deontic logics [Von Wright, 1968]. Similarly, [Marín and Sartor, 1999] provides a formalization for norms and their fulfillment status (also for event calculus) in Prolog syntax.

Then [Berreby et al., 2015, Berreby et al., 2017] show how to represent various ethical theories in planning problems modeled using a variant of event calculus [Shanahan, 1999] and implement them using answer set programming [Lifschitz, 1999]. The same authors continue their work in [Berreby et al., 2018] by identifying various notions by which the agent can be held responsible for unethical behavior and how to compute them. Their work describes an interpretation of the concepts of causality and intentionality and an approach to encoding them through a logic system to infer different types of causality between actions and effects.

Furthermore, in [Lindner et al., 2019], the authors present a characterization of various ethical decision mechanisms such as utilitarianism, different versions of the do-no-harm principle and the doctrine of double effect in the context of the SAS+ planning model. Their formalization permits taking into account causality differently than their previous work on the HERA system [Lindner et al., 2017], which we described before, by analyzing the dependencies between the consequences of actions in a plan.

A definition and formalization of context-dependent norms for STRIPS-based planning domains and an implementation for PDDL are provided in the work of [Panagiotidi and Vázquez-Salceda, 2011].

Also using formal logic-based models, [Govindarajulu and Bringsjord, 2017] defines a variant of event calculus to capture different versions of the doctrine of double effect, mentioned earlier.

Perhaps closer to the field of cognitive agents, [Winfield et al., 2014, Vanderelst and Winfield, 2018] describe an implementation of ethical consequentialism for robots through simulation. Their model, inspired by [Vaughan and Zuluaga, 2006, Bongard et al., 2006], implements a consequence engine (i.e: the simulation module) that can be used by a robotic agent to compare different courses of action through simulation and determine the best one before performing one of them. Through experimental evaluation, they show how their system can make a robot both maintain its safety, by avoiding a hole and also save another robot from the same danger by provoking a collision avoidance response when necessary.

Related to the concept of norms, an extension for the language C+ (which defines transition systems) is presented in [Sergot, 2004], in which fluents and actions can be forbidden under specified circumstances. States and transitions are thus labeled permitted or not, according to whether any of those rules are broken. In addition, they define these concepts with a special focus on institutions and their power to establish these permissions.

3.4.2 Bottom-up ethical systems

Bottom-up ethical systems work by inferring what ethical behavior is from example, via some form of machine learning. As opposed to top-down systems, their reasoning is opaque (sometimes called black-box) because it is not immediately possible to deduce the reasoning behind the machine's decisions since its decisions are determined by copying some exemplary behavior. In addition, systems are harder to verify. Without further constraints, it can be hard to rely on bottom-up ethical systems from a human perspective, as there are no hard guarantees that unethical behavior will not happen. This can be of course compensated in a variety of ways (e.g: via the ethical governor approach). On the other hand, machine learning systems are known to be very effective and often surpass their transparent-reasoning counterparts.

In any case, the machine ethics community has already developed bottom-up systems that can learn ethical behavior from examples. These examples are often elicited from domain experts (ethicists) or via crowdsourcing. In this section, we will describe some of the most notable ones so far.

Decision-making A learning-based consequentialist approach to machine ethics is developed in [Armstrong, 2015]. This work proposes a framework in which an autonomous agent acts following preferences, modeled as utility values, which can be updated as new information is gathered. When actions are performed, the agent receives feedback about their consequences and therefore updates the perceived utility values of its available actions.

The idea of applying a contractarian approach (see Section 3.3.2) in autonomous systems to judge decisions ethically can be materialized through social choice [Arrow et al., 2010]. Social-choice theory deals with the development and analysis of mechanisms to aggregate opinions by the means of voting. As such, many of its mechanisms can be used to reach norm or preference agreement. Regarding social choice as an approach to establishing ethical norms in the spirit of contractarian ethics, [Noothigattu et al., 2018] presents an algorithm to learn societal preferences in ethical dilemmas aggregating their opinions based on a voting procedure and the data collected from the Moral Machine experiment [Awad et al., 2018] discussed previously. Then in [Baum, 2020], the author discusses the importance of clearly defining the entities or people who should have the standing to vote in the social contract and its consequences (as morality is usually culture-dependent), choosing an appropriate method to elicit the votes and choosing an adequate voting procedure (social choice provides many different ways of doing this [Arrow et al., 2010], with different benefits and shortcomings).

Then, in [Malle et al., 2017] the authors develop a deontic-logic-based model for norms that can be learned through machine learning procedures and could be applied to symbolic decision systems.

Also using machine learning methods, [Hegde et al., 2020] presents a simulation-based framework inspired by the cellular automaton model [Wolfram, 1984], in which different agents interact with one another and learn how to behave by donating and stealing from one another. By modeling their gains and the perceived morality of other agents, they infer how to behave to maximize their gains, in a consequentialist fashion, and show good character, like in virtue ethics, if they want to maximize their utility in the long run.

Planning In the context of probabilistic planning [Russell, 2010], [Abel et al., 2016, Wu and Lin, 2018, Rodriguez-Soto et al., 2021] propose adaptations of the Markov Decision Process [Puterman, 1990] model for learning which actions are unethical from past examples by contemplating, in the reward function, additional penalties for those actions that violate ethical behaviors.

3.4.3 Hybrid ethical systems

Hybrid ethical systems combine some of the techniques of both top-down and bottom-up approaches. In this type of framework, agents are equipped with pre-defined ethical notions, principles, or theories, while a learning module is tasked with inferring the precise way to use this knowledge to mimic human behavior from past experiences.

Decision-making One of the first implementations of an ethical system was Arkin's ethical governor [Arkin et al., 2009, Arkin et al., 2011]. It aimed to constrain automatic war systems to enforce the International Laws of War (LOW) and Rules of Engagement (ROE). By adding an ethical governor module to weapon systems, the decisions of the underlying system could be vetoed and new decisions could be demanded. Since then, this approach has remained an effective way of providing ethical understanding to black-box (e.g: machine learning-based) systems to ensure that the agent performs ethical actions. The author suggests a hybrid approach: implementing the LOW and ROE as constraints (top-down) and adapting them (in a limited way) according to the results of actions through bottom-up methods.

In conjunction with the abovementioned JEREMY system, in [Anderson et al., 2005a] the authors also present an alternative decision-making approach that they call W.D., in honor of W.D. Ross, the philosopher behind prima facie duties (see Section 3.3.4). This system uses inductive logic programming [Muggleton, 1991] to make ethical decisions taking into account prima facie duties in different situations. Because actions can privilege distinct duties (e.g: justice vs. non-maleficence) and only some duties do not supersede others, their implementation takes into account both possible conflicts between duties and their severities to assess the correct answers, by producing rules that determine which combinations of duties and severities are more important to uphold than others. Then in [Anderson et al., 2005b, Anderson et al., 2006] they continued this research direction and developed the MedEthEx, which applies the framework behind W.D. to the medical domain. Their system also utilizes inductive logic programming to learn rules but uses prima facie duties specific to the medical domain: Ross's duties of beneficence, non-maleficence, and justice and also the domain-specific duty of autonomy. With this system, medical workers can be advised on ethically ambiguous scenarios such as cases in which the patient refuses an effective treatment and debates between accepting the patient's judgment or trying to convince them through other means. Later, in [Anderson and Anderson, 2018] they present the GenEth system, which applies this same mechanism in the domain of autonomous driving and infers rules through a corpus of past cases constructed with the advice of ethicists.

Using Inductive Logic Programming [Muggleton, 1991] as an inference mechanism, [Dyoub et al., 2020] shows how to learn logical theories, encoded using Answer Set Programming [Lifschitz, 1999], defining ethical behavior in domains with pre-existing knowledge and a corpus of decision-making scenarios with examples of human judgment about the rightness their alternatives.

An alternative approach to learning from ethical dilemmas [Bringsjord et al., 2016] proposes a logic-based framework where machines can learn from interacting with other agents to discover ‘counteridenticals’, i.e: ‘if I were you’ situation reversals, and update their behavior in the case both the other agent and the machine agree.

Finally, in [Chaput et al., 2021], the authors present a hybrid multi-agent framework for ethical reasoning using the BDI model, in which two kinds of agents (judging agents and learning agents) interact to judge and learn ethical behavior. The judging agents are implemented in a top-down manner along the lines of [Cointe et al., 2016] and use symbolic constructs to decide which actions are deemed moral, immoral or neutral according to pre-defined consequentialist and deontological principles. Then the learning agents, represented using an adaptation of the Markov Decision Process model, learn from multiple judging agents what is considered moral in a bottom-up fashion.

4 Representing ethical preferences in classical planning

As a consequence of our increased understanding of what machines can do, the development of the various AI sub-fields and the worldwide availability of big data, agents can now take decisions in certain specialized fields without any human intervention. Previously, automated agents were restricted to a working environment in which the possibility of causing harm was limited. This, however, is being gradually challenged with the advent of intelligent systems with increasing levels of autonomy. For instance, some robots have been developed in order to help people with everyday tasks [Bemelmans et al., 2012], drive autonomously [Levinson et al., 2011] and perform surgery [Diana and Marescaux, 2015].

In chapter 3, we overviewed the field of machine ethics and many of the ways in which it can help align automated agents with ethical notions to avoid undesirable or harmful behaviors. We described various state-of-the-art implementations and the underlying moral theories and concepts by which they are inspired.

Research in machine ethics can not only assist in improving the ethical understanding and the decisions made by autonomous systems but can also help us understand what it is for a machine to behave ethically and what kind of ethical machines we want. Namely, by developing ethically aware machines and encoding decision systems through computational AI frameworks, we can discover inconsistencies in their ethical reasoning and potentially dangerous outcomes as a result of them. On the other hand, we can also identify which kind of ethical theories work well and in what conditions, and compare their reasoning to our own human ethical perceptions of rightness.

Our objective in this chapter is to develop a domain-independent and adaptable framework for computational agents that allows them to understand situations on ethical terms and reason consequently about the rightness of actions in terms of preferences, going further than classifying actions as right or wrong, which would only allow preferences between two classes of actions. We will be focusing on developing a top-down system to explore several moral theories and aim to describe their decision principles through preferences. As we discussed in Chapter 3, most of the existing top-down implementations have concentrated on determining which actions are right or wrong through the moral theories described in Chapter 3. Here, we set out to develop a domain-independent framework, this means that we need to be able to cover several trends of machine ethics because it seems to be the case that depending on situations, we might need (or people seem) to apply different moral theories or combinations of concepts from them, for instance, in the presence of sacred values [Tetlock et al., 2000]. We chose to use preferences because the machine must be always able to make a decision. Most ethical frameworks reviewed in Chapter 3 only assess actions qualitatively (right or wrong), which risks having no ethically right action as a result. This is made obvious in ethical dilemmas, i.e: situations in which no action is perfectly ethical, where typically no action is ethically right (like in the trolley problem). Hence we want to explore the notion of ethical consideration as preferences. Moral theories can serve as a starting point for research to test pre-existing ethical concepts and see how they fare against well-known ethical dilemmas. Then, by working with preferences, we expect to refine existing machine ethics concepts in such a way that a machine can take ethically aware decisions even in ethical dilemmas, while still incorporating the essence of the various moral theories that it is aware of. Some of the previous work addressing preferences in machine ethics include [Awad et al., 2020, Loreggia et al., 2018], although they focus more on bottom-up approaches that can infer human ethical preferences encoded with CP-nets.

We chose to develop our top-down machine ethics framework from the perspective of classical planning, which we described in Chapter 2. By adopting a planning model, we gain the capacity to address an extremely general context. In particular, we can reason about situations in which various consequences unfold as a result of different actions. This is useful in ethically nuanced scenarios, as we can analyze the ethical features of actions and consequences in the long run. Classical planning provides a simple setting to consider unfolding consequences, in which we can abstract operational complexities, such as sensor input and plan execution failure, to focus solely on the ethical elements. In addition, by adopting classical planning as the base of our framework, we gain access to state-of-the-art planners to generate our plans, which has been the subject of a large amount of research, as we also reviewed in Chapter 2. Using classical planning to develop top-down machine ethics frameworks has also been studied before, as we reviewed in Chapter 3, but not by using preferences as we will do here, to the best of our knowledge.

Our treatment of ethics for autonomous agents will be slightly different from past research. Our framework will consider ethics in a different layer from operational requirements. Indeed, considering ethics in a layer above operational requirements has been studied under the model of ethical governors [Arkin et al., 2011] which has been applied to BDI systems [Dennis et al., 2016]. However, our framework will not take the same perspective as ethical governors, which delegate the action/plan generation to a different module and then compare plans on ethical terms. Rather, our approach will consider ethical notions in the action/plan generation itself but will separate the goals and constraints that are operational from those that are ethical. Much past research, such as most of the literature we covered in Chapter 3, considers consequences and constraints that are ethical at the same level as those that are not, and we wish to avoid this because separating the ethical reasoning from the rest of the planning model makes our framework modular and easy to extend to other planning or decision-making models. Additionally, we want to separate what makes an action or plan correct and what a goal is, from all the ethical notions we will describe in our framework. For instance, if a robot was tasked to go from one location to another, and could only do so while bumping into a person and we consider this unethical, we want to separate the notion of the plan being unethical from the task of finding a course of action to go to the requested location. Indeed, if part of the goal is not bumping into a person, no plan will be found. Rather, we want the planner to find all possible plans to go to the location and only then compare them on ethical terms. If then all the plans bump into a person, it is up to the agent to decide whether to execute a plan or not. One might then ask why to bother computing plans that are not perfectly ethical. However, in the context of an autonomous system, the machine might always need to do something if any plan to achieve the predefined objectives exists. If no unethical plan is computed, then it is stuck and hence the situation is at an impasse. An empty plan, which amounts to not doing anything, can be unethical too. In that case, it seems appropriate for all the plans to be considered and compared. So our approach will be to consider and compare all plans. Furthermore, some ethical theories, such as consequentialism (see Chapter 3) will, by definition, need to compute the final state of plans, since their end result needs to be quantitatively assessed and compared between one another.

As a last detail, it should be noted that a practical application of our research can be considered, where one can envisage the sensory input and real-world execution of the actions being handled by an abstract low-level symbolic translation module as depicted in Figure 4.1. However, in this thesis, we will focus on planning on a high level of abstraction to investigate machine ethics. Indeed, by focusing on a high level of abstraction, i.e: the symbolic planner in Figure 4.1, we can concentrate on the ethical reasoning aspects and explore all of the notions we have described symbolically.

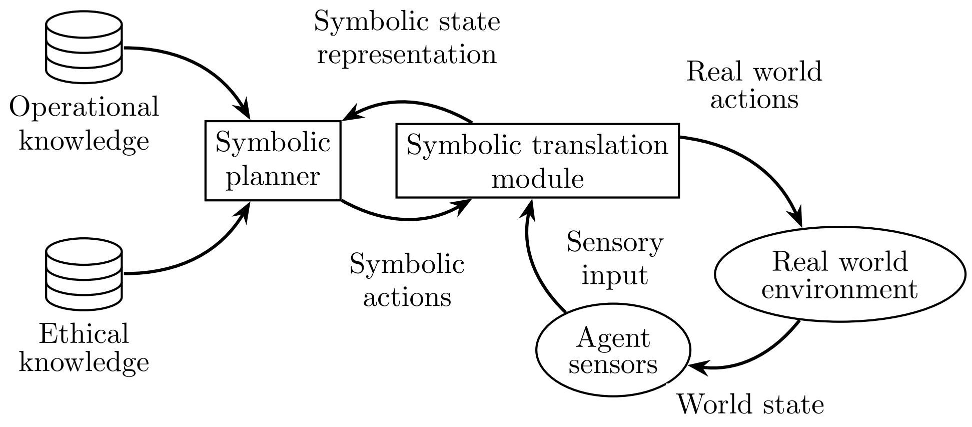


Figure 4.1: Top-down architecture overview.

In short, we want to develop a framework capable of:

- Capturing many of the relevant theories that have been implemented for ethical reasoning,
- Refining morality as a choice in terms of preferences, instead of right and wrong,
- Profiting from existing state-of-the-art technology to determine its actions, and
- Being applied on top of existing implementations, that is, providing an approach that works as an additional layer to the existing operational model and requirements.

As such, our research questions for this chapter are the following:

Research Questions in this Chapter

- *How can we represent ethical considerations in classical planning problems in such a way that it is possible to determine the most ethical plan in terms of preferences?*
- *How can this representation separate operational and ethical requirements and reason with both of them through preferences?*
- *Which well-known ethical theories and concepts can our framework model?*

Let us note that this chapter will be based on and extend some of my previous work [Jedwabny et al., 2021a].

This chapter is structured as follows. Section 4.1 describes how our framework detects and models the ethical elements of actions in the context of classical planning. Section 4.2 shows how to assign, characterize and compare plans on ethical features. Section 4.3 shows how our framework can model different well-known ethical theories and concepts. Then, Section 4.4 compares our formalization with existing research. And finally, Section 4.5 concludes and discusses future work.

4.1 Representing ethical features

This first section will focus on describing certain scenarios in which ethical reasoning seems to be particularly relevant and defining how to represent the ethical values conveyed by actions in a planning model. The action model we will utilize here will be the one introduced in Chapter 2, which corresponds to a simplified version of PDDL.

As we discussed in Chapter 3, there are several concepts a framework for ethical reasoning should consider. Particularly, planning frameworks allow for reasoning several steps in advance before executing an action plan, which makes it an ideal setting for analyzing the inter-dependencies between harms, dangers and other ethical notions as one action succeeds another. However, in this section, we will first focus on ethical values for single actions and then extend them to plans in order to represent more complex ethical concepts in the next section.

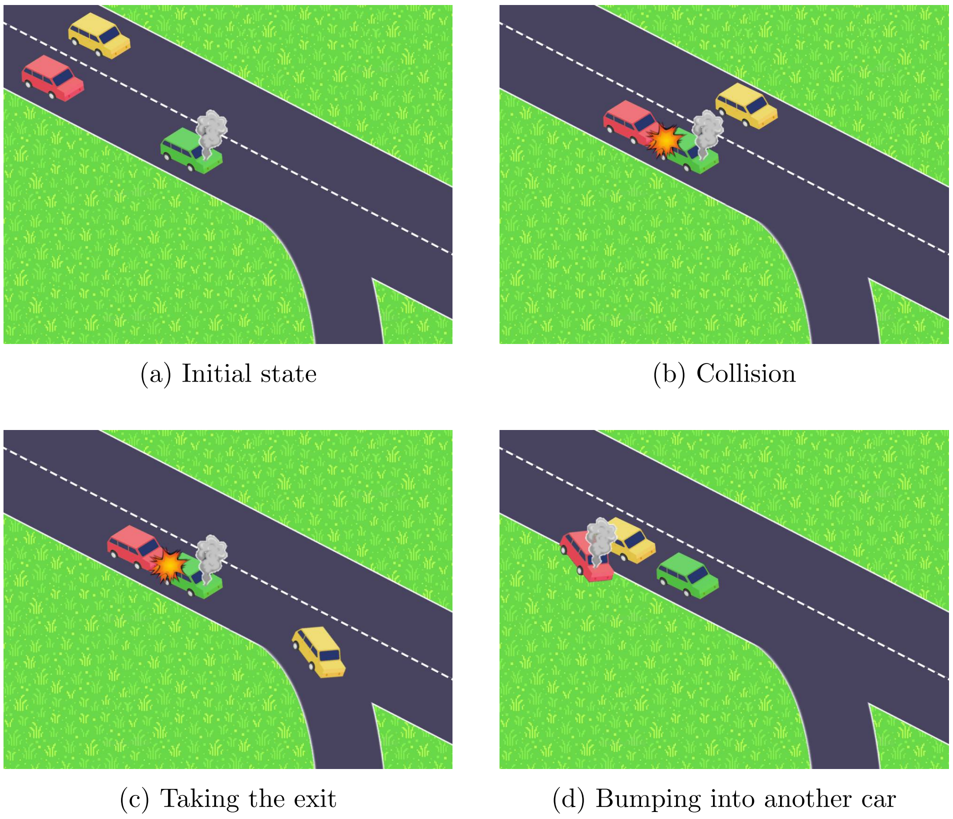
We will use the following example to discuss these notions.

Example 4.1 (Autonomous driver)**.** An autonomous vehicle is tasked to get its passengers from its initial location as depicted in Figure 4.2a to an exit located in the rightmost lane up ahead. There are two lanes and two other cars on the road. The agent's car (illustrated in yellow) is located in the leftmost lane, whereas the two other cars (illustrated in red and green) are in the rightmost lane. While both the yellow and red cars are driving straight up their lanes next to each other, the green car is stopped and visibly damaged further down the road.

Due to the red car's speed, the agent is certain that if nothing prevents it, the red car will crash with the green one, as in Figure 4.2b, endangering the passengers of both cars. Regardless, the agent could avoid the green car and take the exit, as depicted in Figure 4.2c. Because of its advanced technology, the agent's car is potentially able to bump into the red car and prevent this collision with minimal damage to the agent's car and non-lethal damage to the red one, as illustrated in Figure 4.2d.

Although many would agree that bumping into the red car and slightly endangering the agent's car is unacceptable because the agent's passengers were in no danger to start with and intervening would result in the agent incurring additional responsibility, this situation is useful to illustrate many of the ethical concepts which will be of interest to us in this chapter.

Let us describe the previous example using the PDDL model described in Chapter 2. For reference, we will represent it using a tuple $T = \langle \mathcal{D} = (\mathcal{L}, F, O), s_0, g \rangle$ which we use to model classical planning problems.


(a) Initial state

(b) Collision

(c) Taking the exit

(d) Bumping into another car

Figure 4.2: Depiction of Example 4.1.

Example 4.2 (Autonomous driver continued)**.** The underlying logical language $\mathcal{L}$ of this formalization includes the following constants to represent the various entities of the problem:

- $agent, c_1, c_2$: the cars in the problem,
- x_1, x_2: the two lanes, which can be seen as a horizontal 'X' position,
- y_1, y_2, y_3, y_4: an abstraction for the different 'Y' positions, representing how far down the road a car is, and
- $left, straight, right$: the possible directions the agent could take to change lanes.

Furthermore, it includes the following predicate symbols to represent the fluents of the planning problem:

- $equal(C1, C2)$: equality between car constants to help define operators,
- $hasPos(C1, X1, Y1)$: that car $C1$ is located at the position $(X1, Y1)$,

- *hasDir*$(C1, D1)$: that the car $C1$ has direction $D1$,
- *nextX*$(D1, X1, X2)$: that the lane $X2$ is at direction $D1$ from $X1$ (for example, x_2 is at the right of x_1 and x_1 is at the left of x_2),
- *nextY*$(Y1, Y2)$: that the 'Y' position $Y2$ succeeds $Y1$ (we do not use directions as in lanes because we consider cars can only go forward),
- *hasCrashed*$(C1)$ and *hasBumped*$(C1)$: that a car $C1$ has crashed or bumped, respectively, and
- *updated*$()$: that the state of the simulation is updated after executing each step (more on this point shortly).

Then, the fluents F are defined as:

$$\begin{aligned}
F = &\{equal(C1, C2) : C1, C2 \in \{agent, c_1, c_2\}\} \cup \\
&\{dir(D1) : D1 \in \{left, straight, right\}\} \cup \\
&\{hasPos(C1, X1, Y1) : C1 \in \{agent, c_1, c_2\} \\
&\quad \wedge X1 \in \{x_1, x_2\} \wedge Y1 \in \{y_1, y_2, y_3, y_4\}\} \cup \\
&\{hasDir(C1, D1) : C1 \in \{agent, c_1, c_2\} \\
&\quad \wedge D1 \in \{left, straight, right\}\} \cup \\
&\{nextX(D1, X1, X2) : D1 \in \{left, straight, right\} \\
&\quad \wedge X1, X2 \in \{x_1, x_2\}\} \cup \\
&\{nextY(Y1, Y2) : Y1, Y2 \in \{y_1, y_2, y_3, y_4\}\} \cup \\
&\{hasCrashed(C1) : C1 \in \{agent, c_1, c_2\}\} \cup \\
&\{hasBumped(C1) : C1 \in \{agent, c_1, c_2\}\} \cup \\
&\{updated()\}
\end{aligned}$$

We recall that the fluents F represent the properties that a state might have in the planning problem, while the operators O are lifted representations of the actions the agent can execute.

Turning to the operators, we have defined four: o_{dir}, o_{stop}, o_{go} and o_{upd}.

Example 4.3 (Autonomous driver continued)**.** The set of operators O in T is the following:

$$\begin{aligned}
O = \{o_{dir} = &\langle setDir(D1), \\
&\{updated(), dir(D1)\}, \\
&\{\neg hasDir(agent, left), \neg hasDir(agent, straight), \\
&\neg hasDir(agent, right), hasDir(agent, D1)\}\rangle, \\
o_{stop} = &\langle setStop(), \\
&\{updated()\}, \\
&\{\neg hasDir(agent, left), \neg hasDir(agent, straight),
\end{aligned}$$

$$
\begin{aligned}
&\qquad \neg hasDir(agent, right)\}\rangle,\\
o_{go} =&\langle go(),\\
&\qquad \{updated()\}\\
&\qquad \{\neg updated(),\\
&\qquad \forall(C1, D1, Y1, Y2, X1, X2)\\
&\qquad\quad \{\neg hasCrashed(C1), hasPos(C1, X1, Y1), hasDir(C1, D1),\\
&\qquad\quad nextX(D1, X1, X2), nextY(Y1, Y2)\} \Rightarrow\\
&\qquad\quad \{\neg hasPos(C1, X1, Y1), hasPos(C1, X2, Y2)\}\}\rangle\\
o_{upd} =&\langle update(),\\
&\qquad \{\neg updated()\},\\
&\qquad \{updated(),\\
&\qquad \forall(C1, C2, Y1, X1)\\
&\qquad\quad \{\neg equal(C1, C2), \neg equal(C1, agent),\\
&\qquad\quad \neg equal(C2, agent), \neg hasCrashed(C1),\\
&\qquad\quad hasPos(C1, X1, Y1), hasPos(C2, X1, Y1)\} \Rightarrow\\
&\qquad\quad \{hasCrashed(C1), hasCrashed(C2)\},\\
&\qquad \forall(C1, Y1, X1)\\
&\qquad\quad \{\neg equal(C1, agent), hasPos(agent, X1, Y1),\\
&\qquad\quad hasPos(C1, X1, Y1)\} \Rightarrow\\
&\qquad\quad \{hasBumped(agent), hasBumped(C1), \neg hasDir(C1, left),\\
&\qquad\quad \neg hasDir(C1, straight), \neg hasDir(C1, right)\}\}\rangle\}
\end{aligned}
$$

The idea behind this abstracted model is that at each step, the agent:

1. Decides on what to perform, either to move forward in some direction $D1 \in \{left, straight, right\}$ with $setDir(D1)$, or to stop through $setStop()$,
2. Runs the simulation forward a single step in time with $go()$, and
3. Updates the state of the cars in case they crashed or bumped with another, through $update()$.

It is important to notice that in cases where both a fluent and its negation form part of the effects of an action, adding the fluent to the next state takes precedence, as described in Chapter 2. I.e: if an action is executed and has as effects $A()$ and $\neg A()$, then $A()$ is added in the next state.

This simulation is an abstraction that the agent can use before performing any action to analyze their future consequences and takes into account many simplifying assumptions. The model takes into account numerous simplifying assumptions, which we took deliberately to later focus on the ethical aspects of the domain, namely:

- Each 'Y' position y_i has the same distance to the next 'Y' position y_{i+1},

- At each time step, each car that is not stopped moves a single 'Y' position forward,
- Moving one lane left (right) from the leftmost (rightmost) lane is not possible and therefore that car remains in the same lane as before,
- Whenever two cars other than the agent reach the same position they crash and stop moving,
- When the agent and another car reach the same position, they bump into each other, making the other car stop but not crash, i.e: the damage is inferior, and
- Only the agent's car can change its direction on its own.

The fact that other cars cannot change their direction or stop can be interpreted as the agent having only a current representation of the real world and being forced to take immediate action with the information available. In the case it receives updates, one could consider that the agent would re-plan from the updated situation. A point can also be made for incorporating updates inside of the planning model, for instance, with exogenous actions, but it will be left for future work.

Finally, the initial state and goal of the problem are defined as follows.

Example 4.4 (Autonomous driver continued)**.** The initial state s_0 is:

$$\begin{aligned}s_0 = \{&updated(), equal(agent, agent), equal(c_1, c_1), equal(c_2, c_2),\\ &dir(left), dir(straight), dir(right),\\ &hasPos(agent, x_1, y_1), hasPos(c_1, x_2, y_1), hasPos(c_2, x_2, y_3),\\ &hasDir(agent, straight), hasDir(c_1, straight),\\ &nextX(straight, x_1, x_1), nextX(straight, x_2, x_2),\\ &nextX(right, x_1, x_2), nextX(right, x_2, x_2),\\ &nextX(left, x_1, x_1), nextX(left, x_2, x_1),\\ &nextY(y_1, y_2), nextY(y_2, y_3), nextY(y_3, y_4), nextY(y_4, y_4)\}\end{aligned}$$

And the goal:

$$\begin{aligned}g = \{&hasPos(agent, x_2, y_4),\\ &\neg hasCrashed(agent), updated()\}\end{aligned}$$

The initial state s_0 includes fluents that do not change when performing actions, such as $equals(C1, C2)$, $dir(D1)$, $nextX(D1, X1, X2)$ and $nextY(Y1, Y2)$, which capture the static properties of the problem, and other that do change, such as $hasPos(C1, X1, Y1)$ and $hasDir(C1, D1)$, which represent the current position and direction of each car, which can be changed through $setDir(D1)$, $setStop()$ and $go()$.

So for instance, as depicted in Figure 4.2a, in the initial state, the agent is in the position (x_1, y_1) and is going $straight$, c_1 is in the position (x_2, y_1)

and is also going *straight*, and c_2 is in the position (x_2, y_3) and has no direction since it is damaged.

We specified as the goal that the agent should be in the position (x_2, y_4), which is where the exit is, that the agent should not have crashed and that the final state should be updated (to check the crashes).

Let us now consider the following plans which represent the behaviors shown in Figure 4.2.

Example 4.5 (Autonomous driver continued)**.** Regarding the possible plans the agent may produce, we can model those shown in Figure 4.2 as:

- $\pi_1 = [go(), update(), go(), update(), setDir(right), go(), update()]$, and
- $\pi_2 = [setDir(right), go(), update(), setDir(left), go(), update(),$
  $setDir(right), go(), update()]$.

While π_1 denotes the plan resulting from evading the damaged green car as in Figure 4.2b and c, π_2 does so for the scenario in which the agent's car bumps into the red one before the red car crashes into the green one as in 4.2d. It is straightforward enough to check the states resulting from executing the actions of the plans from the initial state. In order to define them, we use the shorthand notation:

$$\begin{aligned} S = \{&equal(agent, agent), equal(c_1, c_1), equal(c_2, c_2),\\ &dir(left), dir(straight), dir(right),\\ &nextX(straight, x_1, x_1), nextX(straight, x_2, x_2),\\ &nextX(right, x_1, x_2), nextX(right, x_2, x_2),\\ &nextX(left, x_1, x_1), nextX(left, x_2, x_1),\\ &nextY(y_1, y_2), nextY(y_2, y_3), nextY(y_3, y_4), nextY(y_4, y_4)\}\end{aligned}$$

The set S contains all the static information of the problem, that as we explained before, will not change from executing actions.

As such, the resulting states of each plan are:

- $$\begin{aligned} Succ(\pi_1, s_0) = S \ \cup \ \{&updated(), hasDir(agent, right),\\ &hasCrashed(c_1), hasCrashed(c_2)\\ &hasPos(agent, x_2, y_4), hasPos(c_1, x_2, y_3),\\ &hasPos(c_2, x_2, y_3)\}\end{aligned}$$
- $$\begin{aligned} Succ(\pi_2, s_0) = S \ \cup \ \{&updated(), hasDir(agent, right),\\ &hasBumped(c_1), hasPos(agent, x_2, y_4),\\ &hasPos(c_1, x_2, y_3), hasPos(c_2, x_2, y_3)\}\end{aligned}$$

In case the reader is interested in running this example with actual PDDL code, we refer to the Appendix Section B.1 for more details.

In order to make ethical reasoning possible in the context of such scenarios, we will introduce a formalization that can capture its ethical nuances.

As mentioned earlier, we will focus on providing an extra reasoning layer for ethical reasoning separate, in principle, from the operational constraints of a problem.

We will call these concepts that characterize the ethical nuances of actions, *ethical features*. These features will be treated in an abstract and generic manner, depending on the problem at hand, meaning that we will not try to categorize all the possible types of ethical features a problem could have, but leave them as part of the formalization of the problem at hand.

The model that will be presented in what follows is separated into two parts, which capture (i) the *ethical features* considered in classical planning problems, and (ii) a *qualitative model* to represent their relative importance, and determine what sequence of actions is best, combining the intuitions prescribed by well-known ethical theories.

For the following, we assume $T = \langle \mathcal{D} = (\mathcal{L}, F, O), s_0, g \rangle$ to be a classical planning problem as defined in Chapter 2.

Definition 4.6 (Ethical feature)**.** An **ethical feature** $e \in E$ is an atom $p_e(t_1, \ldots, t_k)$, where p_e is a predicate and $t_1, \ldots, t_k$ are constants or variables from $\mathcal{L}$. If every t_i is a constant, we say that e is grounded. E is the predefined set of possible ethical features.

Similar to fluents, ethical features are defined as atoms from the underlying logical language $\mathcal{L}$ of the planning problem T.

In order to compare plans on ethical terms, we will define a construct that assigns ethical features to plans when certain conditions are met, which we will call *ethical rule*. In this thesis, we will consider two ways of defining these conditions:

- Via fluents: assign ethical features when the agent arrives at a state s which satisfies some conditions, e.g: arriving at a state in which a car has crashed, and

- Via operators: assign ethical features when a certain operator o is executed in a state s which satisfies some conditions, e.g: moving a car to a specific position when another car is going to the same place.

At a first glance, it can be unclear why both types of conditions are necessary, however, we argue both can be useful depending on the characterization of ethics that the system designer might want to implement. Defining ethical rules through fluents is what first comes to mind when defining ethical features that depend on the consequences of actions, as prescribed by consequentialist theories. This characterization of ethics would require all the ethical features to depend on the fluents and conversely, that the states contain all the necessary information to judge the actions ethically. Alternatively, using operators to define ethical rules can be useful to introduce ethical rules that cannot be captured without adding fluents to the problem, as we will see shortly (see Example 4.9 on page 75). We want to avoid adding fluents to define ethical rules, as this would go against our treatment which separates the operational from the ethical aspects of the planning problem and force the system designer to take into account all possible ethical features in the problem definition, and thus reduce the modularity of our framework.

Definition 4.7 (Ethical rule)**.** An **ethical rule** is a construct of the form:

$$r = \langle Name(r), Pre(r), Act(r), E(r) \rangle$$

Consisting of:

- An atom $Name(r) = p(X_1, \ldots, X_n)$ with predicate p from $\mathcal{L}$ denoting the *name* of the ethical rule, and $X_1, \ldots, X_n$ the set of all the variables that can be used in r, also called the *parameters*,
- A set of literals $Pre(r)$ with variables contained in $\{X_1, \ldots, X_n\}$, called the *preconditions*, which define the conditions in which the ethical rule can be activated,
- $Act(r)$ called the activation condition, is either:
  - $opName(X_1, \ldots, X_i)$, where $o \in O$ is an operator with name $opName$ and parameters $(X_1, \ldots, X_i)$ with $i \leq n$, denoting the action that activates the ethical rule, or
  - $null$, the special constant symbol that indicates that the ethical rule is defined via fluents.
- A set of literals $E(r)$ of ethical features in E with variables contained in $\{X_1, \ldots, X_n\}$, which denote the ethical features that should be added or removed as a result of activating r.

As mentioned, an ethical rule defines the conditions under which it is necessary to assign an ethical feature to a plan. These conditions are represented by the precondition $Pre(r)$ and the activation condition $Act(r)$. The intuition behind them is that when a plan goes from a state s_i satisfying $Pre(r)$ to another state by executing an operator specified by $Act(r)$, the ethical features in $E(a)$ are added or removed as specified. This would amount to the case mentioned before in which ethical rules are defined via operators. Instead, in the special case $Act(r) = null$, the features are added or removed when any state of the plan satisfies $Pre(a)$, which amounts to defining ethical rules via fluents.

As a last detail, the parameters $(X_1, \ldots, X_n)$ of the rule allow using these variables both to (i) identify the states in which the ethical rules should be activated, and (ii) link them to the parameters of the action that activates it $opName(X_1, \ldots, X_i)$. This means that in the case the rule is defined via operator, we force the i first parameters in the ethical rule name to match the full list of parameters in the operator. We restrict the variables in this way, with no loss of generality, to make sure that no variable appears twice in an operator, e.g: $opName(X, X)$, and that the variables in the rule match exactly the parameters of the operator so that later Proposition 5.7 on page 100 is easier to define. Technically, getting rid of these restrictions would require extending our formalism with the equality symbol, or using an extra fluent $equals(X, Y)$, to check if some parameter variables are identical.

Having defined this construct, now we can define the semantics. The motivation behind ethical rules is to assign ethical features to plans.

Given a plan $\pi = [a_0, a_1, \ldots, a_n]$ for the problem T that passes through states $s_0, s_1, \ldots, s_{n+1}$ where $s_{i+1} = Succ(a_i, s_i)$, and a set of ethical rules R:

Definition 4.8 (Ethical features assignment)**.** The set of ethical features assigned to π with respect to the planning domain T and the ethical rules R is denoted $E^R(\pi)$, or more concisely E_π, and defined as:

$$\begin{aligned}
E^R([]) = \{&e \in E : r \in R, e \in E(r)\theta, \\
&Name(r)\theta \text{ is a grounding, } s_0 \models Pre(r)\theta \text{ and} \\
&Act(r) = null\} \\
E^R([a_0, \ldots, a_i]) = &\big(E^R([a_0, \ldots, a_{i-1}]) - \\
&\{e \in E : r \in R, \neg e \in E(r)\theta, \\
&Name(r)\theta \text{ is a grounding, } s_i \models Pre(r)\theta \text{ and} \\
&(Act(r)\theta = a_i, \text{ or } Act(r) = null)\}\big) \cup \\
&\{e \in E : r \in R, e \in E(r)\theta, \\
&Name(r)\theta \text{ is a grounding, } s_i \models Pre(r)\theta \text{ and} \\
&(Act(r)\theta = a_i, \text{ or } Act(r) = null)\}
\end{aligned}$$

In other words, whenever a state satisfies the grounding of the preconditions of some ethical rule and the activation conditions are met, the corresponding ethical features are added. We assume that in case an ethical feature is both added and removed after one action a_i by one or multiple ethical rules, the ethical feature should be added to E_π for simplicity, i.e: adding takes precedence over removing.

Let us exemplify how these rules work.

Example 4.9 (Autonomous driver continued)**.** Given the planning problem T from before, we can define a set of ethical rules R, composed of the following rules:

$$\begin{aligned}
r_1 = \langle &crashRule(C1), \\
&\{hasCrashed(C1)\}, \\
&null, \\
&\{danger(C1, high)\}\rangle \\
r_2 = \langle &bumpRule(C1), \\
&\{hasBumped(C1)\},
\end{aligned}$$

$$\begin{aligned}
&\quad null,\\
&\quad \{danger(C1, low)\}\rangle\\
r_3 = \langle &responsibleCrashRule(),\\
&\quad \{hasCrashed(agent)\},\\
&\quad null,\\
&\quad \{responsibleAgent()\}\rangle\\
r_4 = \langle &responsibleBumpRule(),\\
&\quad \{hasBumped(agent)\},\\
&\quad null,\\
&\quad \{responsibleAgent()\}\rangle
\end{aligned}$$

In this scenario, we have that π_1 activates r_1 and π_2 activates r_2:

$$\begin{aligned}
E_{\pi_1} &= \{danger(c_1, high), danger(c_2, high)\}\\
E_{\pi_2} &= \{danger(c_1, low), danger(agent, low),\\
&\quad responsibleAgent()\}
\end{aligned}$$

Using this representation, we can model the main two requirements mentioned before, (i) $danger(C1, G1)$ the amount of gravity $G1$ of the danger when a car $C1$ bumps (low) or crashes ($high$), and (ii) $responsibleAgent()$ whenever the agent is responsible for causing a crash or bump. We assumed before that only the agent can produce bumps and so we assign the responsibility of it only to the agent in those cases.

Furthermore, these ethical rules are defined via fluents because they utilize the activation condition $null$ and depend on the fluents '$hasCrashed(C)$' and '$hasBumped(C)$'.

Other ethical rules, however, might be more suitable to define via operators. For instance, if in the Example 4.1 a car would bump into some rails at the edge of the road when going left (right) from the leftmost (rightmost) lane but not change lanes or get damaged as a result, we could assign an ethical feature via the o_{go} operator, without adding extra fluents or effects to actions. The following ethical rules would suffice:

$$\begin{aligned}
r_5 = \langle &railLeftRule(C1, X1, Y1),\\
&\quad \{position(C1, X1, Y1), direction(C1, left)\\
&\quad nextX(left, X1, X1)\},\\
&\quad go(),\\
&\quad \{damageRail(C1)\}\rangle\\
r_6 = \langle &railRightRule(C1, X1, Y1),\\
&\quad \{position(C1, X1, Y1), direction(C1, right)\\
&\quad nextX(right, X1, X1)\},\\
&\quad go(),
\end{aligned}$$

$\{damageRail(C1)\}\rangle$

The rule r_5 (r_6) specifies that whenever a car $C1$ at position $(X1, Y1)$ goes left (right) from the leftmost (rightmost) lane, the ethical feature $damageRail(C1)$ is added to the plan, after executing the action $go()$. The fluent $nextX(left, X1, X1)$ ($nextX(right, X1, X1)$) ensures that $X1$ is indeed the leftmost (rightmost) lane.

One of the most essential ethical features of Example 4.1 is the danger of the consequences of actions and their gravities. From a purely consequentialist point of view (see Chapter 3), bumping into the red car carries less danger for everyone than letting it crash against the green car at full speed, and so if we ignore everything else, there is an argument for bumping into that car. Moreover, from the point of view of society, or at least the passengers using this kind of car, it might be highly desirable that they are not treated the same way as the passengers of other cars. After all, no person would utilize an automated car that could spontaneously bump or crash into other cars to prevent worse outcomes. That is, ethical features might need to take into account their recipient in some application domains, for example, if the stakeholders deem the objective of ethical reasoning to be making the system more reliable to the user.

As part of the input, a system designer should specify a set of ethical rules that characterize when an action in a particular context entails a feature that should be judged on ethical terms and its relative level of importance. Then to reason with these features, the planner must compare the ethical features induced by different actions and use them to compare the possible plans an autonomous agent might take. By doing this, we can separate the (model-based) action selection process from the ethical reasoning of the agent.

4.2 Representing ethical planning problems

In this section, we will turn our attention to the challenge of providing a model that allows comparing plans according to their ethical features using preferences.

An important requirement we demand of our model is that it must allow for qualitative preferences. It has been emphasized [Brundage, 2014] that AI systems in which ethics is useful, can take dangerous decisions in situations of extreme trade-offs. This could be a problem if the preference model was strictly quantitative. For instance, we want to be able to model that a set of ethical rules R takes precedence over an arbitrarily large set of rules R' whenever R presents a critical rule $r \in R$ that precedes features in R'.

Here, we will base our preference representation framework on what is known as ranked knowledge bases [Feldmann et al., 2006] to represent qualitative preferences amongst sets of alternatives. In contrast to the original work, we will not define the preferences for arbitrary formulae, but rather for ethical features. We chose and adapted this model as it combines naturally with our ethical rules and is concise and straightforward to elicit from external sources because as we shall see, only requires assigning a relative numeric value to the importance of each feature separately.

Definition 4.10 (Ethical ranked base)**.** Given a set of ethical features E, an ethical ranked base (ERB) is a function:

$$b(e) = \langle Type(e), Rank(e) \rangle$$

That maps a ground ethical feature $e \in E$ to a pair consisting of a symbol $Type(e) \in \{+, -\}$ representing the *type* i.e. whether the ethical feature is ethically right or wrong, and a non-negative integer $Rank(e) \in \mathbb{N}_0$, which denotes the *rank* of the ethical feature i.e. its level of importance. In the case $Type(e) = +$ we will call e a *positive* ethical feature, otherwise we will call it *negative*. We assume that b is a total function and in cases where its value is not specified for a ground feature e, that the rank is zero, i.e: $Rank(e) = 0$, meaning that the feature has no ethical relevance and the type is positive, but can be ignored.

The motivation behind $Type(e)$ is to make it possible to define not only what behaviors should be avoided, or undesirable, but also what is ethically desirable. In the case that an ethical feature is not assigned to a plan, our stance will be that it will not affect the ethical desirability of the plan. That is, if a positive or negative ethical feature is not assigned to a plan, it will not make the plan more or less ethically preferred than another by itself.

In turn, the rank of the ethical feature defines how important it is. Intuitively, if a plan π_1 is assigned a positive ethical feature e of rank j and π_2 has no features of rank j, then the plan π_1 is preferred if both plans have the same features for every higher rank $i > j$. Conversely, if e is negative and both plans have the same features for every higher rank $i > j$, then π_2 is preferred. More generally, we extend this notion to sets of ethical features as follows:

Definition 4.11 (Ethical preference)**.** Let $A, B \subseteq E$ be two sets of ground ethical features included in E and b an ethical ranked base over E.

Then A is at least as preferred as B, denoted $A \succeq_b B$ if and only if:

$$\begin{aligned}
&\forall i \in \mathbb{N}, \text{ it holds that } b_i^+(A) = b_i^+(B) \text{ and } b_i^-(A) = b_i^-(B), \text{ or} \\
&\exists i \in \mathbb{N}, \text{ such that } \big(b_i^+(B) \subset b_i^+(A) \wedge b_i^-(A) \subseteq b_i^-(B), \text{ or} \\
&\qquad b_i^+(B) \subseteq b_i^+(A) \wedge b_i^-(A) \subset b_i^-(B)\big), \text{ and} \\
&\qquad \forall j > i : b_j^+(A) = b_j^+(B) \text{ and } b_j^-(A) = b_j^-(B).
\end{aligned}$$

Where given $i \in \mathbb{N}$ and a set of ethical features $C \subset E$:

$$b_i^+(C) = \{e \in C : \ Type(e) = +, \ Rank(e) = i\}$$
$$b_i^-(C) = \{e \in C : \ Type(e) = -, \ Rank(e) = i\}$$

We denote $>_b$ and $=_b$ as usual: $A >_b B$ if and only if $A \geq_b B$ and $B \not\geq_b A$; $A =_b B$ if and only if $A \geq_b B$ and $B \geq_b A$.

The expressions $b_i^+(C)$ and $b_i^-(C)$ are used as a shorthand to express the positive and negative ethical features, respectively, of a certain rank i. Then the preference relation between sets of ethical features is defined as a lexicographical order with the caveat of separating the positive from the negative ones. In other words, all ethical features of higher ranks being equal, a set A is preferred to B whenever there is a rank j such that the positive ethical features of that rank, $b_i^+(A)$ include those of $b_i^+(B)$, and conversely $b_i^-(A)$ is a subset of $b_i^+(B)$.

Having defined how to compare sets of ethical features, now we will define an extension of classical planning problems to take into account ethical rules and ethical ranked bases as follows:

Definition 4.12 (Ethical planning problem)**.** An **ethical planning problem** is a tuple $T = \langle \mathcal{D} = (\mathcal{L}, F, O), s_0, g, \mathcal{E} = (E, R, b)\rangle$ where:

- $T' = \langle \mathcal{D}, s_0, g\rangle$ is a classical planning problem.
- E is a set of ethical features,
- R is a set of ethical rules over E, and
- b is an ethical ranked base over E.

Furthermore, the semantics of states and actions is the same as in classical planning problems:

- The states $s \in S$ of T are all the ground fluents combinations of F, and thus coincide with those of T',
- The actions $a \in A(T)$ are all the groundings of operators $o \in O$ with constants from $\mathcal{L}$, and thus coincide with $A(T')$,
- Given a state s and an action a, the successor state $Succ(a, s)$ is the same in T and T', and
- A list of actions π is a plan for T whenever $Succ(\pi, s_0) \models g$.

This definition captures all the elements we need to compare plans on ethical terms using our framework. The ethical elements modeled by $\mathcal{E} = (E, R, b)$ are separated from the rest of the planning definition, making the framework modular, as it simply extends classical planning problems.

By definition, a plan for the classical planning problem $\langle \mathcal{D}, s_0, g\rangle$ is also a plan for the ethically extended version.

Proposition 4.13. Let $T = \langle \mathcal{D}, s_0, g, \mathcal{E} \rangle$ be an ethical planning problem and $T' = \langle \mathcal{D}, s_0, g \rangle$ a classical planning problem, then π is a plan for T if and only if it is a plan for T'.

This is straightforward to prove.

Proof 4.14. It follows from Definition 4.12, as the states, actions and successor function definitions are the same as in classical planning problems.

Moreover, our definition of $\mathcal{E} = (E, R, b)$ allows comparing plans on ethical terms as follows.

Definition 4.15 (Ethically preferred plan)**.** Let $T = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b) \rangle$ be an ethical planning problem and π_1, π_2 plans for T, then π_1 is at least as **ethically preferred** as π_2, denoted $\pi_1 \succeq_b \pi_2$ if and only if $E_{\pi_1} \succeq_b E_{\pi_2}$.

We denote $\succ_b$ and $=_b$ as usual: $\pi_1 \succ_b \pi_2$ if and only if $\pi_1 \succeq_b \pi_2$ and $\pi_2 \not\succeq_b \pi_1$; $\pi_1 =_b \pi_2$ if and only if $\pi_1 \succeq_b \pi_2$ and $\pi_2 \succeq_b \pi_1$.

Moreover, we say that plan π_1 is **ethically optimal** if and only if for every plan π_2 for T, it holds that $\pi_1 \succeq_b \pi_2$.

In other words, we use Definition 4.11, which we defined for sets of ethical features, to compare plans through the ethical features that are assigned to them through the ethical rules in R.

Let us exemplify how this would work.

Example 4.16 (Autonomous driver continued)**.** Consider the classical planning task $T = \langle \mathcal{D}, s_0, g \rangle$ and:

- The set of ethical features:

$$\begin{aligned} E = &\{danger(C, G) : C \in \{agent, c_1, c_2\},\ G \in \{low, high\}\} \cup \\ &\{damageRail(C) : C \in \{agent, c_1, c_2\}\} \end{aligned}$$

- The set of ethical rules $R = \{r_1, r_2, r_3, r_4, r_5, r_6\}$ as defined before in Example 4.9.

We can define the following ethical ranked base b:

$$\begin{aligned} &b(damageRail(C)) = \langle -, 1 \rangle \ \forall C \in \{agent, c_1, c_2\} \\ &b(danger(C, low)) = \langle -, 2 \rangle \ \forall C \in \{agent, c_1, c_2\} \\ &b(danger(C, high)) = \langle -, 3 \rangle \ \forall C \in \{c_1, c_2\} \\ &b(danger(agent, high)) = \langle -, 4 \rangle \\ &b(responsibleAgent()) = \langle -, 4 \rangle \end{aligned}$$

All the ethical features in these examples are negative because they all refer to the impacts that the cars may cause. The ranks imply that damaging the rail is not as negative as causing danger to a car, that causing *high* danger is worse than causing *low* danger, and finally, that if the agent is in *high* danger, or is responsible for a collision, either by bumping or crashing into another car, it is the worst-case scenario. The idea behind this rank is to model the intuition mentioned before, that if possible, the agent should refrain from causing itself any damage, no matter if other cars crash between themselves. Furthermore, the agent being in *high* danger should be also avoided at all costs. These intuitions, of course, are debatable, however, it serves to exemplify what our framework can do.

Thus, we have all the building blocks to define the ethical planning problem $T' = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b) \rangle$.

In this scenario, as mentioned before, we have that:

$$\begin{aligned} E_{\pi_1} &= \{danger(c_1, high), danger(c_2, high)\} \\ E_{\pi_2} &= \{danger(c_1, low), danger(agent, low), \\ &\qquad responsibleAgent()\} \end{aligned}$$

Then, $\pi_1 \succ_b \pi_2$ since $R_4^-(\pi_1) = \{responsibleAgent()\}$ and $R_4^-(\pi_2) = \emptyset$, thus $R_4^-(\pi_2) \subsetneq R_4^-(\pi_1)$ and there no other features of higher rank. In other words, π_1 is ethically preferred to π_2 because it was assigned the ethical feature $responsibleAgent()$, which is a negative feature and also the one with the highest rank.

With these last definitions, we can compare plans on ethical terms. However, it was left unspecified (i) how a planner might find ethically optimal plans, and (ii) what would happen in the case two plans have uncomparable ethical features, in the sense that for the highest rank, both plans have disjoint ethical features, and possibly a different number of them. We will address these questions in the following chapter. For now, let us see how this framework can represent different well-known ethical theories.

4.3 Modeling ethical theories

In what follows, we will exemplify how our ethical planning framework can model certain ethical theories and notions described in Chapter 3 and use them to compare plans ethically. For each of these theories, we will present a template in the form of abstract ethical features, rules and ranked bases that can be used to model the ethical elements of a problem according to that theory.

One of the benefits of our model is that all of these theories can be present in an ethical planning problem and used at the same time. Also, by using different ranks, the theories can either operate on several levels of priority (e.g: making deontological preferences stronger than the consequentialist ones) or be used to compare plans at the same levels.

4.3.1 Consequentialist ethics

As we explained in Section 3.3.1, consequentialism is the normative ethical theory that compares the rightness of actions only with respect to their consequences. The way in which the benefits and drawbacks of consequences are measured varies from one interpretation to another. Here, the perspective from which consequences are compared will be the welfare of a certain group of individuals, also called utilitarianism [Mill and Bentham, 1987]. It states that an agent should always choose the alternative that minimizes pain and maximizes the happiness of this select group. Those individuals whose welfare defines the rightness of actions, e.g: the agent and/or the other involved entities, also depend on the interpretation.

In the context of AI planning, the agent plans ahead of time a sequence of actions to perform and can therefore predict both the immediate and future consequences of its actions. As such, past research on machine ethics [Ganascia, 2015, Lindner et al., 2019, Winfield et al., 2014] seems to agree that consequentialism should be implemented by comparing the final state reached by plans on ethical terms.

Using the framework presented in the last section, one can model the ethical elements of a planning domain via ethical features, ranks and rules. Modeling a consequentialist theory amounts to defining all the ethical features that affect the welfare of the relevant individuals. For instance, let us consider an ethical planning problem $T = \langle \mathcal{D} = (\mathcal{L}, F, O), s_0, g, \mathcal{E} = (E, R, b)\rangle$. Assuming that a set of fluent literals

$$\left\{P_1(\hat{X_1}), \ldots, P_n(\hat{X_n}), \neg Q_1(\hat{X_{n+1}}), \ldots, \neg Q_m(\hat{X_{n+m}})\right\}$$

With P_i, Q_j predicates and $\hat{X_i}, \hat{X_j}$ lists of variables, defines a state of affairs that affects the welfare of an individual a for a reason rs (which we will also use as the identifier of the ethical rule) with gravity $g \in \mathbb{N}$, and the welfare of all individuals is equally as important, we can model this with an ethical feature $e \in E$, rule $r \in R$ and assign $b(e)$ as:

$$\begin{aligned}
e =& affects(a, rs, g)\\
r =& \langle ruleC\text{-}a\text{-}rs(\hat{X}),\\
& \quad \left\{P_1(\hat{X_1}), \ldots, P_n(\hat{X_n}), \neg Q_1(\hat{X_{n+1}}), \ldots, \neg Q_m(\hat{X_{n+m}})\right\},\\
& \quad Act(r),\\
& \quad \{affects(a, rs, g)\}\rangle\\
b(e) =& \begin{cases} \langle +, g\rangle \text{ if } rs \text{ is good for } a, \text{ or}\\ \langle -, g\rangle \text{ otherwise}\end{cases}
\end{aligned}$$

with a, rs, g constants of the language $\mathcal{L}$, $\hat{X}$ a list of variables that contains all of those in each $\hat{X_i}$ and $Act(r)$ either $null$ if the set of fluent literals in the rule precondition defines by itself that the welfare of a was affected, or an operator name $opName(X_1, \ldots, X_k)$ of T if executing an action is also needed to do so.

Notice that the individual a can be lifted to a variable A of $\mathcal{L}$, for instance, if some of the fluents P_i or Q_j refer to A in their arguments.

In Example 4.9 we can see a few ethical rules of this kind, such as r_1, which we could rewrite using the template above as:

$$\begin{aligned}
e =& affects(agent, crash, 4)\\
r =& \langle ruleC\text{-}agent\text{-}crash(),\\
& \quad \{hasCrashed(agent)\},\\
& \quad null,\\
& \quad \{affects(agent, crash, 4)\}\rangle\\
b(e) =& \langle -, 4\rangle
\end{aligned}$$

As we saw earlier, this describes the fact that if in the final state we have that the agent crashed, the plan will be morally wrong.

We are taking several assumptions in this model. First, the fluents of the state (and optionally an action) should be sufficient to capture when the agent's behavior affects individuals. Although our framework separates the ethical comparison of plans from the operational model, the agent cannot reason about states it cannot perceive. We also assume that the gravity of the features can be captured with a numerical value $g \in \mathbb{N}$, which sounds reasonable in the context of utilitarianism.

A problem one may face when using the current characterization of ethical preferences is that the model cannot decide between two plans π_1 and π_2 when the two plans have disjoint positive or negative ethical features of a certain rank and the same features of higher rank. By definition 4.11, if π_1 has, say, ten positive features of a certain rank and π_2 only one but they are disjoint, the plans will be incomparable. However, some characterizations of utilitarianism compare plans by the sum of the utilities assigned to their ethical features, which in this case is their rank. We can fix this issue either by the concept of *linearization* we will introduce in Section 5.1, of the next chapter.

4.3.2 Deontological ethics

As opposed to consequentialist views, deontological ethics asserts that an action should be judged on whether it complies with a set of duties and obligations, rather than based on its consequences.

This theory has been applied to automatic systems [Hashmi et al., 2014, Berreby et al., 2017, Lindner et al., 2019] by constructing and enforcing restrictions that characterize what is permitted and what is forbidden. While [Berreby et al., 2017] defines the deontological principle with a set of logical rules that determine when an action is permitted or forbidden, [Lindner et al., 2019] defines a plan as deontologically permissible if the total utility of each action (based on its consequences) is positive.

From a practical perspective, we can think of deontological ethics as a theory that distinguishes right from wrong actions depending on whether that action complies with the duties all members of a society should follow. In our context, the model we defined cannot specify by itself this set of duties and reason about whether the action complies with them or not. Rather, we take the stance that the system designer should define a priori what actions performed at which states are forbidden from a deontological point of view, and encode them as ethical features, ranks and rules. Of course, these could also be generated by an external procedure that reasons in advance to produce these ethical constructs.

Similar to the last section, we can define deontological ethical features/rules as:

$$\begin{aligned}
e =& forbidden(rs, g)\\
r =& \langle \mathit{ruleD\text{-}rs}(\hat{X}),\\
& \{P_1(\hat{X_1}), \ldots, P_n(\hat{X_n}), \neg Q_1(\hat{X_{n+1}}), \ldots, \neg Q_m(\hat{X_{n+m}})\},\\
& Act(r),\\
& \{forbidden(rs, g)\}\rangle\\
b(e) =& \langle -, g\rangle
\end{aligned}$$

where rs, g are constants, P_i, Q_j predicates and $\hat{X}, \hat{X_i}$ are lists of variables exactly like in the last section and $Act(r) = opName(X_1, \ldots, X_k)$ an operator name and parameters of T.

We interpret that the ethical rule must be defined via operators and not fluents because deontological ethics defines the rightness of actions by the action itself and not its consequences, i.e: the state reached by performing it.

Because the rule checks for actions that should be forbidden, the type of the feature is negative. Moreover, if the planning problem designer does not desire any deontologically 'forbidden' feature to be in a plan, the gravity $g \in \mathbb{N}$ should be set to the maximum possible value.

For instance, in Example 4.9 we have ethical rules to check whether a car crashes. From a deontological point of view, we can make a rule that states that going to the same location as another car endangers the passengers of both cars, which would be wrong:

$$\begin{aligned}
e =& forbidden(endanger, 5)\\
r =& \langle \mathit{ruleD\text{-}endanger}(C1, X1, X2, X3, Y1, D1, D2),\\
& \{\neg \mathit{equal}(C1, agent), \mathit{hasPos}(agent, X1, Y1), \mathit{hasPos}(C1, X2, Y1),\\
& \mathit{hasDir}(agent, D1), \mathit{hasDir}(C1, D2), \mathit{nextX}(D1, X1, X3),\\
& \mathit{nextX}(D2, X2, X3)\},\\
& go(),\\
& \{forbidden(endanger, 5)\}\rangle\\
b(e) =& \langle -, 5\rangle
\end{aligned}$$

Notice that the ethical rule also characterizes when *agent* bumps into $C1$ according to Example 4.9, although in this case, the semantics of the ethical rule describes a different aspect that is not operational but rather ethical (r would still hold even if the fluent *hasBumped*($C1$) was unavailable).

A considerable amount of research (see Chapter 3) considers that breaking deontological restrictions is unacceptable, in the sense that the gravity should be infinite and that a plan that breaks such a principle should not be considered a solution, however as we explained before, we take the stance that all plans should be considered and that many times not doing anything could be unethical by itself.

4.3.3 Virtue ethics

Along with the aforementioned theories, virtue ethics is one of the main three normative ethical branches. According to virtue ethics, an agent is deemed ethical when the actions it performs exhibit the characteristics of a virtuous being. What constitutes virtue can vary between interpretations and cultures, but some classical examples are fairness, honesty and compassion. In contrast to the other theories, virtue ethics relies on the moral values of an agent. Of the three main ethical theories, virtue ethics is the one that has been the least present in AI research (see Chapter 3).

Using our framework, we will interpret virtue ethics as an ethical theory that assigns virtues in the form of ethical features, as opposed to deontological ethics, which assigns reasons to forbid an action, as we saw in the previous section. Indeed, we can define the virtuous aspects of action as follows:

$$\begin{aligned}
e =& \,virtuous(rs, g)\\
r =& \langle ruleV\text{-}rs(\hat{X}),\\
& \{P_1(\hat{X_1}), \ldots, P_n(\hat{X_n}), \neg Q_1(\hat{X_{n+1}}), \ldots, \neg Q_m(\hat{X_{n+m}})\},\\
& Act(r),\\
& \{virtuous(rs, g)\}\rangle\\
b(e) =& \langle +, g\rangle
\end{aligned}$$

where rs, g are constants, $g \in \mathbb{N}$, P_i, Q_j predicates and $\hat{X}, \hat{X_i}$ are lists of variables exactly like in the last section and $Act(r) = opName(X_1, \ldots, X_k)$ an operator name and parameters of T.

Also like the previous section, we interpret that the ethical rule must be defined via operators and not fluents because the virtues stem from the action and not its consequences. However, because virtues are positive traits, the ethical feature is positive as well.

Following Example 4.9, we can define the virtue of *generosity* when the *agent* bumps into a car $C1$ in order to prevent it from crashing more dangerously into another car $C2$:

$$\begin{aligned}
e =& \,virtuous(preventCrash, 5)\\
r =& \langle ruleV\text{-}preventCrash(C1, C2, X1, X2, X3, Y1, Y2, D1, D2),
\end{aligned}$$

$$\begin{aligned}
&\{\neg equal(C1, agent), \neg equal(C2, agent), \neg equal(C1, C2),\\
&\ hasPos(agent, X1, Y1), hasPos(C1, X2, Y1), hasPos(C2, X3, Y1),\\
&\ hasDir(agent, D1), hasDir(C1, D2), hasDir(C2, D3),\\
&\ nextX(D1, X1, X4), nextX(D2, X2, X4), nextX(D3, X3, X4), nextY(Y1, Y2)\\
&\ \neg hasCrashed(agent), \neg hasCrashed(C1), \neg hasCrashed(C2)\},\\
&\ go(),\\
&\ \{virtuous(preventCrash, 5)\}\rangle\\
b(e) =&\langle +, 5\rangle
\end{aligned}$$

The ethical rule r reads as follows: if the car $C1$ is in position $(X2, Y1)$, $C2$ in position $(X3, Y1)$, they are both going to a location $(X4, Y2)$ in the next step and the *agent* can go to that the same location in the next turn, then it will prevent the crash by bumping into both cars and stopping them, which would be virtuous.

4.3.4 Prima facie duties

Also a relatively well-known ethical theory, Ross's prima facie duties [Ross and Ross, 2002] judge the rightness of actions according to various predefined moral duties. They present a pluralist view that can be linked both to deontological and virtue theory in that (i) the duties they strive to adhere to judge the intent of actions and not their consequences, and (ii) the duties themselves are inspired by the obligations people have to society and the premise of showing good intent. Recalling Chapter 3, the duties suggested in his work are fidelity, reparation, gratitude, non-maleficence, beneficence, self-improvement and justice.

The main way in which this theory differs from the previous ones is that various classes of duties are taken into account at the same time and not every duty is as important as the others in every situation, for example, non-maleficence is almost always to be prioritized, as harming other people should be avoided in all decisions. Also, if an action supports a more important duty or the same duties that another action plus some, then that action is deemed right.

This pluralist view is suitable for our framework, which models various ethical features and gives them relative ranks of priority. Indeed, if ethical features are interpreted to be prima facie duties and we use the ethical ranks to model their relative importance, our model will fit properly with the perspective of this theory.

Following the templates of the last sections, given a prima facie duty p (constant of $\mathcal{L}$), we can define ethical constructs to represent prima facie duties as:

$$\begin{aligned}
e =& primaFD(p, rs, g)\\
r =& \langle rulePFD\text{-}p\text{-}rs(\hat{X}),\\
&\ \big\{P_1(\hat{X_1}), \ldots, P_n(\hat{X_n}), \neg Q_1(\hat{X_{n+1}}), \ldots, \neg Q_m(\hat{X_{n+m}})\big\},
\end{aligned}$$

$$Act(r),$$
$$\{primaFD(p, rs, g)\}\rangle$$
$$b(e) = \begin{cases} \langle +, g\rangle \text{ if } rs \text{ is good for } p, \text{ or} \\ \langle -, g\rangle \text{ otherwise} \end{cases}$$

where rs, g are constants, $g \in \mathbb{N}$, P_i, Q_j predicates and $\hat{X}, \hat{X}_i$ are lists of variables exactly like in the last section and $Act(r) = opName(X_1, \ldots, X_k)$ an operator name and parameters of T.

Other literature on prima facie duties in the context of machine ethics can be found in [Anderson et al., 2005a, Anderson et al., 2005b, Anderson and Anderson, 2018], however, their work is focused on the inference of preferences between these duties in the context of decision-making, instead of modeling ethical theories for automated planning, as we do in this section.

4.3.5 Doctrine of double effect

Another ethical theory that has been modeled by various top-down machine ethics approaches is the doctrine of double effect (DDE). Recalling Chapter 3, the doctrine characterizes an action as permissible when:

1. the action in itself is good or indifferent,
2. the agent only intends the good effects and if there are any bad ones, s/he would rather be in a situation where the action would not be needed,
3. the bad effects do not cause the good effects by themselves, rather they are both produced by the action independently, and
4. there is a proportionally important or desirable reason to permit the bad effects in light of the good ones.

Some literature [Govindarajulu and Bringsjord, 2017] has tackled the task of modeling all these elements through formal logic.

Here, we will choose to focus on (3) by determining when a certain bad ethical feature causes a good one, and leave the rest of the conditions for future work. However, these concepts require certain adaptations if we hope to model them using our framework. Namely, because good and bad effects will be captured by ethical features, representing the fact that one ethical feature can cause another will require, as we will see shortly, allowing ethical rules inside of the precondition of ethical rules. While Definition 4.7, which defines ethical rules r, only needs to allow ethical features in $pre(r)$, Definition 4.8, which defines the ethical features $E^R(\pi)$ assigned to a plan π, should be slightly modified to take into account the ethical features assigned to a plan after every action, as follows:

$$E^R([]) = \{e \in E : r \in R, e \in E(r)\theta,$$

$$\begin{aligned}
&\qquad Name(r)\theta \text{ is a grounding,}\\
&\qquad s_0 \models Pre(r)\theta, \text{ and}\\
&\qquad Act(r) = null\}\\
E^R([a_0, \ldots, a_i]) = &\ \big(E^R([a_0, \ldots, a_{i-1}]) -\\
&\{e \in E : r \in R, \neg e \in E(r)\theta,\\
&\quad Name(r)\theta \text{ is a grounding, ,}\\
&\quad s_i \models (Pre(r)\theta \cap F),\\
&\quad E^R([a_0, \ldots, a_{i-1}]) \models (Pre(r)\theta \cap E) \text{ and}\\
&\quad (Act(r)\theta = a_i, \text{ or } Act(r) = null)\}\big) \cup\\
&\{e \in E : r \in R, e \in E(r)\theta,\\
&\quad Name(r)\theta \text{ is a grounding, ,}\\
&\quad s_i \models (Pre(r)\theta \cap F),\\
&\quad E^R([a_0, \ldots, a_{i-1}]) \models (Pre(r)\theta \cap E) \text{ and}\\
&\quad (Act(r)\theta = a_i, \text{ or } Act(r) = null)\}
\end{aligned}$$

Then, we can model the DDE using our framework using the following ethical feature, rule and rank:

$$\begin{aligned}
e =& forbiddenDDE(I1, I2, g)\\
r =& \langle ruleDDE(I1, I2),\\
& \{eDDE(bad, I1), eDDE(good, I2), causes(I1, I2)\},\\
& null,\\
& \{forbiddenDDE(I1, I2, g)\}\rangle\\
b(e) =& \langle -, g\rangle
\end{aligned}$$

with $g \in \mathbb{N}$ a constant, $I1, I2$ variables, and $eDDE(bad, I1), eDDE(good, I2)$ and $causes(I1, I2)$ ethical features. We will make the simplifying assumption that $eDDE(bad, I1)$ and $eDDE(good, I2)$ are generic ethical features, where $eDDE(bad, I1)$ is activated by other already specified ethical rules, that $eDDE(bad, I1)$ has a negative type and $eDDE(good, I2)$ positive, and that they only count with one argument $I1$ and $I2$, respectively, which is a constant that denotes their identifier.

Briefly, r is a lifted ethical rule that assigns the $forbiddenDDE(I1, I2, g)$ ethical feature to a plan when a negative feature with identifier $I1$ causes a positive feature with identifier $I2$. Notice that using this template we may define several identifiers for positive and negative ethical features to use with the rule above and it would also be straightforward to extend these ethical features with more parameters. Of course, the problem here is providing ethical rules that activate the $causes(I1, I2)$ ethical feature.

Capturing causality between the effects of actions in an AI planning model is a problem by itself that has been addressed in different ways. Most notably, [Berreby et al., 2018] develops various definitions of causality for planning using a modified event calculus and [Lindner et al., 2019] presents another interesting approach for STRIPS planning with exogenous actions.

However, the mentioned articles do not make the distinction we made between ethical features and fluents. Typically, they model causality taking into account the effects and preconditions of actions. For instance, some accounts of causality express that an action a causes not only its effects $e_1, \ldots, e_n$, which may be ethical or not, but also another future effect e_{n+1} if a subsequent action a' has e_{n+1} as an effect and some e_i with $i \in [1, n]$ is a precondition of a'.

In our framework, we need to model causality differently. First, the feature $causes(I1, I2)$ can only be activated through ethical rules. And second, we can not use the inter-dependence between action effects and preconditions to model causality, but rather we can say that $eDDE(S1, I1)$ causes $eDDE(S2, I2)$ whenever the second ethical feature is activated through a rule that has the first as a precondition.

Essentially, given two ground ethical features $eDDE(s_1, i_1)$ and $eDDE(s_2, i_2)$, where $s_1, s_2 \in \{good, bad\}$ and i_1, i_2 two constants denoting identifiers, we can define a special ethical rule to capture $causes(i_1, i_2)$ as:

$$\begin{aligned} r' = \langle & ruleCauses(), \\ & \{eDDE(s_1, i_1)\} \cup A, \\ & null, \\ & \{eDDE(s_2, i_2), causes(i_1, i_2)\}\rangle \end{aligned}$$

where $A = \left\{P_1(\hat{X_1}), \ldots, P_n(\hat{X_n}), \neg Q_1(\hat{X_{n+1}}), \ldots, \neg Q_m(\hat{X_{n+m}})\right\}$ is a set of literals that represents any extra conditions needed to activate r'.

We also assume the $causes(I1, I2)$ predicate to be a transitive relation, which we can specify with a single ethical rule:

$$\begin{aligned} r'' = \langle & ruleCausesTrans(I1, I2, I3), \\ & \{causes(I1, I2), causes(I2, I3)\}, \\ & null, \\ & \{causes(I1, I3)\}\rangle \end{aligned}$$

Notice that the ethical rule definitions for $forbiddenDDE(I1, I2, g)$ and $causes(I1, I2)$ demand ethical features to be present in their preconditions. And as we explained earlier, allowing ethical features in the precondition of ethical rules requires extending Definition 4.7 and 4.8 to take into account ethical feature literals in the precondition of ethical rules, along with fluent literals.

4.3.6 Do-no-harm principle

Some research [Lindner et al., 2019] has also put forward the interest in modeling ethics as the prevention of harm. Harm prevention is a consequentialist principle that usually involves checking that the agent does not produce any harm through any single action, instead of the whole plan. As such, preventing harm requires:

1. Checking that the harmful consequences were produced by the agent and were not present in the initial state, and

2. Preventing the harmful consequences of any action even if this harm is erased through a future action.

Supposing we have a negative ethical feature $e = affects(a, rs, g)$, rule r and rank $b(e)$ as defined with the consequentialist ethics template of Section 4.3.1, we can define a new ethical feature $e_h = producesHarm(a, rs, g_h)$ which models the fact that the agent itself has produced the harm $affects(a, rs, g)$ through one of its actions and that it was not present in the initial state.

More precisely, (1) can be modeled by checking whether e was present in a past state. Then, we can add ethical constructs to perceive when this harm is being produced by the agent as follows:

$$\begin{aligned} e_h =& producesHarm(a, rs, g_h) \\ r_h =& \langle ruleDNH\text{-}a\text{-}rs(\hat{X}), \\ & Pre(r) \cup \{\neg affects(a, rs, g)\}, \\ & Act(r), \\ & \{producesHarm(a, rs, g)\}\rangle \\ b(e_h) =& \langle -, g_h \rangle \end{aligned}$$

with $a, rs, g, r, \hat{X}$ as defined in Section 4.3.1 and $g_h \in \mathbb{N}$ a number that defines the gravity of producing harm rs to a.

Notice that, just like the last section, this principle requires extending the precondition of ethical rules to consider ethical features along with fluents.

Moreover, (2) can be ensured simply by not specifying any ethical rule that deactivates $producesHarm(a, rs, g)$, in other words, for no ethical rule r' it can hold that $\neg producesHarm(a, rs, g) \in E(r')$.

4.4 Related work

As explained in the previous section, our work is a direct contribution to top-down machine ethics in the context of AI planning. In particular, our framework seeks to model explicit ethical agents (see Chapter 3), i.e: those that can reason ethically by following principles that are encoded in their representation.

In the context of this subject, the separation between the operational and ethical aspects of problems can be related to past literature on ethical governors [Arkin et al., 2009] for BDI agents. In this article, the authors embedded an extra layer within the agent and rejects action plans that violate ethical constraints. Similarly, [Dennis et al., 2016] proposed a framework for BDI agents that instead selects action plans which minimize ethical constraint violations. Then, [Cointe et al., 2016] introduced a framework for ethical planning with preferences, also based on BDI agents, that presents a construct called moral rule and valuations, which coincide with our ethical rules and features, respectively. On the other hand, their framework deals with preferences between ethical principles and not ethical features and therefore does not permit combining elements of multiple ethical theories at the same level as we do here, e.g: when the ethical features of different ethical theories are assigned the same rank. Furthermore, BDI approaches compare precomputed plans obtained via external planning modules on ethical terms. However, our framework behaves differently than all BDI approaches because our ethical preferences, as we shall see in the next chapter, can be taken into account by a planner and its heuristic strategies and not after the planning phase.

On a different venue, [Berreby et al., 2015, Berreby et al., 2017] present an event-calculus-based framework that implements several ethical theories through answer set programming. In their work, they present several definitions adapting well-known ethical principles for automated reasoning that can determine when plans are right or wrong from the point of view of different ethical theories. Similarly, [Ganascia, 2007, Ganascia, 2015] define several ethical principles adapted to automated reasoning for answer set programming and in [Berreby et al., 2018], the authors extend that work by presenting several different interpretations of causality. And then, in [Bourgne et al., 2021] their ideas are expanded by allowing concurrency and multiple agents. All of this research is closer to our framework presented in this chapter in that: (i) they consider the ethical aspects of a problem at planning time, unlike BDI agents, and (ii) the semantics of states and actions is handled by what they call an event motor, which can be kept separated from the ethical reasoning. As we have seen in Section 4.3, we can adapt some ethical theories similarly using our framework, but our treatment of ethics takes a step aside by considering preferences. That is, our framework can help decide between multiple plans on ethical terms even when all of them are unethical according to all ethical theories, which as we have discussed, can be considered an advantage in certain cases.

With the purpose of defining normative ethical restrictions, [Govindarajulu and Bringsjord, 2017, Hashmi et al., 2014, Marín and Sartor, 1999] developed different models for ethically-aware planning based on deontic logic and the event calculus. Also related, in [Panagiotidi and Vázquez-Salceda, 2011], restrictions were characterized using context-dependent norms and applied to STRIPS-based planning domains. Our work contrasts with their work in that we consider multiple ethical theories and model ethical theories through preferences.

Moreover, in [Lindner et al., 2017, Lindner et al., 2019], the authors present a characterization of various ethical decision mechanisms such as utilitarianism, different versions of the do-no-harm principle and the doctrine of double effect in the context of the SAS+ planning model. Our work is indeed similar to theirs in that it permits modeling various ethical theories in the same framework. However, just like the previously discussed corpus of research, our framework differs from all of them in that it handles ethics via preferences.

As we have shown, one of the advantages of our framework is that it went further than classifying actions as right or wrong by using preferences. Qualitative preferences between the ethical features are represented using ranks as in [Feldmann et al., 2006]. In this aspect, our work adapts their definition of ranked bases to consider two types of features, positive and negative. Also, in our framework, these preferences only deal with ethical features which are kept separated from the other fluents and are only activated through ethical rules.

Finally, past research on causality [Berreby et al., 2018, Govindarajulu and Bringsjord, 2017] did not make the distinction we made between ethical features and fluents and did not deal with preferences, thus making our approach to causality and the principle of double effect, discussed in Section 4.3 different from other literature.

4.5 Discussion

In this chapter, we have developed a domain-independent planning framework through an extension of PDDL that allows agents to analyze plans on ethical terms and compare them through preferences. We chose to focus on a high level of abstraction with the classical planning language PDDL so that we could concentrate on the ethical reasoning aspects.

As mentioned at the start of this chapter, we separated the ethical and operational aspects of a planning problem by introducing ethical features. From a practical standpoint, ethical features are similar to fluents in the sense that they characterize properties evolving with the execution of actions, however, we chose to keep them separate for different reasons. To begin with, state fluents are used to characterize the state of the world and define action preconditions, effects and goals. None of this holds for the ethical features. The world state is not affected by the ethical features of a domain, rather ethical features refer to the properties of a plan, and it is not our intention to make the definition of actions and goals depend on ethical features. Furthermore, by keeping the fluents and ethical features separate, we ensure modularity between the problem definition and its ethical side, meaning that (i) the ethical definitions do not change the problem description in the case one would want to add our ethical extension to an existing problem, and (ii) the ethical features, rules and ranks of one problem can be used in another by adapting the ethical rules to the fluents and actions of the second problem, thus making it possible to reuse already defined ethical preferences and principles, which can be hard and expensive to elicit. We also introduced ethical rules, a construct that assigns ethical features to plans when certain conditions are met. The concept of ethical rules is the method we provided to assign ethical features to plans without mixing them with other fluents. By doing this, we ensured that ethical features can depend on fluents but not the other way around, thus keeping the ethical and operational aspects of the problems separated.

In Chapter 3, we explained that past research on top-down machine ethics (i.e: providing a model of ethics to the machine instead of inferring it) has focused on producing mechanisms to characterize which plans are ethically acceptable or not. By introducing preferences, our framework went further than classifying actions as right or wrong. We advocated that in the context of autonomous systems, the machine always has to be able to compute a plan. If no unethical plan is considered, then the situation could be at an impasse. Our argument against not computing any plan that is not perfectly ethical is threefold: (i) doing nothing can sometimes also be unethical, (ii) certain domains, such as autonomous driving, might arrive at a scenario that always demands the machine to do something, e.g: another car about to crash with the agent, and (iii) the task of the planning phase is to determine the most ethical way of achieving the specified goal and the agent could always change the goal and replan if the computed plan does not satisfy certain constraints, that is, the agent can always analyze the plan in a posterior phase handled by another reasoning mechanism and reparametrize the problem.

In our framework, preferences are modeled through ethical ranked bases, which determine the level of importance of ethical features and whether they are positive or negative ethically, based on [Feldmann et al., 2006]. This type of preference may seem restrictive compared to other languages used for ethical planning like [Gerevini and Long, 2005], but as we have shown can be general if the right modelization of ethical features and rules are encoded. Furthermore, it (i) combined naturally with our ethical rules, (ii) allows us to treat multiple ethical theories concurrently by using different ethical ranks, or mixing them together using intersecting ranks, and (iii) is concise and straightforward to elicit from external sources because it only requires assigning a relative numeric value to the importance of each feature separately.

A framework that separates ethical aspects into different priorities is imperative in ethical domains. For instance, consider a problem in which an agent is assigned a unit of utility for executing a trivial action, such as giving away ice creams, while on the other hand is assigned a thousand units or any other fixed utility for not killing a person. Summing up simple utilities would effectively compensate for killing a person if the agent gives away enough ice creams. In contrast, the rank-based approach prohibits such kind of behavior by assigning different ranks to the ethical features.

We have shown how the intuitions of many ethical theories may be modeled through our framework. Interpreting ethical theories in the context of automated reasoning is not simple in most cases. Ethics by itself is a deeply human subject and machines will frequently not have access to all the information people consider when judging choices ethically. However, we show that many of its concepts may be represented or adapted.

In the context of this chapter, the use of ethical features, rules and preferences changed slightly the interpretation of the ethical theories compared to past research on machine ethics. However, we have shown through some examples that our approach is effectively capable of realizing many useful ethical theory intuitions.

Moreover, past research on causality did not make the distinction we made between ethical features and fluents and did not deal with preferences. Instead, they seek to model which actions are permissible or not.

Indeed, both in the matter of ethical reasoning and causality, we have shown that our approach provides a step forward in the field by presenting an alternative view on top-down machine ethics through preferences.

We insist that one of the main advantages of our framework is that all of these theories can be present in an ethical planning problem at the same time and used concurrently due to the ethical ranked bases. If the system designer wants to compare the results of different theories separately, the ranks of features can be adapted to prioritize a specific one (e.g: making deontological preferences stronger than the consequentialist ones). Otherwise, because ethical theories get reduced to features by using the templates we specified, they can be combined seamlessly by using the same ranks through the preference relation we defined between plans.

Future work It would be interesting to apply the ideas of this chapter to other planning models that allow for multiple agents, exogenous actions, uncertainty in the form of belief states, or probabilistic planning.

Furthermore, a simple improvement we can make to the assignment of ethical features to plans is considering multisets. Indeed, Definition 4.8 does not account for repeated ethical features, but extending this to multisets would be straightforward. This could be helpful to model cases in which activating a positive or negative ethical feature multiple times makes a difference in the ethical consideration of a plan.

Regarding the preference model, one possible way of extending this work would be by providing a more general language to express them, as described in [Brewka, 2004].

Finally, ethical rules could also be extended to be activated when certain conditions hold for every, or a certain number of states in a row, similar to PDDL3 preferences [Gerevini et al., 2009].

5 Planning with ethical preferences

In this chapter, we will show how ethically optimal plans may be obtained by transforming ethical planning problems, as defined in the last chapter, into classical planning problems with utilities. Moreover, by employing this translation procedure, ethical planning problems will be able to be solved using existing state-of-the-art planners. Like in the previous chapter, the work presented here can be considered an extension of my work on [Jedwabny et al., 2021a].

In order to do this, we will first introduce a *valuation* function, based on [Feldmann et al., 2006] but adapted to take into account our positive and negative ethical features, that assigns a numerical value to plans according to their ethical features.

Furthermore, by utilizing a valuation function we can solve the incomparability problem mentioned in the last chapter. It will often be the case that two plans are incomparable, whenever their ethical features at their highest level are disjoint. Assigning a numerical value to plans solves this problem because by comparing plans in terms of their assigned value, no two plans will be incomparable.

We will also discuss two implementations of the mentioned translation procedures we have made publicly available and test the effectiveness of our approach in terms of computational efficiency using various state-of-the-art planners.

As such, our research questions for this chapter are the following:

Research Questions in this Chapter

- *How can our ethical planning problems, as described in the last chapter, be solved with existing AI planning technology?*
- *How effective is this approach in practice in terms of computational efficiency?*

This chapter is structured as follows. Section 5.1 demonstrates how our ethical planning problems can be translated into utilities by using soft goals. Section 5.2 provides an overview of the two implementations we developed for the ideas discussed in the previous section. In Section 5.3, we describe the various experiments we formulated to test the effectiveness of our approach in terms of computational efficiency. Then, Section 5.4 discusses the related work. And finally, Section 5.5 concludes and describes the different ways in which the ideas in this chapter can be extended.

5.1 Translating ethical preferences to utilities

In this section, we will show how ethical planning problems can be reduced to classical planning problems with a transformation routine similar to [Feldmann et al., 2006].

As a first step, we will show how the ethical preference relation $\succeq_b$ introduced in the last chapter can be modified so that every plan is comparable. More formally, it is simple to verify that the $\succeq_b$ preference relation is reflexive (i.e: $\pi =_b \pi$) and transitive (i.e: $\pi_1 \succeq_b \pi_2 \ \wedge \ \pi_2 \succeq_b \pi_3 \implies \pi_1 \succeq_b \pi_3$), and thus a preorder. However, it may often be the case that certain plans satisfy disjoint elements at a level $i \in \mathbb{N}$, so we cannot say this order is total. Such a preorder can be made total with the use of linearizations [Feldmann et al., 2006]:

Definition 5.1 (Linearization)**.** A linearization of $\succeq$, a preorder over set A, is a total preorder $\succeq^{lin}$ over A that extends $\succeq$, i.e: $\forall a, b \in A,\ a \succeq b \implies a \succeq^{lin} b$, and $\forall a, b \in A$ either $a \succeq^{lin} b$, or $b \succeq^{lin} a$.

This extension is useful as there is always at least one linearization for any preorder and it can be constructed by using a valuation function. In the case of our ethical preferences $\succeq_b$, we define this function as follows:

Definition 5.2 (Ethical valuation)**.** Let $T = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b)\rangle$ be an ethical planning problem, π a plan for T, and $maxval_0 = 0$, then $\forall i \in \mathbb{N}$:

1. $val_i = maxval_{i-1} + 1$
2. $maxval_i = |\{b_i^+(E) \ \cup \ b_i^-(E)\}| \times val_i + maxval_{i-1}$
3. $val(\pi) = val(E_\pi)$, where given $E' \subseteq E$:

$$val(E') = \sum_{i \in \mathbb{N}} |b_i^+(E') \ \cup \ (b_i^-(E) - b_i^-(E'))| \times val_i$$

Intuitively, val_i is a number that expresses how much valuation to add to an ethical plan, when it is assigned a positive ethical feature of rank i, or when it is not assigned a negative one. So for instance, if a plan π_1 has one more positive ethical feature than another plan π_2, which is of rank i, and both plans have the same negative features, then $val(\pi_1) = val(\pi_2) + val_i$.

According to (2), $maxval_i$ represents the maximum value a plan can achieve up to rank i. In other words, it is the valuation a plan π can have if it is assigned every positive and no negative ethical feature for every rank $j \in [1, i]$. In conjunction with (1), which defines $val_i = maxval_{i-1} + 1$, we can see that when a plan π is assigned a positive or is not assigned a negative ethical feature of rank i, it gives π more valuation than what it can achieve with all the ethical features of lower ranks $j \in [1, i-1]$. This is important because it will guarantee that the semantics we want to achieve is satisfied, in the sense that higher-level ethical features take precedence over all the other lower-ranked features.

Then (3) defines the valuation of a plan π as the sum of the amount of ith ranked positive features $e \in E_\pi$ and negative features $e \notin E_\pi$, multiplied by val_i. Taking this into consideration, we can see that by using such a valuation function, an automated planner will choose plans that satisfy the most amount of positive and least amount of negative ethical features of the highest ranks.

Proposition 5.3. Consider $T = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b) \rangle$ an ethical planning problem and π, π' plans for T. Let the preference relation between two plans $\succeq_b^{lin}$ be defined as $\pi \succeq_b^{lin} \pi'$ if and only if $val(\pi) \geq val(\pi')$, then it is indeed a linearization of $\succeq_b$.

Proof 5.4. Suppose that $\pi =_b \pi'$, then trivially $val(\pi) = val(\pi')$ because $b_i^+(E_\pi) = b_i^+(E_{\pi'})$ and $b_i^-(E_\pi) = b_i^-(E_{\pi'})$ for every $i \in \mathbb{N}$.

In the case $\pi \succ_b \pi'$, then according to Definition 4.11 and 4.15:

1. $\exists i \in \mathbb{N}$, such that $(b_i^+(E_{\pi_2}) \subset b_i^+(E_{\pi_1}) \wedge b_i^-(E_{\pi_1}) \subseteq b_i^-(E_{\pi_2})$, or $b_i^+(E_{\pi_2}) \subseteq b_i^+(E_{\pi_1}) \wedge b_i^-(E_{\pi_1}) \subset b_i^-(E_{\pi_2}))$, and
2. $\forall j > i : b_j^+(E_{\pi_1}) = b_j^+(E_{\pi_2})$ and $b_j^-(E_{\pi_1}) = b_j^-(E_{\pi_2})$.

Therefore, because of (1), π has more positive or less negative ethical features of rank i:

$$|b_i^+(E_\pi) \cup (b_i^-(E) - b_i^-(E_\pi))| > |b_i^+(E_{\pi'}) \cup (b_i^-(E) - b_i^-(E_{\pi'}))|$$

Then, by construction of val_i, other ethical features of lower ranks do not matter. In other words, when π is assigned a positive or is not assigned a negative ethical feature of rank i, it gives π more valuation than what it can achieve with all the ethical features of lower ranks $j \in [1, i-1]$, thus:

$$|b_i^+(E_\pi) \cup (b_i^-(E) - b_i^-(E_\pi))| \times val_i > \sum_{k=1}^{i} |b_k^+(E_\pi) \cup (b_k^-(E) - b_k^-(E_\pi))| \times val_k$$

And because of (2), i.e: all higher ranks being equal:

$$\sum_{j>i} |b_j^+(E_\pi) \cup (b_j^-(E) - b_j^-(E_\pi))| \times val_j = \sum_{j>i} |b_j^+(E_{\pi'}) \cup (b_j^-(E) - b_j^-(E_{\pi'}))| \times val_j$$

Thus, $\pi \geq_b \pi' \implies val(\pi) \geq val(\pi')$ and by definition, $val(\pi) \geq val(\pi')$ if and only if $\pi \geq_b^{lin} \pi'$.

Finally, because val assigns an integer to every plan, $\geq_b^{lin}$ is total.

Let us see $\geq_b^{lin}$ in action in the following example.

Example 5.5 (Autonomous driver continued)**.** Following our running example, there are three ethical features of rank 1 and 2, and two ethical features of rank 4, respectively. Thus it holds that $val_1 = 1, val_2 = 4, val_3 = 16$ and $val_4 = 48$, then:

$$\begin{aligned} val(\pi_1) =& |\{damageRail(C) : C \in \{agent, c_1, c_2\}\}| \times val_1 + \\ & |\{danger(C, low) : C \in \{agent, c_1, c_2\}\}| \times val_2 + \\ & |\{danger(agent, high)\}| \times val_4 + \\ & |\{responsibleAgent()\}| \times val_4 = 111 \\ val(\pi_2) =& |\{damageRail(C) : C \in \{agent, c_1, c_2\}\}| \times val_1 + \\ & |\{danger(c_2, low)\}| \times val_2 + \\ & |\{danger(C, high) : C \in \{c_1, c_2\}\}| \times val_3 + \\ & |\{danger(agent, high)\}| \times val_4 = 87 \end{aligned}$$

As expected, due to the fact that the ethical feaatures $responsibleAgent()$ and $danger(agent, high)$ have rank 4, which is the highest, and only π_1 avoids them, $val(\pi_1) > val(\pi_2)$ and thus $\pi_1 >_b^{lin} \pi_2$. Notice that in the sum we include the positive ethical features assigned to each plan and the negative ones that are not assigned to the plan. This means that for each negative ethical feature that is not assigned to the plan, the valuation of that plan will increase. In particular, this example only deals with negative features and because π_1 is not assigned the ethical features $responsibleAgent()$ and $danger(agent, high)$, the valuation $val(\pi_1)$ increases significantly.

In order to find an optimal plan, we will show that any ethical planning problem T can be transformed into an equivalent classical planning problem with utilities $T_{utility}$ by using the valuation function defined before.

In what follows, we make the following assumptions for ethical planning problems:

- The ethical features are *distinct* from the fluents i.e. $\nexists e \in E$ s.t. $e \in F$,
- The fluent $check()$ is fresh, i.e: it is not included in the fluent set of any problem, and

- The operator name *checkOp*() is fresh, i.e: it is not included in the operator set of any problem.

Definition 5.6. Given an ethical planning problem $T = \langle(\mathcal{L}, F, O), s_0, g, (E, R, b)\rangle$, then its *transformed* classical planning problem with utilities is defined as:

$$T_{utility} = \langle(\mathcal{L}, F', O'), s_0, g', c, u\rangle$$

Where:

- $F' = F \cup \{check()\} \cup \{e \in E : e \text{ is a ground ethical feature}\}$,
- $O' = \{o' : o \in O\} \cup \{o_{check}\}$, where:

$$\begin{aligned} o' = \langle & Name(o),\\ & Pre(o) \cup \{check()\},\\ & Eff(o) \cup \{\neg check()\}\\ & \quad \cup \ \{\forall(X_{i+1}, \ldots, X_n)\ Pre(r) \Rightarrow E(r) : \exists r \in R\\ & \quad \text{such that } Act(r) = Name(o), (X_1, \ldots, X_n) \text{ are the}\\ & \quad \text{parameters of } r \text{ and } (X_1, \ldots, X_i) \text{ are those of } o\}\rangle\\ o_{check} = \langle & checkOp(),\\ & \{\neg check()\},\\ & \{check()\} \cup \{\forall(X_1, \ldots, X_n)\ Pre(r) \Rightarrow E(r) : \exists r \in R\\ & \quad \text{such that } Act(r) = null \land\\ & \quad X_1, \ldots, X_n \text{ is the set of variables in } Name(r)\}\rangle \end{aligned}$$

- $g' = g \cup \{check()\}$,
- The action costs are all zero, i.e: $c(a) = 0$ for every $a \in A(T_{utility})$.
- The utility u of a state $s \in S$ is defined as follows:

$$u(s) = \sum_{i \in \mathbb{N}} |b_i^+(E \cap s) \ \cup \ (b_i^-(E) - b_i^-(E \cap s))| \times val_i$$

The fluent set F is extended with the fluent *check*() and an extra fluent for each ground ethical feature. Whenever an ethical rule is activated, the transformation will guarantee that the fluents corresponding to ethical features are added or removed according to the rule, so at the end of the plan execution, the final state will include all the ethical features assigned to the original plan. The fluent *check*() is used to ensure that after each action is executed, the planner will be forced to check if any ethical rules defined via fluents, i.e: with activation condition *null*, have been activated.

Then, each original operator o is extended as well through o', with the precondition *check*() and the effect $\{\forall(X_1, \ldots, X_n)\ Pre(r) \Rightarrow E(r) : \exists r \in R \text{ such that } Act(r) = o\}$. The precondition forces each plan to execute o_{check}

after every action, to ensure that all ethical rules defined via fluents are checked after executing every one of them. The additional effect guarantees that whenever an action is executed and an ethical rule defined via operator r is activated, the ethical features in the rule are added or removed accordingly. In conjunction with the operator o_{check}, which takes care of ethical rules defined via fluents, the transformation guarantees that the final state will include all the ethical features that should be assigned to the original plan π.

The goal is extended to ensure that in the final state, all the ethical features have been checked, otherwise, it could be the case that an ethical rule defined via fluents is activated after the last action, but that the ethical feature fluents are not updated accordingly.

Finally, the utilities are defined simply by summing the valuations according to the formulas in Definition 5.2 of every ethical feature fluents in the final state reached by a plan, which as we will see, coincides with the ethical features assigned to the plan in T.

Notice that although this transformation does indeed add a fluent for each ethical feature, it guarantees that the two classes of fluents (original vs. induced by ethical features) do not interact with one another in a seamless way to the domain designer, ensuring that the modularity between the operational and ethical aspects of the problem is preserved.

Proposition 5.7. Given an ethical planning problem T and its transformation $T_{utility}$, a sequence of actions $\pi = [o_0\theta_0, o_1\theta_1, \dots, o_n\theta_n]$, where each o_i is an operator and θ_i a grounding substitution, is a plan for T if and only if $\pi_{utility}$ is a plan for $T_{utility}$, where:

$$\begin{aligned}\pi_{utility} = [&o_{check}, o'_0\theta_0, o_{check}, o'_1\theta_1, o_{check}, \dots, \\ &o'_n\theta_n, o_{check}]\end{aligned}$$

Furthermore, given two plans π, π' of T, if $u(\pi_{utility}) \geq u(\pi'_{utility})$ with respect to $T_{utility}$, then $\pi \geq_b^{lin} \pi'$ with respect to T.

Let us prove Proposition 5.7.

Proof 5.8. It is straightforward to see that any plan π of T can be transformed into a plan $\pi_{utility}$ for $T_{utility}$ and vice versa. Due to the fact that:

- The fluents of T are included in those of $T_{utility}$, i.e: $F \subset F'$,
- The only change to preconditions is that $Pre(o') = Pre(o) \cup \{check()\}$, which is removed after every operator o' such that $o \in O(T)$, but added after executing o_{check}, which $\pi_{utility}$ does before and after every action of π, and
- The only changes to effects of actions o' with respect to $o \in O$, relate to the fluents in $E \cup \{check()\}$, which are disjoint from F by

assumption, the fluents of the original problem T.

This is why, we can say that for every state s and action $o'\theta$ of $T_{utility}$, where o' is an operator of $T_{utility}$, o is an operator of T, and θ a grounding substitution, it holds that:

$$Succ_{T_{utility}}([o_{check}, o'\theta], s) \cap F = Succ_T([o\theta], s \cap F)$$

In other words, executing the two actions $[o_{check}, o'\theta]$ at state s in $T_{utility}$ is equivalent to executing the action $o\theta$ at state $s \cap F$ in the original problem T if we only consider the original fluents in F, captured by the intersection at the left side of the equation.

Therefore, because the initial state s_0 is the same and the last action o_{check} of $\pi_{utility}$ does not modify any original fluent in F:

$$Succ_{T_{utility}}(\pi_{utility}, s_0) \cap F = Succ_T(\pi, s_0 \cap F)$$

I.e: executing the plan $\pi_{utility}$ at state s_0 in $T_{utility}$ is equivalent to executing π at state s_0 in the original problem T if we only consider the original fluents in F.

Thus, because $g' = g \cup \{check()\}$, we have that $g' \cap F = g$, and because the last action of $\pi_{utility}$ is o_{check}, it adds the fluent $check()$ to $Succ_{T_{utility}}([\pi_{utility}], s_0)$, so then, it holds that:

$$\begin{aligned}
&Succ_{T_{utility}}([\pi_{utility}], s_0) \models g, \text{ and}\\
&Succ_{T_{utility}}([\pi_{utility}], s_0) \models \{check()\}, \text{ thus}\\
&Succ_{T_{utility}}([\pi_{utility}], s_0) \models g'
\end{aligned}$$

So we have proven that $\pi_{utility}$ is a plan for $T_{utility}$ if and only if π is a plan for T.

Now, we need to prove that given two plans π, π' of T, if $u(\pi_{utility}) \geq u(\pi'_{utility})$ with respect to $T_{utility}$, then $\pi \geq_b^{lin} \pi'$ with respect to T.

As a first step, let π be any plan for T and $\pi_{utility}$ its transformation for $T_{utility}$, we have to prove that the valuation function $val(\pi)$ of plan π for T coincides with the utility of the plan $\pi_{utility}$ for $T_{utility}$. In other words, we need to verify that $u(\pi_{utility}) = val(\pi)$, and so:

$$\begin{aligned}
&u(\pi_{utility}) = val(\pi) \xLeftrightarrow{\text{Definition 2.28}}\\
&\Big(u(Succ(\pi_{utility}, s_0)) - \sum_{a \in \pi_{utility}} c(a)\Big) = val(\pi) \xLeftrightarrow{\forall a,\ c(a)=0}\\
&u(Succ(\pi_{utility}, s_0)) = val(\pi) \xLeftrightarrow{\text{Definition 5.2}}\\
&u(Succ(\pi_{utility}, s_0)) = \sum_{i \in \mathbb{N}} |b_i^+(E_\pi) \ \cup \ (b_i^-(E) - b_i^-(E_\pi))| \times val_i
\end{aligned}$$

And using u as Definition 5.6 in the proposition we are proving, let $s^* = Succ(\pi_{utility}, s_0)$ be the final state of the plan $\pi_{utility}$:

$$u(Succ(\pi_{utility}, s_0)) = \sum_{i \in \mathbb{N}} |b_i^+(E_\pi) \ \cup \ (b_i^-(E) - b_i^-(E_\pi))| \times val_i \iff$$

$$\sum_{i \in \mathbb{N}} |b_i^+(E \cap s^*) \ \cup \ (b_i^-(E) - b_i^-(E \cap s^*))| \times val_i =$$

$$\sum_{i \in \mathbb{N}} |b_i^+(E_\pi) \ \cup \ (b_i^-(E) - b_i^-(E_\pi))| \times val_i \iff E \cap s^* = E_\pi$$

In other words, we need to prove that the ethical feature fluents in the final state s^* of the plan $\pi_{utility}$ for $T_{utility}$ are the same as the ethical features assigned to π in the ethical planning problem T. This follows from these facts:

1. The ethical feature fluents in F' are only added or removed from a state by the operator o_{check}:

   $$\{\forall (X_{i+1}, \ldots, X_n) \ Pre(r) \Rightarrow E(r) : \exists r \in R \text{ such that } Act(r) = o, (X_1, \ldots, X_n) \text{ are the parameters of } r \text{ and } (X_1, \ldots, X_i) \text{ are those of } o\}$$

   and the added effects to the original operators:

   $$\{\forall (X_1, \ldots, X_n) \ Pre(r) \Rightarrow E(r) : \exists r \in R \text{ such that } Act(r) = null \ \wedge X_1, \ldots, X_n \text{ is the set of variables in } Name(r)\}$$

   which add and remove the ethical fluents exactly as prescribed in Definition 4.8.

2. For any rule $r \in R$ such that $Act(r) = null$, it holds that r is activated by $\pi = [o_0\theta_0, \ldots, o_n\theta_n]$ in T if and only if there is an intermediate state:

   $$s_i = Succ([o_0\theta_0, \ldots, o_i\theta_i], s_0)$$

   with $i \leq n$, such that $s_i \models Pre(r)\theta_i$. This can only happen when the intermediate state:

   $$s_i' = Succ([o_{check}, o_0'\theta_0, o_{check}, \ldots, o_i'\theta_i, o_{check}], s_0)$$

   of plan $\pi_{utility}$ for $T_{utility}$ satisfies that $s_i' \models Pre(r)\theta_i$, because as we saw before, the original fluents in F are unchanged between the plans π and $\pi_{utility}$ in their respective planning problems. Therefore, the last action o_{check} of $[o_{check}, o'_0\theta_0, o_{check}, \ldots, o'_i\theta_i, o_{check}]$ will add and remove the fluents in $E(r)$ according to the definition of o_{check}.

3. For any rule $r \in R$ such that $Act(r) \neq null$, it holds that r is activated by $\pi = [o_0\theta_0, \ldots, o_n\theta_n]$ in T if and only if there is an intermediate state:

$$s_i = Succ([o_0\theta_0, \ldots, o_i\theta_i], s_0)$$

with $i \leq n$, such that $s_i \models Pre(r)\theta_i$ and $Act(r)\theta_{i+1} = o_{i+1}\theta_{i+1}$. This can only happen when the intermediate state:

$$s'_i = Succ([o_{check}, o'_0\theta_0, o_{check}, \ldots, o_{check}, o'_i\theta_i], s_0)$$

of plan $\pi_{utility}$ for $T_{utility}$ satisfies that $s'_i \models Pre(r)\theta_i$ and $Act(r)\theta_{i+1} = o'_{i+1}\theta_{i+1}$, because as we saw before, the original fluents in F are unchanged between the plans π and $\pi_{utility}$ in their respective planning problems.

Thus, $E \cap s^* = E \cap Succ(\pi_{utility}, s_0) = E_\pi$ and as we saw before this proves that $val(\pi) = u(\pi_{utility})$.

Therefore, given two plans π, π' of T, and $\pi_{utility}, \pi'_{utility}$ the transformed plans for $T_{utility}$, then:

$$\begin{aligned} u(\pi_{utility}) \geq u(\pi'_{utility}) \text{ with respect to } T_{utility} \iff \\ val(\pi) \geq val(\pi') \text{ with respect to } T \iff \\ \pi \geq_b^{lin} \pi' \end{aligned}$$

Furthermore, the utility of a plan $\pi_{utility}$ for $T_{utility}$ can be calculated simply by summing the ranks of the positive ethical features in the last state s^* reached by the plan, and the ranks of the negative ones not included in it.

Proposition 5.9. Let T be an ethical planning problem, π a plan for T, $T_{utility}$ and $\pi_{utility}$ the planning problem and plan obtained with the transformation of Proposition 5.7, and s^* the last state reached by $\pi_{utility}$, then:

$$u(\pi_{utility}) = \sum_{\{e \in E \cap s^* : Type(e) = +\} \,\cup\, \{e \in (E - s^*) : Type(e) = -\}} val_{Rank(e)}$$

Let us prove this proposition.

Proof 5.10. As we saw in Proof 5.8:

$$u(\pi_{utility}) = u(Succ(\pi_{utility}, s_0))$$

$$= \sum_{i \in \mathbb{N}} |b_i^+(E \cap s^*) \ \cup \ (b_i^-(E) - b_i^-(E \cap s^*))| \times val_i$$

Due to the fact that multiplying val_i by the amount of members of a set is the same as summing val_i for each member:

$$u(\pi_{utility}) = \sum_{i \in \mathbb{N}} \sum_{e \in b_i^+(E \cap s^*) \ \cup \ (b_i^-(E) - b_i^-(E \cap s^*))} val_i$$

Then, because val_i is equivalent to the valuation of ethical features of rank i according to Definition 4.11:

$$u(\pi_{utility}) = \sum_{i \in \mathbb{N}} \sum_{e \in b_i^+(E \cap s^*) \ \cup \ (b_i^-(E) - b_i^-(E \cap s^*))} val_{Rank(e)}$$

And finally, because b_i^+ and b_i^- are respectively the positive and negative ethical features at each level i, according to Definition 4.11:

$$u(\pi_{utility}) = \sum_{\{e \in E \cap s^* : Type(e) = +\} \ \cup \ \{e \in (E - s^*) : Type(e) = -\}} val_{Rank(e)}$$

The approach we took in this section was to transform ethical planning problems into classical ones using utilities. This is practical because, as we saw in Chapter 2, there are numerous implemented planners for classical planning problems with utilities.

In practical terms, our transformation:

- Takes each ethical feature $e \in E$ and adds it as a fluent to the utility planning domain,
- Adds an operator o_{check} to check which ethical rules defined via fluents are activated,
- Adds a conditional effect to every action for each ethical rule defined via operators to check if the rule was activated, and
- Defines a utility function based on the rank of the ethical feature fluents included in a state.

This will, of course, have an impact on the computational costs of finding plans for the resulting planning problem. Finding the optimal plan in classical planning problems with utilities can be intractable in the general case (see Chapter 2), so it will not always be feasible to find the optimal one for the transformation we have shown. However, it is possible to do so in many cases, as current state-of-the-art planners are capable of finding solutions even for very hard problems. In the following sections, we will discuss how we implemented this transformation for the PDDL language and provide some experimental results for our approach.

5.2 Implementation of our framework

One of the main benefits of our approach is that by transforming an ethical planning problem into a classical one with utilities, it is possible to apply PDDL planners designed for this purpose. This is why in order to exemplify our framework we have implemented:

1. An extension of the language PDDL that models ethical rules and our qualitative preference model, and
2. Two routines to translate PDDL problems encoded with our ethical rules into equivalent PDDL encodings using soft goals and actions costs, by applying Proposition 5.7.

A distinction is made between the PDDL *formalization* of Chapter 2 and PDDL *code*, the computer language as defined in [Fox and Long, 2003]. We remark that the formalization presented previously is very similar to actual PDDL2.1 code, but with some reasonable simplifications. PDDL code provides a syntax to model classical planning problems with many possible extensions. The main difference between our PDDL formalization and the PDDL code we will use in our implementation is that the code will be able to use disjunctions in operator preconditions and effects, the equality symbol = to compare constants and variables, and object types. As explained in Appendix B, PDDL code defines the kinds of constructs that are allowed in a problem description through *requirements*. The requirements that we will allow in the PDDL code of our implementation are the following:

- **:strips** : allows using ‘add’ and ‘delete’ effects as specified in STRIPS,
- **:typing** : allows the specification of types and subtypes and assigning them to objects,
- **:equality** : allows using of the equality symbol = to compare two objects,
- **:conditional-effects** : allows using conditional expressions of the form $(\mathit{when}(\phi_1)(\phi_2))$ denoting that when the expression ϕ_1 holds in a state, then the expression ϕ_2 will be applied as an effect,
- **:disjunctive-preconditions** : allows using disjunction via the *or* symbol in operator preconditions,
- **:quantified-preconditions** : allows using quantification via the $forall$ and *exists* symbols in operator preconditions,
- **:adl** : macro requirement that adds all the preceding requirements, and
- **:negative-preconditions** : allows using negation via the *not* symbol in operator preconditions.

For more information about PDDL2.1, we refer the reader to [Fox and Long, 2003]. We chose this language, as we explained in Chapter 2, due to its wide use, particularly in all the International Planning Competitions (IPC). As a note, all of these extensions can be added to our formalization in the future, but we decided not to for simplicity reasons.

Briefly, PDDL files are divided into ‘domain’ and ‘problem’ files:

- Domain files serve to define the requirements of PDDL that are allowed, constants that can be used in actions, the fluents of the domain, the actions, and the types of objects, and
- Problem files are used to define the initial state, the goal state and the rest of the objects (constants that cannot be used in actions).

Some example PDDL code, including that of Example 4.1 can be found in Appendix B.

5.2.1 PDDL code extension

In order to implement the ethical planning elements we introduced in the previous sections, we will define an extension for the PDDL code which we will describe in what follows. Our extension of the PDDL code defines three constructs to specify ethical features, ethical ranked bases and ethical rules. All constructs are to be included in the domain file of the PDDL code.

The construct to represent the ethical features of a domain, as in Definition 4.6, is very similar to the one PDDL code uses to specify the fluents of a domain. It is defined as follows:

```
(:ethical-features
    (name_1 ?X1_1 - type1 ... ?Xn - typen_1)
    ...
    (name_m ?X1_m - type1 ... ?Xn - typen_m))
```

Listing 5.1: Ethical features definition PDDL code.

Where:

- **name**$_i$ is a unique string, i.e: does not coincide with any fluent, operator, constant, ethical rule name, or the identifier of any other element of the domain or problem file, and
- **?X1**$_i$ **- type1**$_i$ ... **?Xn**$_i$ **- typen**$_i$ is a list of the form **?Xj**$_i$ **- typej**$_i$ where each **Xj**$_i$ is a variable and **typej**$_i$ an object type defined in the domain file. Specifying the type of variables is an optional feature and can be ignored.

This construct allows to define all of the ethical features and as such, there should be only one definition using this type of construct in a domain file.

For instance, the ethical features of Example 4.9 can be modeled as follows:

```
(:ethical-features
    (danger ?C1 - car ?G1 - gravity)
    (damageRail ?C1 - car))
```

Listing 5.2: Ethical features PDDL code for Example 4.1.

Next, we can define ethical ranked bases, as in Definition 4.10, using the following construct:

```
(:ethical-rank
    :feature (name C1 ... Cn)
    :type T
    :rank R)
```

Listing 5.3: Ethical ranked base definition PDDL code.

Where:

- **(name C1 . . . Cn)** specifies the ground ethical feature for which one specifies its type and rank, **name** being the name of an ethical feature as specified in the first construct and each **Ci** a constant with a type specified in the domain file which matches **typei**, the type specified in the ethical feature definition, if the type is defined,
- T is either $+$ or $-$, denoting if the ethical feature is positive or negative, and
- R is a positive integer $i \in \mathbb{N}$, denoting the rank of the ethical feature.

The construct defines the rank of each ground ethical feature and whether it is positive or negative. We assume that at most one construct of this type is defined for each ground ethical feature and that in the case that a plan is assigned one ground ethical feature that has no rank and type, the ethical feature will be ignored, i.e: has rank zero, and we do not consider it in the ethical valuation of plans.

For instance, the ethical ranked base of Example 4.16 can be modeled as follows:

```
(:ethical-rank
    :feature (damageRail agent)
    :type -
    :rank 1)
(:ethical-rank
    :feature (damageRail c1)
    :type -
    :rank 1)
(:ethical-rank
    :feature (damageRail c2)
    :type -
    :rank 1)
(:ethical-rank
    :feature (danger agent low)
    :type -
    :rank 2)
(:ethical-rank
    :feature (danger c1 low)
    :type -
    :rank 2)
(:ethical-rank
    :feature (danger c2 low)
    :type -
    :rank 2)
(:ethical-rank
    :feature (danger agent high)
    :type -
    :rank 4)
(:ethical-rank
    :feature (danger c1 high)
    :type -
    :rank 3)
(:ethical-rank
    :feature (danger c2 high)
    :type -
    :rank 3)
(:ethical-rank
    :feature (responsibleAgent)
    :type -
    :rank 4)
```

Listing 5.4: Ethical ranked base PDDL code for Example 4.1.

Finally, the code of an ethical rule, as in Definition 4.7, is:

```
(:ethical-rule name
    :parameters (?X1 - type1 ... ?Xn - typen)
    :precondition ϕ1
    :activation ϕ2
    :features ϕ3)
```

Listing 5.5: Ethical rule definition PDDL code.

Where:

- **name** is a unique string, i.e: does not coincide with any fluent, operator, constant, ethical rule name, or the identifier of any other element of the domain or problem file,
- **?X1 - type1** … **?Xn - typen** is a list of the form **?Xj - typej** where each **Xj** is a variable and **typej** (optional) an object type defined in the domain file,
- ϕ_1 is a conjunction (encoded using the PDDL construct *and*) of literals over the fluents of the domain, using constants or variables from the parameter list $?X1-type1\ldots?Xn-typen$, specifying the preconditions of the ethical rule.
- ϕ_2 is either:
  - **null**, denoting that the ethical rule is defined via fluents, or
  - **(op P1 … Pk)**, an action formula where **op** is the name of an operator and **P1 … Pk** is a list of constants or variables included in the parameter list **?X1 - type1 … ?Xn - typen**, specifying the activation condition of the ethical rule.

- ϕ_3 is a conjunction (encoded using the PDDL construct *and*) of literals over the ethical features of the domain, using constants or variables from the parameter list $?X1 - type1 \ldots ?Xn - typen$, specifying the ethical features assigned or removed when this ethical rule is activated.

For instance, the ethical rules of Example 4.9 can be modeled as follows:

```
(:ethical-rule crashRule
    :parameters (?C1 - car)
    :precondition (hasCrashed ?C1)
    :activation null
    :features (danger ?C1 high))
(:ethical-rule bumpRule
    :parameters (?C1 - car)
    :precondition (hasBumped ?C1)
    :activation null
    :features (danger ?C1 low))
(:ethical-rule responsibleCrashRule
    :parameters ()
    :precondition (hasCrashed agent)
    :activation null
    :features (responsibleAgent))
(:ethical-rule responsibleBumpRule
    :parameters ()
    :precondition (hasBumped agent)
    :activation null
    :features (responsibleAgent))
(:ethical-rule railLeftRule
    :parameters (?C1 - car ?X1 - xPos ?Y1 - yPos)
    :precondition (and
        (position ?C1 ?X1 ?Y1)
        (direction ?C1 left)
        (nextX left ?X1 ?X1))
    :activation (go)
    :features (damageRail ?C1))
(:ethical-rule railRightRule
    :parameters (?C1 - car ?X1 - xPos ?Y1 - yPos)
    :precondition (and
        (position ?C1 ?X1 ?Y1)
        (direction ?C1 right)
        (nextX right ?X1 ?X1))
    :activation (go)
    :features (damageRail ?C1))
```

Listing 5.6: Ethical rules PDDL code for Example 4.1.

PDDL code that uses our constructs should include the **:ethical** requirement. With the constructs we defined in this section, we can represent ethical planning problems as specified in Definition 4.12.

The full PDDL code of the Example 4.1 can be found in Appendix B.

5.2.2 PDDL code translation routine

In this section, we will show how problems encoded using the three ethical constructs introduced in the last section can be translated to more classical PDDL problems without them.

Plan utilities in our formalization of classical planning could be specified using utility functions for final states, also called *soft goals*, or through *action costs*. PDDL code also provides the possibility to specify soft goals and action costs through different requirements. As demonstrated in Proposition 5.7, ethical preferences can be replaced with soft goals. Soft goals were introduced in PDDL3 along with other kinds of preferences. Therefore, we have implemented a routine that translates PDDL code with ethical constructs to normal PDDL3 code, so that any PDDL3 planner that can handle preferences may be used to plan with our ethical constructs.

Planning with preferences using PDDL3 was a major topic in the IPC5, i.e: the fifth International Planning Competition [Gerevini et al., 2009]. In particular, the *SimplePreferences* track of the competition served as a basis to compare different planners on PDDL3 problems using only soft goals, which is the only kind of preference our translation routine uses.

However, more recent IPCs [Torralba and Pommerening, 2018] have stopped holding planning competition tracks with the original PDDL3 preferences and instead have chosen to focus on action costs. Interestingly, it has been shown that soft goals can be compiled away [Keyder and Geffner, 2009], i.e: transformed into equivalent problems encoded differently, by replacing them with action costs via a simple routine. For this reason, we have implemented a second routine that transforms PDDL problems with ethical constructs into PDDL problems with action costs to allow more recent planners to be used to solve problems encoded with our framework.

In summary, we have implemented two transformation routines that will allow using a wide variety of state-of-the-art planners for problems encoded using our PDDL extension:

1. PDDL with ethical constructs into PDDL with soft goals, and
2. PDDL with ethical constructs into PDDL with actions costs.

These routines have been implemented using Python and are publicly available[1].

Routine 1: ethical to soft goals

As a first step, we have implemented a routine to translate PDDL problems with ethical constructs as detailed in the last section to PDDL3 problems with soft goals. Soft goals can be included in PDDL code through the **:preferences** requirement. This requirement enables the specification of an extensive set of preference constructs, which were introduced in PDDL3 [Gerevini and Long, 2005].

In this case, our translation uses the **preference** construct to specify soft goals:

```
(preference name_i (ϕ_i))
```

Listing 5.7: Preferences PDDL code.

[1]https://github.com/martinjedwabny/pddl-ethical

Where $name_i$ denotes the name of the ith preference and ϕ_i a PDDL expression. Expressions of this type can only be added to the goal description of a problem in PDDL code.

The translation uses this construct once for each defined ethical rank, which is the same as once per ground ethical feature. For instance, following Example 4.9, the preferences generated for the ethical features are:

```
(preference
    p_damagerail-agent (not (damageRail agent)))
(preference
    p_damageRail-c1 (not (damageRail c1)))
(preference
    p_damageRail-c2 (not (damageRail c2)))
(preference
    p_danger-agent-low (not (danger agent low)))
(preference
    p_danger-c1-low (not (danger c1 low)))
(preference
    p_danger-c2-low (not (danger c2 low)))
(preference
    p_danger-agent-high (not (danger agent high)))
(preference
    p_danger-c1-high (not (danger c1 high)))
(preference
    p_danger-c2-high (not (danger c2 high)))
(preference
    p_responsibleAgent (not (responsibleAgent)))
```

Listing 5.8: Example preference translation.

Briefly, for each ground feature e_i the translation generates a preference p_i with a name matching a formatted version (using the $-$ symbol) of e_i and the expression ϕ_i is **(not** e_i**)** in the case e_i is a negative ethical feature, or just **(**e_i**)** if it is positive. Intuitively, each preference expresses whether we desire the ethical feature to be present in the fluents of the final state reached by a plan.

Then, plans can be compared with these preferences using the **metric** construct:

```
(:metric minimize
    (+
        (* (is-violated p1) r1)
            ...
        (* (is-violated pn) rn)))
```

Listing 5.9: Metric PDDL code.

Where each p_i is a preference and r_i the corresponding rank of e_i the ith ethical feature. A metric defines a function that can be used to compare plans. This function takes two arguments, the first being either **minimize** or **maximize** and a list of numerical expressions. We will not delve into the details of the syntax of all the possible numerical expressions that can be encoded, which are specified in [Gerevini and Long, 2005]. Here we will simply use a sequence of **(* (is-violated (**p_i**)) (**r_i**))**, which amounts to minimizing the preferences violations, specified using the **is-violated** construct, which is the only way in which this version of PDDL permits to use preferences in the metric definition. According to the transformation defined in Proposition 5.7, we want to maximize the preferences assigned to a plan. However, due to the **is-violated** construct, we had to invert the metric from a maximization of preferences to a minimization of preference violations, for which equivalence is simple to check.

By translating PDDL code with our extended syntax into PDDL with soft goals, we enable any automated planner that supports the **:preferences** requirement to find plans using our PDDL extension and our translator. Many such planners can be found in the fifth International Planning Competition[2] proceedings [Gerevini et al., 2009] and more recent literature, such as:

- SGPlan[3] [Hsu et al., 2006],
- MIPS-XXL[4] [Edelkamp et al., 2006], and
- LPRPG-P[5] [Coles and Coles, 2011].

Routine 2: ethical to action costs

Recent International Planning Competitions [Torralba and Pommerening, 2018] have chosen to focus on preferences modeled only through action costs. This is why, as a second step into making our ethical planning problems compatible with state-of-the-art planners, we implemented a second transformation routine that translates PDDL code with ethical constructs into PDDL with action costs.

It has been shown [Keyder and Geffner, 2009] that soft goals can be compiled away (i.e: transformed into equivalent problems) and replaced with action costs with a simple routine. We have combined this approach with our previously described transformation to produce PDDL problems with action costs just as described in their research.

Action costs can be added to a PDDL problem description through the **:action-costs** requirement. PDDL problems that use this requirement should minimize or maximize a numerical variable `total-cost`. Furthermore, the requirement allows to add expressions of the type `(increase (total-cost) n)` and `(decrease (total-cost) n)` to operator effects, where *n* is a fixed numerical value.

This second routine replaces the preferences that the first routine adds to the 'problem' file of a translated PDDL description, using `(increase (total-cost) n)` and `(decrease (total-cost) n)` to operator effects in the 'domain' file, as described in [Keyder and Geffner, 2009]. Briefly, a set of operators are added to the PDDL code and they are forced to be executed at the end of each plan. Each of these operators checks which of the original soft goals are satisfied and therefore increases or decreases the `total-cost` accordingly.

[2]https://lpg.unibs.it/ipc-5/
[3]https://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/
[4]http://sjabbar.com/mips-xxl-planner
[5]https://nms.kcl.ac.uk/planning/software/lprpgp.html

We can find several action-cost-optimal planners through the Fast Downward project[6] and the ninth International Planning Competition[7] [Torralba and Pommerening, 2018], such as:

- Fast Downward Stone Soup [Seipp and Röger, 2018],
- Fast Downward LM-Cut [Helmert and Domshlak, 2011], and
- Delfi [Katz et al., 2018].

The resulting PDDL code of both transformation routines for the PDDL code description of Example 4.1 can be found in Appendix B.3 and B.4.

5.3 Experimentation

Classical planning with soft goals and action costs is known to be a PSPACE-complete problem in the general case [Menkes Van Den Briel et al., 2004, Aghighi and Bäckström, 2015]. However, from a practical standpoint, we wanted to test the two implementations described in the last section and compare them using different state-of-the-art planners to better understand their computational efficiency. For this reason, we asked ourselves the following questions:

Q1: How much do ethical features and rules degrade the running time performance of a planning problem?

Q2: Which of the two transformation routines (described in the last section) has better running time efficiency when coupled with their respective (soft goal vs. action cost) state-of-the-art planners?

Q3: How does the planning problem size (in terms of fluents and actions) affect the performance when adding ethical features and rules?

Experiments

We designed two sets of experiments to answer these questions. All these experiments consisted in measuring the running time a planner takes to find the optimal plan for various planning problems. Both sets of experiments took as basis problems from the IPCs and added ethical features, ranks and rules to measure the performance degradation as the number of ethical constructs increased. While the first set of experiments was targeted to analyze the performance degradation when adding ethical features, the second one looked into the performance loss when adding ethical rules. Ethical ranks were not studied separately as they have a one-to-one correspondence with ethical features, i.e: for each added ethical feature, a corresponding ethical rank (and ethical type) must be defined.

[6]https://www.fast-downward.org/
[7]https://ipc2018-classical.bitbucket.io/

Concretely, we took eight problems of medium complexity from the IPC5[8] [Dimopoulos et al., 2006], half from the 'pathways' domain, while the other half belonged to the 'openstacks' domain, and added them ethical features, ranks and rules. The problems we used were propositional, i.e: no variables, conditional effects, or quantification, to prevent compatibility issues between the tested planners, as each planner supported a different subset of the PDDL syntax. It remains to test the performance of planners using these extra elements, although in preliminary examination they did not seem to increase the computational performance compared to their grounded counterparts, presumably because most planners ground the variables in a planning problem before solving it regardless.

Experiment 1 The first set of experiments consisted in analyzing the performance degradation when the number of ethical features N increased. For each planning problem T_0 (of the base IPC problems mentioned before) and integer $N = 1, 2, \ldots$, we generated an extended problem T_N resulting from adding to T_{N-1} a new ethical feature f_N with a randomized rank in the range $[1, (N/2) + 1]$ and a random type $+$ or $-$, and an ethical rule $r = \langle r_N(), \{\}, Act(r), \{f_N\}\rangle$ where $Act(r)$ is any action of the problem, chosen randomly. That is, we only added ethical rules without any parameters (as we tested problems without variables) or preconditions, that only activate the ethical feature f_N that was added in the current iteration. With this approach, we guaranteed that each problem T_N directly extends the previous iteration T_{N-1}. Analyzing ethical rules defined via fluents, and/or different sets of preconditions and activated ethical features remain for future work, however, preliminary testing suggested the added complexity is similar. Considering variables in ethical features/rules, however, would lead to adding as many ethical features/rules as all of their ground instances, according to our implementations.

Experiment 2 For the second set of experiments, we first took each original planning problem T_0 and added ten ethical features $f_{i \in [0,9]}$ with a randomized rank in the range $[1, 6]$ and a random type $+$ or $-$. Then for each integer $N = 1, 2, \ldots$, we generated an extended problem T_N resulting from adding to T_{N-1} a new ethical rule $r = \langle r_N(), \{\}, Act(r), \{f\}\rangle$ where $f \in \{f_0, \ldots, f_9\}$ is any of the ethical features and $Act(r)$ is any action of the problem chosen randomly. Compared to the first experiments, this second batch did not add an ethical feature at each step, but rather added ten ethical features at the start and then increased the number of ethical rules at each iteration.

Methodology of evaluation

For each planning problem tested in these experiments, we averaged the running time across 10 identical runs, increasing the number $N = 1, 2, \ldots$ of ethical features/rules until either: (i) the average running time across the

[8]https://lpg.unibs.it/ipc-5/

10 runs for a certain N surpassed a predefined time limit, which we set at 3 minutes, or (ii) N reached a certain limit, which we set at 60 for ethical features, and 200 for ethical rules.

As one of the purposes of our experimentation was to compare the two translation routines we described in the last chapter when using their corresponding state-of-the-art planners, we ran all the experiments with both soft-goal and action-cost compatible planners, to test the two translation routines and compare their results. We used the LPRPG-P planner [Coles and Coles, 2011] for the 'ethical to soft goals' translation, which was shown to have better time performance across most IPC domains than the IPC5 planners (e.g: MIPS-XXL [Edelkamp et al., 2006] and SGPlan [Hsu et al., 2006]) which we did not test because they faced several compatibility issues related to the operating system (Mac OSX) used in the experimentation machine. More recent IPCs did not feature competition tracks for planners supporting soft goals. Thus, for the 'ethical to action costs' routine, we tested the translated problems against some state-of-the-art planners which competed in the latest IPCs [Torralba and Pommerening, 2018], belonging to the Fast Downward project. In particular, we used Fast Downward Stone Soup version 1 [Seipp and Röger, 2018], and LM-Cut [Helmert and Domshlak, 2011].

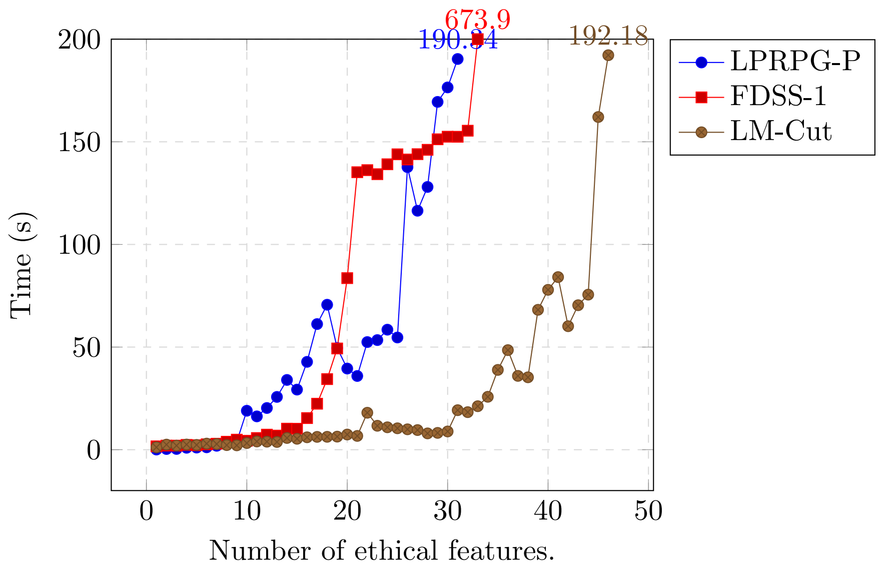
Turning to the running conditions, these experiments were conducted on a 1,6 GHz Intel Core i5 CPU MacOS system with 8 GB 2133 MHz LPDDR3 RAM. We ensured the maximum resources of the machine were assigned to each experiment and ran under the same conditions by removing all nonessential processes and utilizing the Python `os.nice`[9] functionality, which ensured our tests were given the highest possible priority and CPU time.

Results

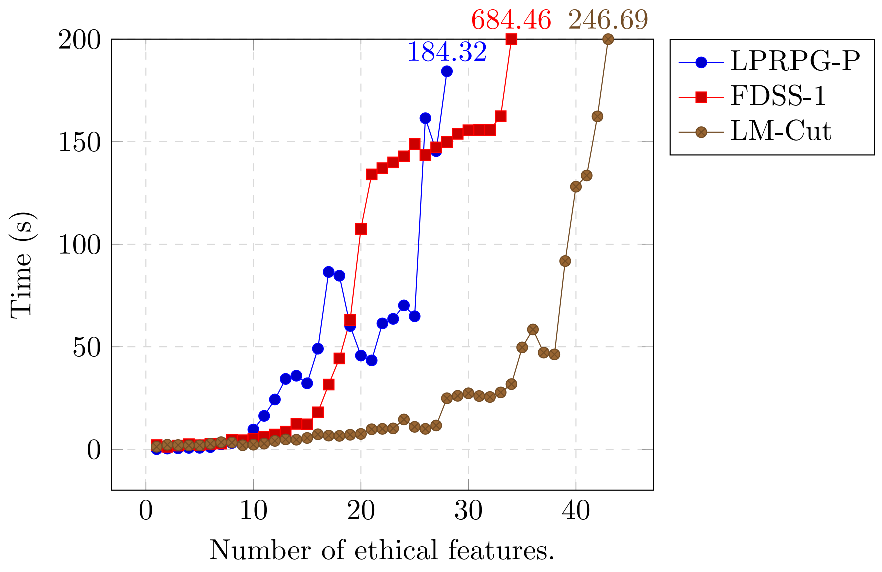
As we can see in Figure 5.1 and 5.2, the increase in planning time when adding ethical features is significant and apparently above linear across the tested domains and planners for both translation schemes (except for Problems 1 and 2 in Figure 5.2 for some planners). The Y axis denotes the running time of planners to find the optimal plans, while the X axis measures the number of ethical features, which is equal to the number of ethical ranks and rules in this set of experiments. In terms of the translation routines, adding ethical features amounts to extending problems with one fluent per ethical feature and one action effect per ethical rule. Then, the first routine also adds one preference (soft goal) for each ethical feature, while the second routine adds two actions per ethical feature to be executed at the end of each plan to add a numerical cost for each broken preference (for more detail on this, see [Keyder and Geffner, 2009]).

We observe in Figure 5.1 that for the four problems of the 'openstacks' domain, the planners can handle 30-45 features until exceeding the time limit. The results in this domain were fairly similar across problems, in

[9]https://docs.python.org/3/library/os.html



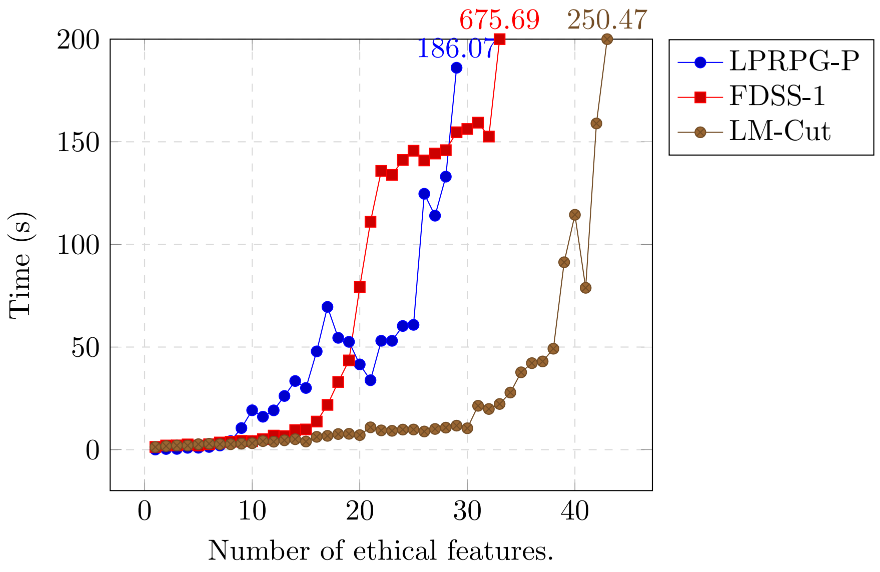
(a) Problem 1



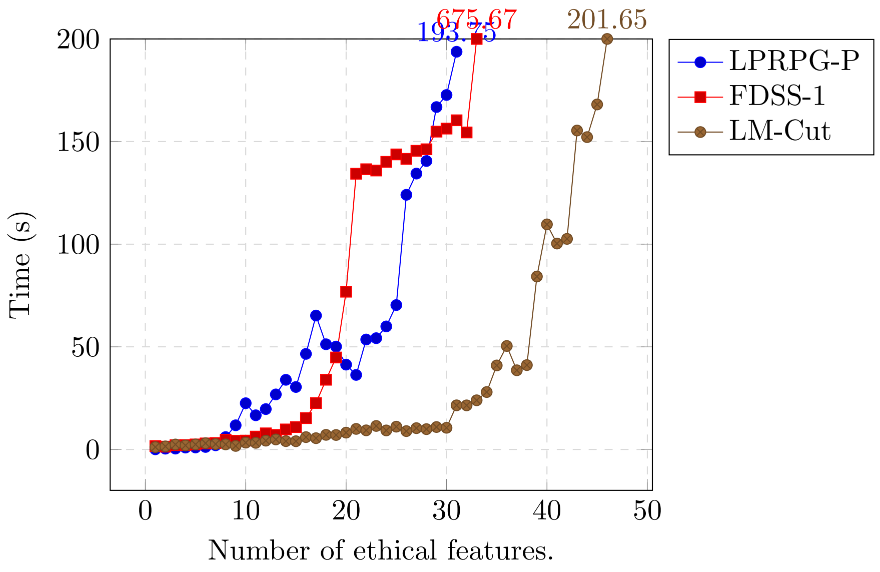
(b) Problem 2

Figure 5.1: Openstacks planning runtime by ethical features.

that the action-cost-based FDSS-1 and LM-Cut were more performant than the soft-goal-based LPRPG-P, and LM-Cut could handle the most amount of ethical features in all cases. Then, for the 'pathways' domain (Figure 5.2), the action-cost-based planners FDSS-1 and LM-Cut can handle 25 and 50 features, respectively, for the small problems (1 and 2) until exceeding the time limit, while the soft-goal-based LPRPG-P could compute optimal plans even with 60 ethical features, which as we mentioned, is the maximum. For the medium-sized 'pathway' problems (3 and 4), however, FDSS-1 could barely handle any ethical features and exceeded the time limit almost immediately, while LM-Cut and LPRPG-P got similar results,



(c) Problem 3



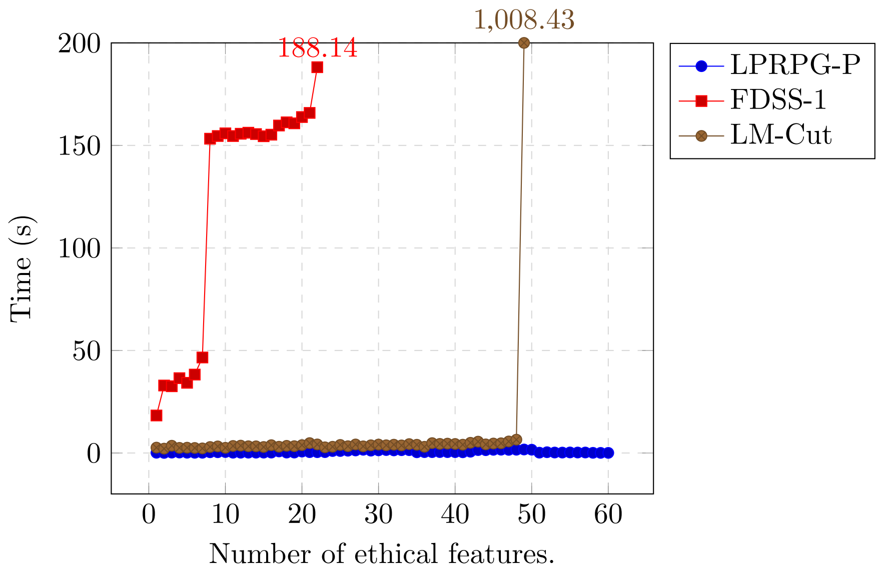
(d) Problem 4

Figure 5.1: Openstacks planning runtime by ethical features. (cont.)

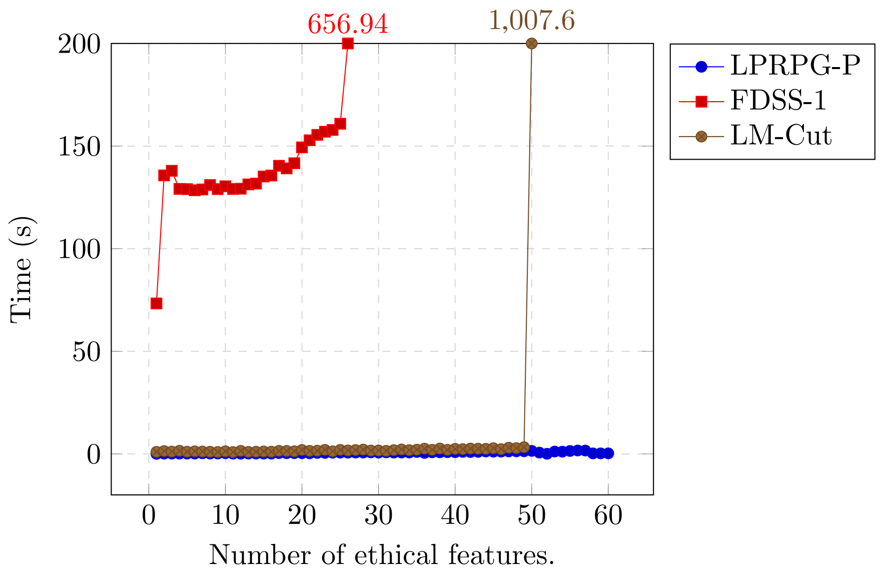
begin able to handle 40-50 features in Problem 3 and 25-30 in Problem 4.

Overall, we can see that in most cases, the increase in running time seems to follow a similar trend across the three planners and the two translation routines. As for the planners, the action-cost-based (FDSS-1 and LM-Cut) were more performant in the 'openstacks' domain than the soft-goal-based one (LPRPG-P), while the inverse could be said for the 'pathways' domain. Focusing on the action-cost-based ones, LM-Cut was more performant than FDSS-1 in all domains and problems.

We observe a different trend for ethical rules in Figure 5.3 and 5.4 to the previous sets of experiments. The Y axis denotes the running time of



(a) Problem 1

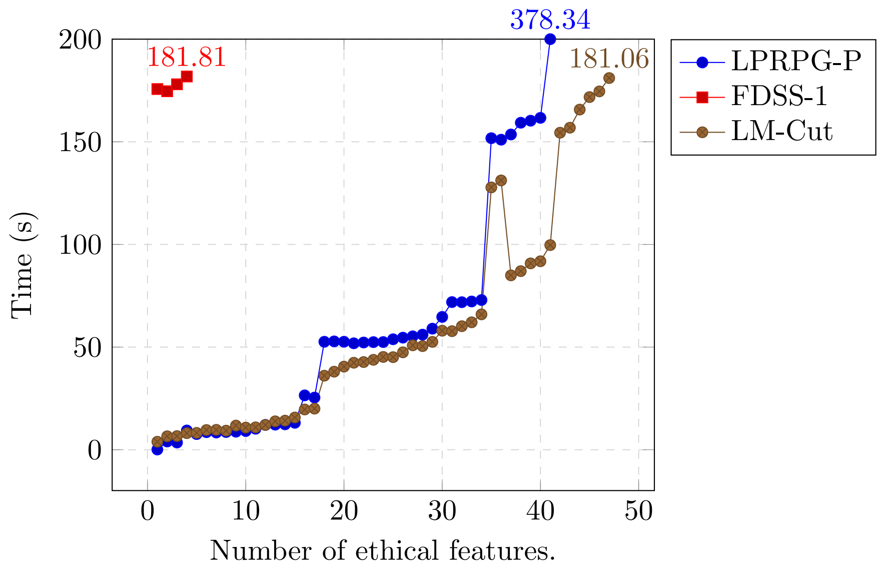


(b) Problem 2

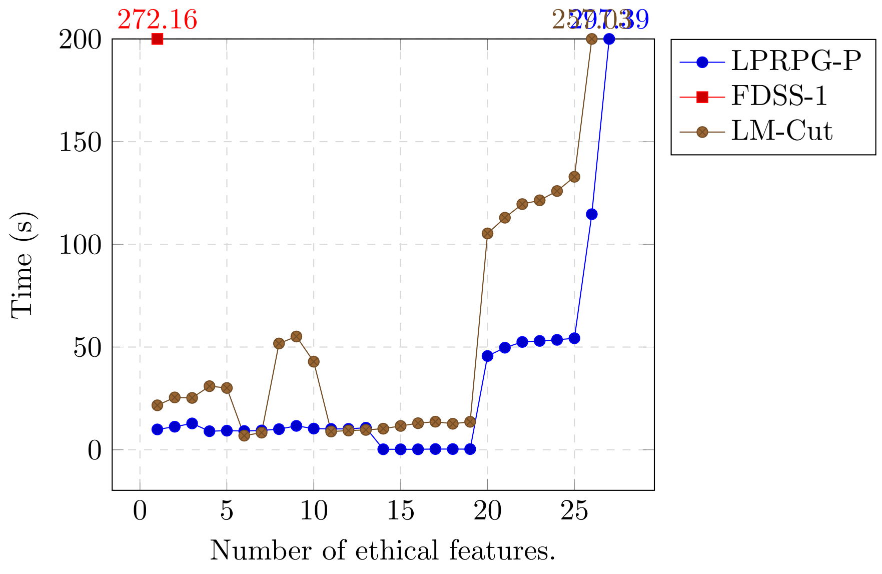
Figure 5.2: Pathways planning runtime parametrized by features.

planners to find the optimal plans and the X axis measures the number of ethical rules, while the ethical features were kept fixed at 10 across experiments. For the translation routines, adding ethical rules at each step amounts to adding an action effect to the action which is the activation condition of the rule. Although in these experiments the ethical features were kept fixed at 10 and we simply increased the number of ethical rules, the increase in planning time is substantial across the tested domains and planners for both translation schemes.

We see that in all problems of the 'openstacks' domain, the planners could handle the maximum amount of ethical rules (200) while not reach-



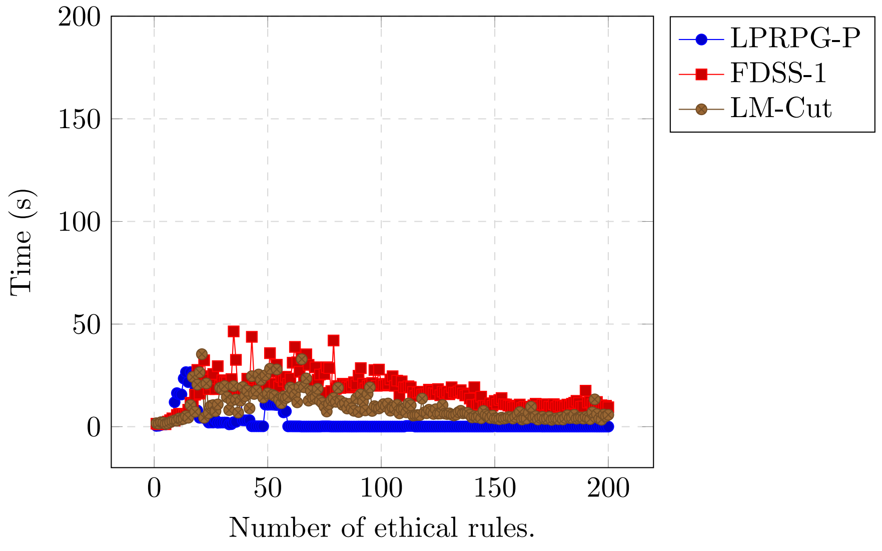
(c) Problem 3



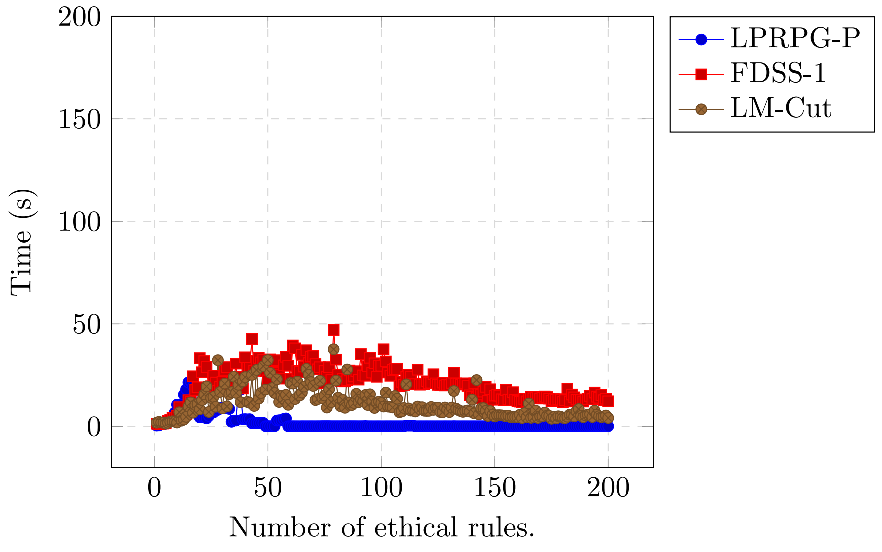
(d) Problem 4

Figure 5.2: Pathways planning runtime parametrized by features. (cont.)

ing the time limit. LPRPG-P was consistently the fastest planner, followed by LM-Cut, and then FDSS-1. Interestingly, for all planners, the running time increased slightly until they reached 50-100 ethical features and then decreased, but never surpassed 50 seconds, still far from the time limit. This phenomenon repeated across all the problems of this domain. The experiments for the 'pathways' domain only showed similar results for the first two problems. Namely, in Problems 1 and 2 the running times increased and then decreased, and never surpassed the time limit, but the same did not happen for the other two problems. Both for problems 3 and 4, the planner FDSS-1 exceeded the time limit almost immediately.



(a) Problem 1

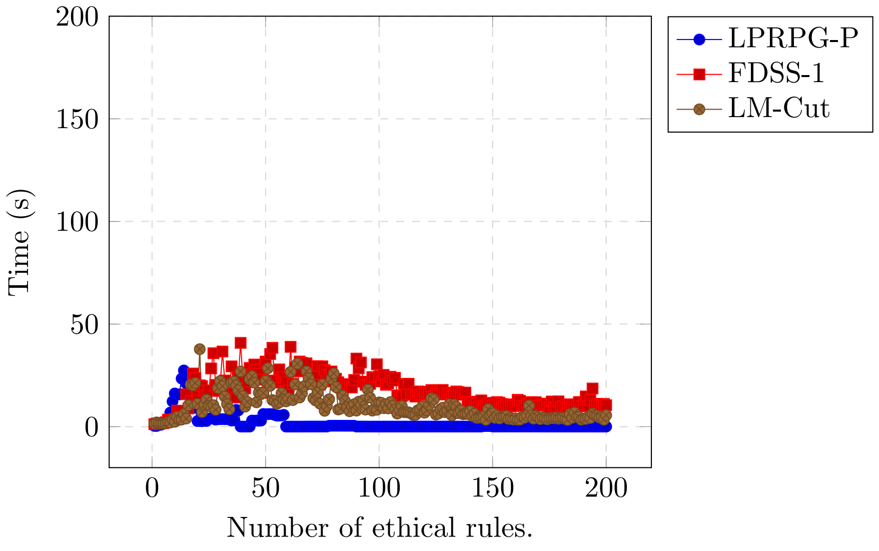


(b) Problem 2

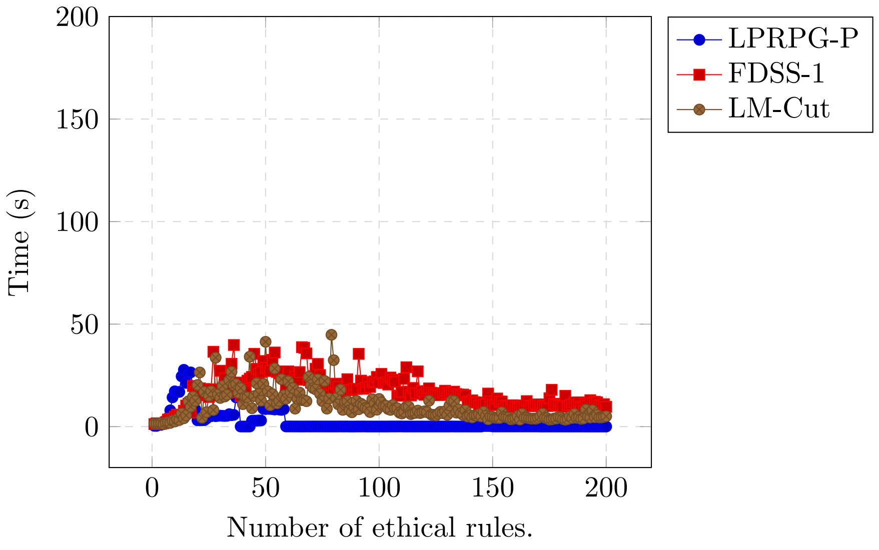
Figure 5.3: Openstacks planning runtime parametrized by rules.

Then for Problem 3, both LPRPG-P and LM-Cut could handle up to 200 ethical rules, LPRPG-P took more time until 100 ethical rules, but then the planning time became lower than LM-Cut. And lastly, for Problem 4, LPRPG-P exceeded the time limit at around 25 ethical rules, while LM-Cut could handle up to 150.

In most cases, adding ethical rules increased the running time slightly and all the planners could handle up until the maximum amount (200) of ethical rules, with the exception of the last problem of the 'pathways' domain. Differently than for ethical features, the action-cost-based planners (FDSS-1 and LM-Cut) performed worse than the soft-goal-based one (LPRPG-P) across all the problems of both domains, except for the last



(c) Problem 3



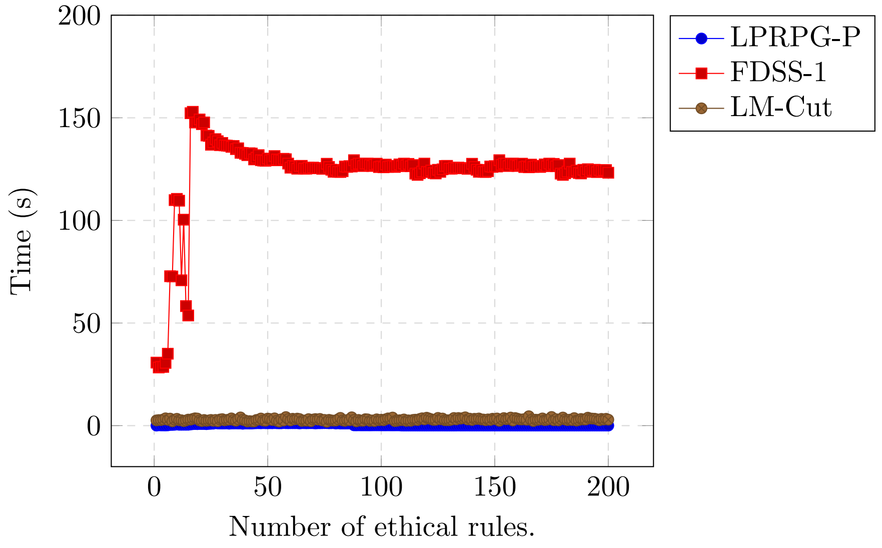
(d) Problem 4

Figure 5.3: Openstacks planning runtime parametrized by rules. (cont.)

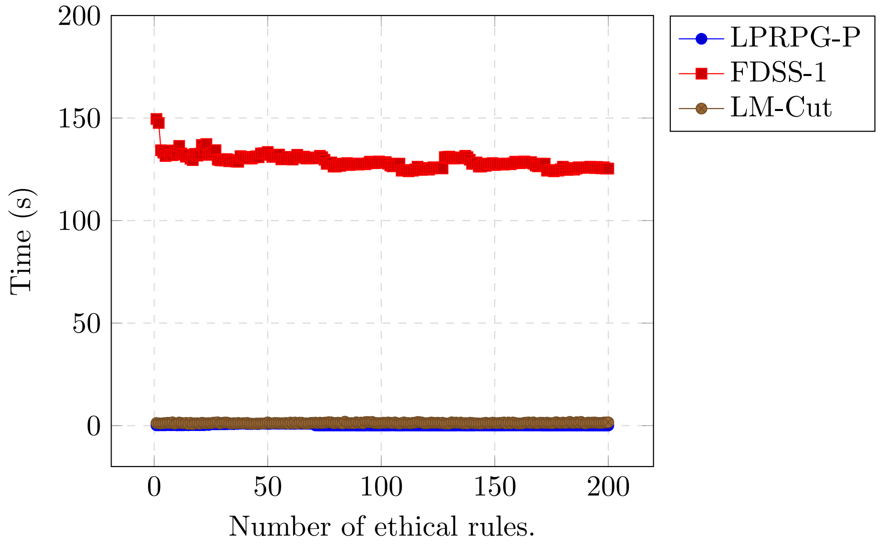
problem of the 'pathways' domain. Because most planners could handle the maximum amount of ethical rules for almost all the problems and the running time seemed to decrease after a certain amount of them, we suspect that either (i) adding ethical rules does not increase the complexity significantly in all cases if the amount ethical features are not also increased, or (ii) the tested problems were not complex enough for the number of ethical rules to matter, as Figure 5.4d might suggest.

To answer the three questions posed at the start of this section:

Q1: For all the tested domains and problems, it seems that the increase in planning time when adding ethical features is above linear across planners for both translation schemes, but if the ethical features are



(a) Problem 1

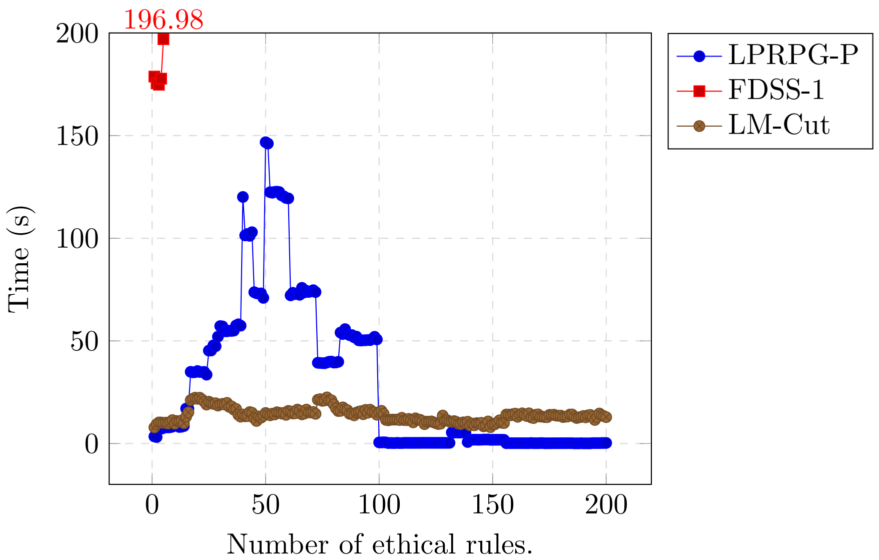


(b) Problem 2

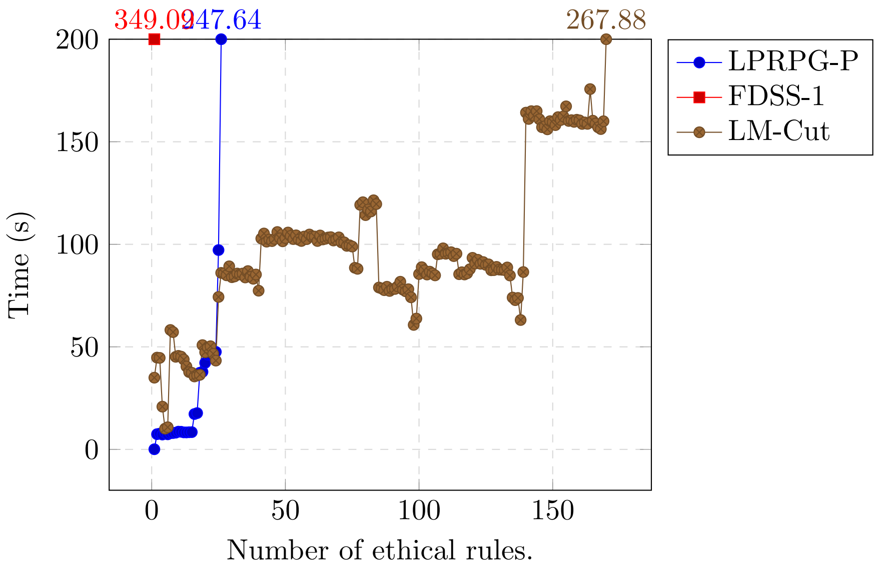
Figure 5.4: Pathways planning runtime parametrized by rules.

kept fixed and only the amount of ethical rules increases, the running time of the planner tends to increase until a certain point and then decrease. From a practical standpoint, the state-of-the-art planners we tested could handle a sizeable amount of ethical features (25-60) and ethical rules (175-200) for all the tested instances considering the high complexity of the IPC problems we used.

Q2: Although the first translation routine adds significantly fewer actions to planning domains compared to the second one, we cannot always say that the planners that are compatible with the output of the routine had better performance than the others across all experiments.



(c) Problem 3



(d) Problem 4

Figure 5.4: Pathways planning runtime parametrized by rules. (cont.)

For the first set of experiments (testing ethical features), the action-cost-based planners (FDSS-1 and LM-Cut) seemed to have better performance for the 'openstacks' domain than the soft-goal-based (LPRPG-P), but the same does not hold for the 'pathways' domain. Then for the second set of experiments (testing ethical rules), the soft-goal-based planner (LPRPG-P) had better performance in all but the last problems of the 'pathways' domain, however, in all of these problems LPRPG-P was the best planner and all of the planners could handle all of the ethical rules until 200 (with the exception of FDSS-1 in Figure 5.4c). Thus, the results seem inconclusive in this matter.

Q3: There was only a noticeable difference observed in the performance degradation between small (1 and 2) and medium-sized (3 and 4) problems for the 'pathways' domain. In Figure 5.2a and b, we can see that LPRPG-P can handle all the ethical features up until 60 (the limit) and LM-Cut over 50, but in the medium-sized problems (Figure 5.2c and d), the degradation is much more pronounced. Then regarding ethical rules, we can see in Figure 5.4a and b, that all the planners can handle all the ethical rules up until 200 (the limit), while the degradation is much higher for the medium-sized problems in Figure 5.4c and d.

Lastly, all the code and domains used here are publicly available[10].

5.4 Related work

We based the linearization and valuation function from Section 5.1 on [Feldmann et al., 2006] but adapted it to take into account our positive and negative ethical features, which assigns a numerical value to plans according to their ethical features. Compared to their work, our model and translation routine into classical planning problems have to deal with the extra constructs of ethical features and rules. Additionally, we provide a second translation routine, into the more currently used PDDL language with only action costs.

Regarding our translation routines, the second one was based on the work of [Keyder and Geffner, 2009], which demonstrated how soft goals could be compiled away. Yet, compared to their implementation which is for STRIPS problems, ours can deal with several advanced PDDL features described in Section 5.2.

To test our implementation, in Section 5.3 we tested several planners against some problems from the IPC5 [Gerevini et al., 2009]. The planners we tested were LPRPG-P [Coles and Coles, 2011] for the 'ethical to soft goals' translation, which was shown to have better time performance across most IPC domains than the IPC5 planners (e.g: MIPS-XXL [Edelkamp et al., 2006] and SGPlan [Hsu et al., 2006]). We did not test these IPC5 planners because they faced several compatibility issues related to the operating system (Mac OSX) used in the experimentation machine. More recent IPCs did not feature competition tracks for planners supporting soft goals. And then, for the 'ethical to action costs' routine, we used some state-of-the-art planners which competed in the latest IPCs [Torralba and Pommerening, 2018]: Fast Downward Stone Soup version 1 [Seipp and Röger, 2018], and LM-Cut [Helmert and Domshlak, 2011].

5.5 Discussion

Throughout this chapter, we have:

[10]https://github.com/martinjedwabny/pddl-ethical

1. shown how to extend the preferences introduced in the last chapter so that all plans are comparable through a linearization and valuation function,
2. demonstrated how the ethical planning problems we introduced in the last chapter can be solved using existing planners by compiling the ethical preferences into either soft goals or action costs,
3. provided two translation routines showing the effectiveness of our approach, and
4. analyzed the computational runtime performance of the translated problems using some problems from the IPCs.

The linearization and valuation functions defined in Section 5.1 dealt with the problems we mentioned in the last chapter about many plans being incomparable when they have disjoint positive or negative ethical features of the same rank. Practically, using this valuation function amounts to enforcing a lexicographical ordering on the plans with respect to their assigned ethical features. In other words, we start by checking from the highest ranked features until at some level we determine that the positive features assigned to a plan minus the negative ones have a higher count than other plans. From an ethical standpoint, this means that the problem designer has to make sure that ethical features of the same rank have indeed the same priority.

Turning to Proposition 5.7, we have shown that our ethical planning problems can be translated into classical planning problems with soft goals. This poses the question of whether our approach really is qualitative or quantitative as it is reduced to a maximization of utilities, which is something that we wanted to avoid a priori. However, the ethical ranks we used and the way in which they are compiled guarantee that features of higher rank maintain priority over all the other lowered rank ones, thus keeping a certain level of qualitativeness.

Regarding the two questions asked at the beginning of this chapter, we have shown how to solve our ethical planning problems, as described in the last chapter, using existing AI planning technology by developing a translation into classical planning problems. We have provided two translation routines, one into soft goals and another into action costs. The second translation routine was based on the work of [Keyder and Geffner, 2009], which demonstrated how soft goals could be compiled away.

In terms of experimentation, we have performed some preliminary tests to show that our approach is efficient for some medium-sized instances of very hard problems from the IPCs.

Future work It remains to confirm the reason behind the odd behavior of planners in the experiments we analyzed, regarding ethical rules. Also, we wish to test the performance of planners using more advanced PDDL features, such as variables, conditional effects and quantification, although in preliminary examination they did not seem to increase the computational performance compared to their grounded counterparts. Likewise, we would like to measure the performance of more complex ethical rules, e.g: using parameters, complex activation conditions and a large number of activated and deactivated ethical features.

Furthermore, specifying that a certain feature e of a specific rank r is worth five times more than the others of the same rank, is not possible. However, one simple extension for future work could be to add an extra value number in the ethical rank, in other words, instead of having $b(e) = \langle t, r \rangle$ in Definition 4.10, we could have $b(e) = \langle t, r, n \rangle$, where $t \in \{+, -\}$, $r \in \mathbb{N}$ is the ethical rank and $n \in \mathbb{N}$ is a value that indicates that feature e is worth n features at the rank r.

Finally, it would be an interesting challenge to provide a planner specific to ethical planning problems that do not need these translation routines and compare its performance against state-of-the-art planners solving the translated counterparts to these problems.

6 Inferring ethical preferences

As we explained in Chapter 3, ethical reasoning systems can be divided into *top-down* (modeling and applying ethical principles or restrictions), *bottom-up* (learning what decisions are ethical, based on datasets through learning algorithms), and *hybrid* (combining methods of both).

Learning how to act in ethically-nuanced domains from a corpus of data like bottom-up approaches is, of course, a highly debated subject. The main reason for this is related to the concept of explainability. Namely, some authors [Tolmeijer et al., 2020] have pointed out the difficulty of trusting a black-box system in ethical domains because the machine would not be able to justify its decisions, so any potential harm to humans or property would be hard to understand or fix. In recent years, the Artificial Intelligence (AI) field has shown a strong interest in developing systems that provide explanations for their answers, giving rise to what is known as explainable AI (XAI) [Xu et al., 2019]. Several methods have been developed to address the concerns raised by the lack of explainability in various AI subfields. Previous XAI literature has established two primary ways of providing explainability [Arrieta et al., 2020]: (i) transparent systems in which the reasoning is directly interpretable by humans, and (ii) post-hoc models to explain black-box systems [Ribeiro et al., 2016]. Naturally, both approaches have their drawbacks. While transparent systems frequently sacrifice response quality or computing performance in order to provide fully interpretable results, post-hoc explanations are often seen as limited since they provide reconstructions that may or may not directly represent the black-box system's reasoning process.

So far, we have developed a framework to compare sequences of actions on ethical terms through numerical ranks in the context of classical planning (Chapter 4) and provided two methods of applying state-of-the-art planners to solve them through translation procedures back into classical planning with utilities (Chapter 5). By itself, the system described in the previous chapters can be applied as a *top-down* ethical reasoning framework because it implements the ethical aspects of a problem by modeling them in a symbolic manner. However, in this chapter, we will demonstrate how our approach can be used as a *hybrid* ethical system by inferring the ranks of ethical features through learning-based methods in the style of *bottom-up* methodologies.

Eliciting ethical preferences for automated systems has been studied in the context of the GenEth system [Anderson and Anderson, 2018]. In their work, the authors used the opinion of domain experts (ethicists) to learn ethical choices with Inductive Logic Programming (ILP) [Muggleton, 1991], a technique for inferring logic rules from examples in the form of logic facts. Yet, this system cannot handle conflicting judgments, meaning that for two similar situations, it does not admit two domain experts submitting different answers. Furthermore, eliciting the opinion of ethicists can be an expensive solution when an enormous amount of situations and special cases need to be considered.

Instead, in this chapter, we will consider probabilistic logic, which extends facts and rules with probabilistic annotations, which will allow handling conflicting judgments in datasets and, more importantly, aggregating these judgments from many non-experts, making it possible to construct the dataset in a crowdsourced manner and cover a much bigger number of possible situations and special cases.

Moreover, we have chosen to use a probabilistic logic-based method instead of other machine learning approaches because of its capability of providing explanations in the form of logic rules. That is, logic-based systems provide a transparent mechanism of reasoning. It is paramount that the reasoning of a system, and in our case its preferences, be directly interpretable by humans in ethically-nuanced domains to make sure the agent follows societal principles, particularly when these preferences are inferred from a large corpus of noisy data.

In this chapter, we will use a technique called parameter learning [Gutmann et al., 2011], described in Chapter 2, which allows to infer probabilistic annotations for facts and rules encoded in Problog [De Raedt et al., 2007], a probabilistic logic programming language (see Section 2.1.2). Using this method, we will show how to infer ethical ranks using the opinions of users or domain experts by learning the probability of each combination of ethical features and ranks resembling the dataset of opinions.

Thus, our approach will be focused on: (i) considering a corpus of ethical choices submitted by a large number of people that are not necessarily experts, (ii) handling conflicting judgments about what the most ethical option is in a certain situation, and (iii) learning the ethical ranks of different features as described by our framework in Chapter 4.

As such, the questions we will try to solve in this chapter are the following:

Research Questions in this Chapter

- How can one determine the ethical rank of the features involved in an ethical planning problem based on a corpus of opinions?
- How computationally efficient is this method in practice?

Let us note that this chapter is a continuation of the work in [Jedwabny et al., 2021b].

The next sections are structured as follows. In Section 6.1, we give an overview of the method we will use to infer the rank of ethical features. Section 6.2 covers the implementation of the parameter learning problem using the Problog language. Then, in Section 6.3, we will exemplify the mechanics of this approach through a case study. In Section 6.4, the reader will find some experimental results regarding the computational costs of the approach. Section 6.5 compares our work with previous literature. And finally, Section 6.6 discusses the contributions of this chapter and perspectives for future work.

6.1 Method overview

Thus far, Chapters 4 and 5 described an extension of PDDL to compare plans on ethical terms. We defined an ethical planning problem as a tuple:

$$T = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b) \rangle$$

Which can be divided into $(\mathcal{D}, s_0, g)$, the elements of a classical planning problem (see Chapter 2), and (E, R, b), the ethical aspects of T, where:

- E is the set of ethical features,
- R are the ethical rules used to assign ethical features to plans, and
- $b(e) = \langle Type(e), Rank(e) \rangle$ the ethical ranked base, which assigns a type to a feature $e \in E$, i.e: $Type(e) \in \{-, +\}$, and a rank, i.e: $Rank(e) \in \mathbb{N}_0$ denoting whether the feature is ethically good or bad, and its level of priority, respectively.

Then, we described how our model can be used to obtain the most ethical plan π for a problem $T = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b) \rangle$ using a planner compatible with our ethical constructs (which we implemented by translating T to an equivalent classical planning problem with utilities), as depicted in Figure 6.1.

Problem description The task we will tackle in this chapter is learning the ranks of ethical features when they are not provided in advance, captured by $Rank : E \mapsto \mathbb{N}_0$ (marked in blue in Figure 6.1), a function that corresponds to the right side of the ethical ranked base b.

Instead, we will use a **dataset of opinions** from users or experts to infer them, given that all other elements of the problem $\mathcal{D}, g, E, R$ and $Type$:

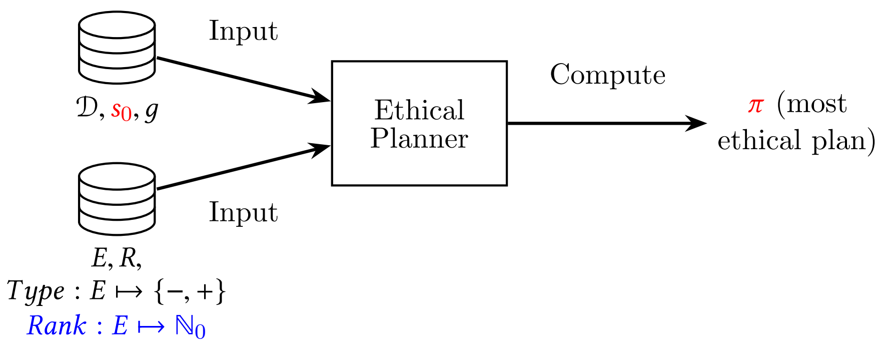

Input
$\mathcal{D}, s_0, g$
Ethical Planner
Compute
π (most ethical plan)
Input
$E, R,$
$Type : E \mapsto \{-, +\}$
$Rank : E \mapsto \mathbb{N}_0$

Figure 6.1: Problem overview: predefined elements (black), dataset elements (red), and target function to learn (blue).

$E \mapsto \{-, +\}$ (black elements in Figure 6.1) are predefined. This dataset can be constructed by asking the users/experts to choose the most ethical plan π in a series of possible initial situations s_0 (both marked in red in Figure 6.1), according to their own opinion. We can consider an iterative elicitation of opinions as the one described in Figure 6.2, in which users/experts u_i ($i = 1, \ldots, n_u$) are presented with different initial situations s_j ($j = 1, \ldots, n_s$) and possible plans described in text format and choose the plan $\pi_{i,j}$ that u_i considers the most ethical in s_j.

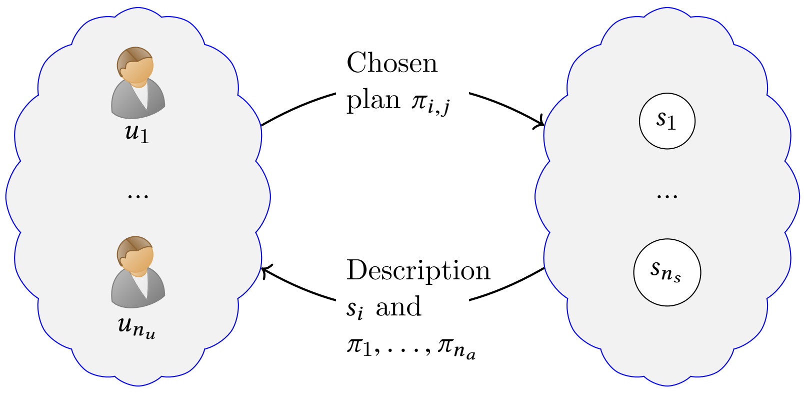


Figure 6.2: Preference elicitation.

As such, the situations s_j and plans π_k would correspond to those of T and could be generated automatically by running a planner and extracting all the sequences of actions that reached the goal. However, the generation of these situations and plans will not be addressed in this thesis, but rather, we will focus on the learning of ethical ranks, as mentioned earlier. Also, we will note that such an elicitation procedure could also require the textual descriptions of situations and plans to be carefully constructed, which could be manually crafted after the situations and plans are generated.

In short, the problem of this chapter can be described as: (i) *learning* the function $Rank : E \mapsto \mathbb{N}_0$, (ii) *according* to a dataset composed of pairs $(s_j, \pi_{i,j})$ denoting the chosen plan of user u_i for the initial situation s_j, (iii) *given the predefined* elements $\mathcal{D}, g, E, R, Type : E \mapsto \{-, +\}$ of an ethical planning problem.

Parameter learning Let us recall the parameter learning technique described in Chapter 2 (see Section 2.1.3), a method that consists of:

- **Finding** a set of probabilistic annotations $p_1, \ldots, p_n$,
- **For a given** parametrized Problog program $P(p_1, \ldots, p_n) = P_{fixed} \cup P_{param}$ where P_{param} is the only part of the program that uses $p_1, \ldots, p_n$ as probabilistic annotations,
- **Such that** it resembles a dataset Is, composed of evidential interpretations $I_i = (I_i^+, I_i^-)$ as close as possible. In other words, maximizing the probabilities $Pr(I_i|P(p_1, \ldots, p_n))$ of each interpretation I_i holding according to $P(p_1, \ldots, p_n)$.

In the next sections, we will develop an encoding for: (i) $P(p_1, \ldots, p_n) = P_{fixed} \cup P_{param}$, that represents all of the relevant information of a planning problem T and determines which plan is the most ethical in each situation, and (ii) Is, such that it contains the opinions users/experts regarding which plan was the most ethical in the different initial situations.

More specifically, P_{fixed} will contain all the information about the initial situations, possible plans, and ethical features of a planning problem (including their types and which features are assigned to each plan), and a set of rules to deduce Problog facts of the type `best(A,S)` denoting that the plan A is the best in the initial situation S. However, P_{fixed} will not contain information regarding the ranks of ethical features, and thus will only be able to deduce `best(A,S)` for a situation S if this information is added in the form of a logical fact.

The ranks of ethical features will be encoded using a *rank assignment*, which is a ground fact of the form `rank_assignment(...)` containing all the information regarding the rank of every ethical feature, and we will describe it more precisely in the following section.

Thus, $P_{fixed} \cup \{$`rank_assignment(...)`$\}$ would be able to deduce `best (A,S)` by querying the Problog program, i.e: which plan A is the best in each situation S.

Due to the fact that the rank of ethical features is the target of this learning procedure, it will be the objective of the parameter learning method. As such, it will be part of P_{param}, the parametrized part of the overall Problog program $P(p_1, \ldots, p_n)$, that will consist of various logical facts:

```
p1::rank_assignment(...);
      ⋮
pn::rank_assignment(...).
```

Where each fact `rank_assignment(...)`, which we will specify in the next section, will denote one possible rank assignment. In other words, if we would fix the values of $p_1, \ldots, p_n$, then $P(p_1, \ldots, p_n) = P_{fixed} \cup P_{param}$ would be able to deduce a probability for each ground fact of the form `best(A,S)`.

Please note that the ';' symbol after each rank assignment in P_{param} denotes annotated disjunctions. By using annotated disjunctions we ensure that the sum of all probabilistic annotations sums to 1 and that no two rank assignments will be true at the same time in any possible world, as we explained in Section 2.1.2.3.

On the other hand, Is (the dataset) will be composed of evidential interpretations $I_i = (I_i^+, I_i^-)$ composed of facts of the form $\texttt{best}(a, s)$ indicating that for the ith user or domain expert the plan a is the best in situation s, where:

$$I_i^+ = \{\texttt{best}(a_{i,j}, s_j) : a_{i,j} \text{ is the best plan for user } i \text{ in situation } s_j\}$$
$$I_i^- = \{\texttt{best}(a, s_j) : s_j \text{ has plan } a \text{ and } a \neq a_{i,j}\}$$

In summary, the task of this chapter will be to find the best probabilistic annotations $p_1, \ldots, p_n$ for P_{param}, which encode the probability of each rank assignment:

```
p1::rank_assignment(...);
       ⋮
pn::rank_assignment(...).
```

And thus, using the parameter learning technique, we will obtain a probability distribution over the possible rank assignments deducing the opinions in the dataset and ultimately, which is the rank assignment that most resembles the dataset, which will correspond to the one with the highest probabilistic annotation.

6.2 Problem encoding

In this section, we will describe the Problog encoding of the program $P(p_1, \ldots, p_n)$, mentioned in the previous section, and the dataset Is. We will also show how we can use these encodings to infer the ethical rank of features using parameter learning, as described in Chapter 2 (see Section 2.1.3).

As discussed previously, the Problog encoding of a parameter learning problem $P(p_1, \ldots, p_n) = P_{fixed} \cup P_{param}$ is divided into a parametrized part P_{param} containing the probabilistic annotations to learn, and a fixed part P_{fixed} without parameters. For clarity, we decided to separate these second part into two Problog programs $P_{fixed} = P_s \cup P_t$, thus:

- The fixed Problog program P_s will encode the information about the situations, plans, ethical features and their possible ranks,
- The fixed Problog program P_t will encode a set of rules that can deduce the predicate `best(A,S)` identifying that plan A is the best in situation S, and
- The parametrized Problog program P_{param} will encode the possible rank assignments and the probabilistic annotations to learn.

It is important to note that these encodings will use some common Problog extensions that we did not mention in Chapter 2, namely negation-as-failure, numerical expressions, and lists, which are built-in functionalities of the language, and are explained in Appendix Section C.1.

In the following subsections, we will describe the encodings of these Problog programs and the set of interpretations Is representing the dataset.

6.2.1 Domain encoding P_s

The background knowledge in P_s about the situations $s_1, \ldots, s_{n_s}$, plans $a_1, \ldots, a_{n_a}$ and ethical features $e_1, \ldots, e_{n_e}$ can be easily encoded using first-order logic ground facts of the form:

- `situation(`s`)` : specifying the possible (initial) situations s,
- `plan(`a`)` : specifying the possible plans a,
- `ethical_feature(`e`)` : specifying the ethical features e,
- `rank(r)` : specifying the possible ranks r that the ethical features might take,
- `type(e,t)` : indicating the type $t \in \{-, +\}$ (positive or negative) of the ethical feature e,
- `has_plan(a,s)` : stating that plan a can be executed in situation s, and
- `has_feature(e,a,s)` : denoting that the ethical feature e is assigned to plan a in situation s.

More precisely, by using a Problog program P_s as follows:

```
situation(s_1).
    ⋮
situation(s_{n_s}).
plan(a_1).
    ⋮
plan(a_{n_a}).
ethical_feature(e_1).
    ⋮
ethical_feature(e_{n_e}).
type(e_1,t_1).
    ⋮
type(e_{n_e},t_{n_e}).
rank(r_1).
    ⋮
rank(r_{n_r}).
has_plan(a_{1,1},s_1). ... has_plan(a_{1,m_1},s_1).
```

```
    ⋮
has_plan($a_{n,1}, s_n$). ... has_plan($a_{n,m_n}, s_n$).
has_feature($e_{1,1,1}, a_{1,1}, s_1$).
    ⋮
has_feature($e_{1,m_1,w_{1,m_1}}, a_{1,m_1}, s_1$).
    ⋮
has_feature($e_{n,1,1}, a_{n,1}, s_n$).
    ⋮
has_feature($e_{n,m_n,w_{n,m_n}}, a_{n,m_n}, s_n$).
```

Listing 6.1: Problog encoding of situational background knowledge P_s

Please note that both the ground facts of the type `has_plan(a,s)` and `has_feature(e,a,s)` should be specified depending on previously defined knowledge, for instance, based on the specification of the planning problem at hand.

6.2.2 Theory encoding P_t

The Problog program P_t defines a series of (non-probabilistic) logical rules that build up to encode the predicate `best(A,S)`, denoting that plan A is the best in situation S, ethically speaking.

These rules assume they already count with one rank assignment. A rank assignment is a ground fact of the form:

```
rank_assignment([$e_1$, $r_1$, ..., $e_{n_e}$, $r_{n_e}$])
```

Where $[e_1, r_1, \ldots, e_{n_e}, r_{n_e}]$ is a list, the e_i are constants representing an ethical feature and each r_i is a number denoting its rank. Thus, the list in the argument of this predicate encapsulates all the ranks of every ethical feature. For instance, if the set of ethical features is $\{a, b\}$, the ground fact `rank_assignment([a,1,b,2])` represents that ethical feature a has rank 1 and b has rank 2.

Given a ground fact of this form, the Problog program P_t defines `best (A,S)` as follows:

```
best(A,S) :-
    has_plan(A,S),
    not(worse(A,B,S)).

worse(A,B,S) :-
    has_plan(A,S),
    has_plan(B,S),
    A \= B,
    val(A,S,N1),
    val(B,S,N2),
    R1 < R2.
```

Listing 6.2: Problog encoding of *best* in background knowledge P_t

Where:

- `best(A,S)` indicates that plan A is the best in situation S, and is defined by checking that no other plan B has a higher valuation than A using the predicate `worse(A,B,S)` and negation as failure, and
- `worse(A,B,S)` represents that plan A is worse than B in situation S, which is defined by checking that the plans are not identical and comparing the valuations of each plan with the predicate `val(A,S,N)`.

In Chapter 5, we defined the valuation $val(A) = val(E_A) \in \mathbb{N}_0$ of plan A, which is a non-negative integer that represents how ethically good the plan A is, according to the ethical features assigned to A in a given ethical planning problem. This is exactly what the predicate `val(A,S,N)` represents. It unifies N with the valuation of plan A in the considered planning problem and initial situation S, according to Definition 5.2, as follows:

```
val(A,S,N) :- val_until_rank(A,S,$n_r$,N).

val_until_rank(A,S,0,0).
val_until_rank(A,S,R,N) :-
    rank(R),
    R > 0,
    amount_satisfied_of_rank(A,S,R,N1),
    val(R,V),
    R1 is R-1,
    val_until_rank(A,S,R1,N2),
    N is N1*V+N2.

val(0,0).
val(R,V) :-
    rank(R),
    R > 0,
    R1 is R-1,
    max_val(R1,V1),
    V is V1+1.

max_val(0,0).
max_val(R,V) :- rank(R), R > 0,
    amount_of_rank(R,N),
    val(R,V1),
    R1 is R-1,
    max_val(R1,V2),
    V is V1*N+V2.
```

Listing 6.3: Problog encoding of *val* in background knowledge P_t

Where:

- `val(A,S,N)` specifies that the plan A has valuation N in situation S, which is defined using the predicate `val_until_rank(A,S,`n_r`,N)`, that unifies N with the valuation of A considering features of rank at most n_r (specified in P_s as the highest rank),

- val_until_rank(A,S,R,N) indicates that the total valuation of a plan A in situation S is N if one only considers ethical features of rank up until R. That is, it val_until_rank(A,S,R,N) is an auxiliary predicate used to aggregate the valuation of A across each rank.
- val(R,V) unifies V with val_R as described by Definition 5.2, indicating the gain V in valuation that a set of ethical features obtains by including a feature of rank R, and
- max_val(R,V) unifies V with $maxval_R$ as described by Definition 5.2.

Notice that val_until_rank(A,S,R,N) depends on the logical predicate amount_satisfied_of_rank(A,S,R,N), which unifies N with the amount of ethical features satisfied that have rank R. Similarly, max_val(R,V) depends on the predicate amount_of_rank(R,N), which unifies N with the total amount of ethical features of rank R.

These two predicates are defined using the following rules:

```
amount_of_rank(R,N) :-
    rank(R),
    rank_assignment(RA),
    amount_of_rank(R,N,RA).

amount_of_rank(R,0,[]) :-
    rank(R).
amount_of_rank(R,N,[F,R1|RA]) :-
    rank(R),
    R = R1,
    amount_of_rank(R,N1,RA),
    N is N1+1.
amount_of_rank(R,N,[F,R1|RA]) :-
    rank(R),
    R \= R1,
    amount_of_rank(R,N,RA).

amount_satisfied_of_rank(A,S,R,N) :-
    has_plan(A,S),
    rank(R),
    rank_assignment(RA),
    amount_satisfied_of_rank(A,S,R,N,RA).

amount_satisfied_of_rank(A,S,R,0,[]) :-
    has_plan(A,S),
    rank(R).
amount_satisfied_of_rank(A,S,R,N,[F,R1|RA]) :-
    has_plan(A,S),
    rank(R),
    R = R1,
    satisfies(F,A,S),
    amount_satisfied_of_rank(A,S,R,N1,RA),
    N is N1+1.
amount_satisfied_of_rank(A,S,R,N,[F,R1|RA]) :-
    has_plan(A,S),
    rank(R),
    R = R1,
    not(satisfies(F,A,S)),
    amount_satisfied_of_rank(A,S,R,N,RA).
amount_satisfied_of_rank(A,S,R,N,[F,R1|RA]) :-
    has_plan(A,S),
    rank(R),
    R \= R1,
    amount_satisfied_of_rank(A,S,R,N,RA).

satisfies(F,A,S) :-
    has_feature(F,A,S),
    type(F,'+').
satisfies(F,A,S) :-
    type(F,'-'),
    not(has_feature(F,A,S)).
```

Listing 6.4: Problog encoding of $amount_satisfied_of_rank$ and $amount_of_rank$ in background knowledge P_t

Where:

- `amount_of_rank(R,N)` unifies N with the total amount of ethical features of rank R by using the predicate `rank_assignment(RA)` which unifies RA with the list of ranks and ethical features, and the auxiliary predicate `amount_of_rank(R,N,RA)`,
- `amount_of_rank(R,N,RA)` unifies N with the total amount of ethical features of rank R by iterating over the elements of RA, for a given a rank R and rank assignment RA,
- `amount_satisfied_of_rank(A,S,R,N)` unifies N with the total amount of features of rank R satisfied by action A in situation S, using `rank_assignment(RA)` to get the rank of features, and the auxiliary predicate `amount_satisfied_of_rank(A,S,R,N,RA)`,
- `amount_satisfied_of_rank(A,S,R,N,RA)` unifies N with the total amount of features of rank R satisfied by action A in situation S by iterating over the rank assignments in RA, and
- `satisfies(F,A,S)` indicates that plan A satisfies ethical feature F in situation S, this means that either A has ethical feature F and F is a positive feature, or that A does not have feature F and the feature is negative,

Please note that the combination of the programs P_s and P_t is self-contained in the sense their rules only use predicates that are either provided by Problog or are present in the program itself, except for one: `rank_assignment(RA)`. Indeed, because the purpose of this encoding is to allow us to **infer** these ranks using parameter learning from evidence, it will be the target of the parametrized part of the final program.

For the full listing of P_t, we refer the reader to Appendix C.

6.2.3 Dataset encoding Is

As mentioned before, the ranks of ethical features can be inferred through parameter learning from evidence by using a set of evidential interpretations $I_i = (I_i^+, I_i^-)$ encoding the opinion of user u_i about the situations. Namely, by using an elicitation procedure like the one described before, we can encode the opinion of each user u_i that picks the plan $a_{i,j}$ as the best for situation s_j using an interpretation I_i, where:

$$\begin{aligned} I_i^+ &= \{\texttt{best}(a_{i,j}, s_j) : j \in [1, m] \text{ and } a_{i,j} \text{ is the choice of } u_i \text{ in } s_j\} \\ I_i^- &= \{\texttt{best}(a, s_j) : j \in [1, m],\ s_j \text{ has plan } a \text{ and } a \neq a_{i,j}\} \end{aligned}$$

So, given the opinion of a set of users $u_1, \dots, u_{n_u}$, the set of evidential interpretations $Is = \{I_1, \dots, I_{n_u}\}$ will be the dataset for the parameter learning procedure.

6.2.4 Parameter learning encoding

Finally, we are ready to present the encoding of the parameter learning problem to infer the rank of features, according to its definition in Chapter 2 (see Section 2.1.3).

Definition 6.1 (Ethical ranks learning)**.** The problem of finding the probability of each rank assignment being the one that most resembles the opinion of users/experts can be encoded via parameter learning from evidence as follows:

- **Find** a set of probabilistic annotations:

$$\widehat{p_1}, \dots, \widehat{p_n}$$

- **Given** a set of possible rank assignments:

$$\begin{aligned} &\texttt{rank\_assignment([}e_1\texttt{,}r_{1,1}\texttt{,}\dots\texttt{,}e_{n_e}\texttt{,}r_{1,n_e}\texttt{]);} \\ &\quad\vdots \\ &\texttt{rank\_assignment([}e_1\texttt{,}r_{n,1}\texttt{,}\dots\texttt{,}e_{n_e}\texttt{,}r_{n,n_e}\texttt{]).} \end{aligned}$$

And the parametrized Problog program:

$$P(p_1, \ldots, p_n) = P_s \cup P_t \cup P_{param}$$

Where P_s and P_t are as defined in the previous sections, and P_{param} is the parametrized set of ground facts:

$$p_1\texttt{::rank\_assignment([}e_1\texttt{,}r_{1,1}\texttt{,}\ldots\texttt{,}e_{n_e}\texttt{,}r_{1,n_e}\texttt{]);}$$
$$\vdots$$
$$p_n\texttt{::rank\_assignment([}e_1\texttt{,}r_{n,1}\texttt{,}\ldots\texttt{,}e_{n_e}\texttt{,}r_{n,n_e}\texttt{]).}$$

- **Such that** $P(\widehat{p_1}, \ldots, \widehat{p_n})$ resembles a dataset Is, composed of interpretations $I_i = (I_i^+, I_i^-)$, as close as possible:

$$(\widehat{p_1}, \ldots, \widehat{p_n}) = \underset{\substack{(p_1,\ldots,p_n)\in[0,1]^n \\ p_1+\ldots+p_n=1}}{\text{argmax}} \prod_{i=1,\ldots,k} Pr(I_i | P(p_1, \ldots, p_n))$$

Essentially, this definition reduces the task of inferring the optimal ranks to an encoding of a parameter learning problem, as described in Chapter 2, but taking the possible rank assignments as part of the input. As mentioned before, the program $P_s \cup P_t$ can deduce `best(A,S)` for any plan A and situation S if `rank_assignment(RA)` is specified. Therefore, $P(p_1, \ldots, p_n) = P_s \cup P_t \cup P_{param}$ can also deduce `best(A,S)`, for instance, if $p_1 = 1$ and all other probabilistic annotations are set to zero. However, in the general case, this can result in a poor overall probability $Pr(I_i|P(p_1, \ldots, p_n))$ if we consider every evidential interpretation I_i. That is, it will not adjust to all the members of the dataset as best as possible. In turn, using parameter learning here will provide a probability of each rank assignment being the correct one according to the dataset of evidential interpretations.

We can use this result to pick the optimal (ith) `rank_assignment([`e_1 `,` $r_{i,1}$`,` ..., e_{n_e}`,` r_{i,n_e}`])`, such that $p_i = \max_{j=1,\ldots,n} p_j$. In other words, we can find the rank assignment that maximizes the probability of the Problog program $P(p_1, \ldots, p_n)$ entailing the dataset.

It is also important to note that this procedure can be simplified by removing impossible, or undesired rank assignments from P_{param}. This can be useful in cases in which one knows beforehand that some ethical features will always have a higher rank than others, and thus removing all rank assignments that do not satisfy this constraint will make sense.

In the next section, we will see this parameter learning problem in action by presenting a case study and exemplifying how undesired rank assignments can be eliminated beforehand.

6.3 Example: a case study

Here, we will show how the Problog parameter learning problem from the last section works in practice.

Situation	Plan	Ethical features				Expert's opinion				
		e_{da}	e_{d1}	e_{d2}	e_{ra}	u_1	u_2	u_3	u_4	u_5
s_1	go(left)	x	x	x						
	go(right)					x	x	x	x	x
s_2	go(left)	x								x
	go(right)		x	x		x	x	x	x	
s_3	go(left)			x		x	x	x		
	go(right)		x						x	x
s_4	go(left)		x	x		x	x	x	x	x
	go(right)	x	x		x					
s_5	go(left)	x				x	x	x	x	x
	go(right)	x			x					
s_6	go(left)		x			x	x	x	x	x
	go(right)		x	x						

Table 6.1: Experts' opinions and ethical features for the case study regarding plans $go(left)$ and $go(right)$

We will employ a setting loosely based on Example 4.1, which we utilized in the last two chapters, and apply the parameter learning technique using an **artificial** dataset to illustrate its results.

As system designers, we are tasked to implement an ethical reasoning module for an autonomous vehicle using the framework we presented in the previous chapters, with the model specified in Example 4.16. All of the input is given as an ethical planning problem $T = \langle \mathcal{D}, s_0, g, \mathcal{E} = (E, R, b) \rangle$, with the exception of the ethical ranks $Rank : E \mapsto \mathbb{N}_0$, specified in $b(e) = \langle Rank(e), Type(e) \rangle$ of the features $e \in E$.

For this reason, we ask domain experts $u_1, \ldots, u_5$ to submit their opinion about different situations the system might face. Each situation $s_1, \ldots, s_6$ describes textually a scenario that the autonomous vehicle could encounter in which the agent's car is circulating along with two other cars c_1 and c_2. These situations can be considered a possible initial state of the planning problem, created to elicit the expert's opinions. We can think of this opinion elicitation as something similar to the Moral Machine experiment [Awad et al., 2018].

In each situation s_i, there are two possible plans the vehicle can execute: $go(left)$ and $go(right)$ representing a sequence of actions that make the vehicle change its direction and go to a certain lane. There are a total of four ethical features: $e_{da} = danger(agent)$, $e_{d1} = danger(c1)$, $e_{d2} = danger(c2)$ indicating that a certain car is in danger, and $e_{ra} = responsible(agent)$ denoting that whatever danger is produced by the plan, the responsibility will be assigned to the agent.

We describe all the information regarding the ethical features of each plan and some artificial opinions from experts in Table 6.1. Briefly, each cross on an 'Ethical feature' column denotes that the corresponding plan and situation are assigned that ethical feature, while crosses in the 'Expert's opinion' column represent which plan the expert chose as the most ethical.

In situations s_1, s_5 and s_6 every expert agrees on the same option because only one of the two possible plans has a set of ethical features that is strictly contained in the ones of the other, and all ethical features are negative. Then, most experts choose $go(right)$ in s_2 as they consider that an automated agent should prioritize the well-being of its own passengers rather than that of the other cars' passengers. A similar argument can be made for $go(left)$ in s_4. And last, the opinion is divided on s_3 as the agent experts cannot clearly differentiate between producing danger towards c_1 and c_2.

The description of the situations, plans, ethical features and ethical ranks are captured by the Problog facts P_s in the following code listing, as described in Table 6.1. We assume that ethical features can only take a rank of 1 or 2.

```
situation(s1).
     ⋮
situation(s6).

plan(go(left)).
plan(go(right)).

ethical_feature(danger(agent)).
ethical_feature(danger(c1)).
ethical_feature(danger(c2)).
ethical_feature(responsible(agent)).

type(danger(agent), '-').
type(danger(c1), '-').
type(danger(c2), '-').
type(responsible(agent), '-').

rank(1).
rank(2).

has_plan(go(right), s1).
has_plan(go(left), s1).
has_feature(danger(c1), go(left), s1).
has_feature(danger(c2), go(left), s1).
has_feature(danger(agent), go(left), s1).

     ⋮

has_plan(go(right), s6).
has_plan(go(left), s6).
has_feature(danger(c1), go(right), s6).
has_feature(danger(c2), go(right), s6).
has_feature(danger(c1), go(left), s6).
```

Listing 6.5: Case study Problog encoding of situational background knowledge P_s

For the full listing of the code, we refer the reader to Appendix Section C.3.1.

Then, the opinion of experts $u_1, \ldots, u_5$ is captured by the set of evidential interpretations $Is = \{I_1, \ldots, I_5\}$, as described in Table 6.1:

$$
\begin{aligned}
I_1^+ = \{&\texttt{best(go(right),s1)}, \texttt{best(go(right),s2)},\\
&\texttt{best(go(left),s3)}, \texttt{best(go(left),s4)},\\
&\texttt{best(go(left),s5)}, \texttt{best(go(left),s6)}\}\\
I_1^- = \{&\texttt{best(go(left),s1)}, \texttt{best(go(left),s2)},\\
&\texttt{best(go(right),s3)}, \texttt{best(go(right),s4)},\\
&\texttt{best(go(right),s5)}, \texttt{best(go(right),s6)}\}\\
&\vdots\\
I_5^+ = \{&\texttt{best(go(right),s1)}, \texttt{best(go(left),s2)},\\
&\texttt{best(go(right),s3)}, \texttt{best(go(left),s4)},\\
&\texttt{best(go(left),s5)}, \texttt{best(go(left),s6)}\}\\
I_5^- = \{&\texttt{best(go(left),s1)}, \texttt{best(go(right),s2)},\\
&\texttt{best(go(left),s3)}, \texttt{best(go(right),s4)},\\
&\texttt{best(go(right),s5)}, \texttt{best(go(right),s6)}\}
\end{aligned}
$$

We refer the reader to Appendix Section C.3.2 for the complete specification of the evidential interpretations.

Having defined P_s (describing the situations, plans, ethical features and ranks) and I_s (denoting the experts' opinions), we can use Definition 6.1 in conjunction with the Problog program P_t as described in the last section, to encode an ethical rank learning problem that finds the probability distribution for rank assignments.

Nonetheless, we will not consider all 16 possible rank assignments. In particular, we will only consider those that regard the well-being of cars $c1$ and $c2$ with the same priority. This reduces the parameters of the learning problem to 8, half of the maximum amount. Because the possible rank assignments are part of the input, the system designer can specify which rank assignments will be tested and compared. As we shall see in the next section, the total number of possible rank assignments has a big impact on the computation time of the parameter learning algorithm.

The ethical rank learning problem is defined as follows:

- **Find** a set of 8 **optimal** probabilistic annotations:

$$\widehat{p_1}, \ldots, \widehat{p_8}$$

- **Given** the parametrized Problog program:

$$P(p_1, \ldots, p_8) = P_s \cup P_t \cup P_{param}$$

Where P_{param} is the parametrized set of ground facts:

```
p1::rank_assignment([danger(agent),1,danger(c1)
    ,1,danger(c2),1,responsible(agent),1]);
p2::rank_assignment([danger(agent),1,danger(c1)
    ,1,danger(c2),1,responsible(agent),2]);
p3::rank_assignment([danger(agent),1,danger(c1)
    ,2,danger(c2),2,responsible(agent),1]);
p4::rank_assignment([danger(agent),1,danger(c1)
    ,2,danger(c2),2,responsible(agent),2]);
p5::rank_assignment([danger(agent),2,danger(c1)
    ,1,danger(c2),1,responsible(agent),1]);
p6::rank_assignment([danger(agent),2,danger(c1)
    ,1,danger(c2),1,responsible(agent),2]);
p7::rank_assignment([danger(agent),2,danger(c1)
    ,2,danger(c2),2,responsible(agent),1]);
p8::rank_assignment([danger(agent),2,danger(c1)
    ,2,danger(c2),2,responsible(agent),2]).
```

- **Such that** $P(\widehat{p_1}, \ldots, \widehat{p_8})$ resembles Is as close as possible:

$$(\widehat{p_1}, \ldots, \widehat{p_8}) = \underset{\substack{(p_1,\ldots,p_8)\in[0,1]^8 \\ p_1+\ldots+p_8=1}}{\operatorname{argmax}} \prod_{i=1,\ldots,k} Pr(I_i | P(p_1, \ldots, p_8))$$

By executing the parameter learning module in Problog with this encoding, we find the optimal set of probabilistic annotations:

$$\widehat{p_1} = 0.281427747318699$$
$$\widehat{p_2} = 0.04244114096646$$
$$\widehat{p_3} = 0.010678487452607$$
$$\widehat{p_4} = 0.072823894422998$$
$$\boldsymbol{\widehat{p_5} = 0.323127403277696}$$
$$\widehat{p_6} = 0.200735781945672$$
$$\widehat{p_7} = 0.002320025221626$$
$$\widehat{p_8} = 0.066445519394241$$

Therefore, the rank assignment with the highest probabilistic annotation ($\widehat{p_5}$) is that encoded by:

```
rank_assignment([
    danger(agent),2,
    danger(c1),1,
    danger(c2),1,
    responsible(agent),1]).
```

This means that for the given opinions, the rank of `danger(agent)` should be higher than the other ethical features, which are all ranked 1, which makes sense considering the data in Table 6.1, described earlier.

Next, we will test this approach and analyze its running time performance when increasingly large amounts of ethical features, ranks, or experts' opinions are considered.

6.4 Experimentation

Parameter learning is a computationally expensive task, due to the fact that it requires computing the probability of each evidential interpretation holding as a result of a parametrized Problog program and changing these parameters iteratively [Gutmann, 2011]. Furthermore, querying in Problog can be a hard problem by itself [De Raedt et al., 2007].

In this section, we will analyze how varying different properties of the problem increases the running time of the parameter learning algorithm used for inferring ethical ranks. In order to use the framework we described in the last chapters as a hybrid approach, the learning process must be computationally feasible. For this reason, we asked ourselves the following questions:

Q1: How much does the number of rank assignments, ethical features, and possible ranks degrade the parameter learning running time?

Q2: How much does the size of the input dataset (amount of expert opinions) degrades the parameter learning running time?

Experiments

To answer these questions, we designed two sets of experiments, which consist in measuring the running time for the learning of the parameters (i.e: the probability of each rank assignment) with varying amounts of ethical features and dataset size. These experiments were based on the case study problem introduced in the last section and expanded with additional elements, such as the ethical features, ranks, rank assignments and dataset opinions. To describe these experiments, we will use the following notations that quantify the number of elements in the learning problem:

- F: number of ethical features,
- R: maximum rank, denoting that the possible ranks are $1, 2, \ldots, R$,
- S: number of initial situations,
- F_{xp}: number of ethical features assigned to each plan in the problem,
- O: number of opinions, i.e: the size of the dataset, and

- A: number of possible rank assignments.

Both the data and test programs are publicly available[1].

Experiment 1 In the first set of experiments, we analyzed the performance loss as the number of possible rank assignments $A = 1, 2, \ldots, 100$ increased and ran an experiment for different amounts of ethical features $F = 5, 7, 10$ and ranks $R = 2, 3, 4$. All other elements were kept constant across these experiments. Namely, each problem contained the same possible initial situations $S = 20$ which presented two possible plans $go(left)$ and $go(right)$ (as in the case study) as its options, each of which was assigned the same number $F_{xp} = 2$ of randomly chosen ethical features, which were disjointed between the two plans, for each situation. And then, we generated a dataset of $O = 100$ random opinions from users who chose either the plan $go(left)$ or $go(right)$ in each situation.

So for instance, given $A = 50$ and $F = 10$, the experiment would: (i) add ethical features $danger(1), \ldots, danger(10)$ to the problem, (ii) assign two random features (such as $danger(4)$ and $danger(6)$) to the plan $go(left)$, and two other random features (such as $danger(1)$ and $danger(9)$) to $go(right)$ in each situation, and (iii) generate 50 random rank assignments mapping each of the 10 features to a rank between 1 and R, and (iv) run the parameter learning algorithm with the encoding described in the previous chapters.

It is important to note that the specific ethical features assigned to plans and choices of experts did not change the complexity of the parameter learning problem at hand, but rather the total number of ethical features.

Experiment 2 Regarding the second batch of experiments, we also took as a basis the problem described in the previous section but varied the number $O = 1, 2, \ldots, 1000$ of expert opinions in the dataset. We generated parameter-learning problems in the same fashion as in the previous batch of experiments. Briefly, we generated $S = 20$ situations, with two plans each ($go(left)$ and $go(right)$), a fixed number of ethical features $F = 5$ and assigned $F_{xp} = 2$ disjointed sets of ethical features to each plan. We set $R = 4$, allowing ranks between 1 and 4 and generated a dataset of O opinions from experts who chose either $go(left)$ or $go(right)$ in each situation randomly. For the rank assignments, we set $A = 10$ random possibilities across runs, because this variable was already tested in the previous set of experiments.

Methodology of evaluation

For each parameter learning problem tested in these experiments, we averaged the running time across 10 identical runs, increasing the number of rank assignments $A = 1, 2, \ldots, 100$ (in the first experiments), or the size of the dataset $O = 1, 2, \ldots, 1000$.

[1] https://github.com/martinjedwabny/lfi-ethical

As for the parameter learning algorithm, we used the implementation provided by Problog [Gutmann et al., 2011], which is publicly available[2].

Just like in the experiments of Section 5.3, these experiments were conducted on a 1,6 GHz Intel Core i5 CPU MacOS system with 8 GB 2133 MHz LPDDR3 RAM, using the Python `os.nice`[3] functionality, to ensure the maximum resources of the machine were assigned to the tests.

Results and analysis

Experiment 1 As we can see in Figure 6.3, there is a pronounced increase in parameter learning time when the number of rank assignments is increased. We display three separate subfigures a,b and c, each corresponding to a different number $F = 5, 7, 10$ of ethical features in the tested problems. In each of these subfigures, the Y axis denotes the running time of the parameter learning algorithm, while the X axis measures the number of possible rank assignments. Furthermore, each subfigure shows three series of measurements, each corresponding to a different maximum possible rank $R = 2, 3, 4$.

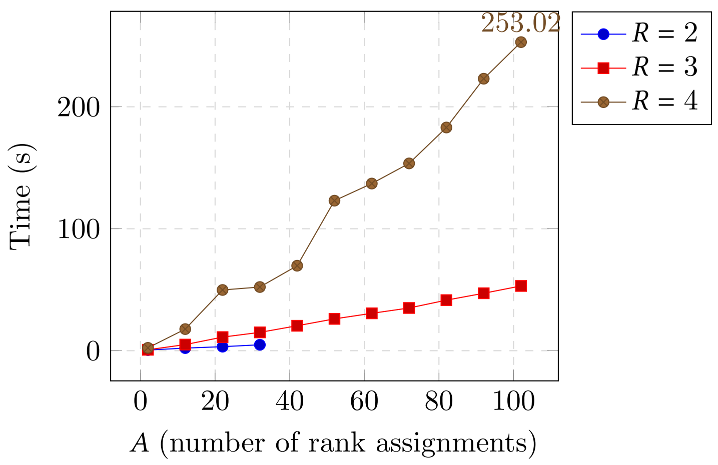
Please note that in the case of Figure 6.3(a) the measurements for $R = 2$ stop at $A = 32$ because there are only $2^5 = 32$ possible rank assignments when one counts with 2 possible ranks and $F = 5$ ethical features. This limitation is not encountered for the rest of the measurements.

At first glance, it seems that the trend of the curve is above linear and the running time increased considerably when considering a higher number of ethical features F or possible ranks R. The maximum running time (value of the Y coordinate) was always reached for $R = 4$ and was 250 seconds (4 minutes) for $F = 5$, 4428 seconds (1 hour, 13 minutes) for $F = 7$ and 11554 seconds (3 hours and 12 minutes) for $F = 10$.

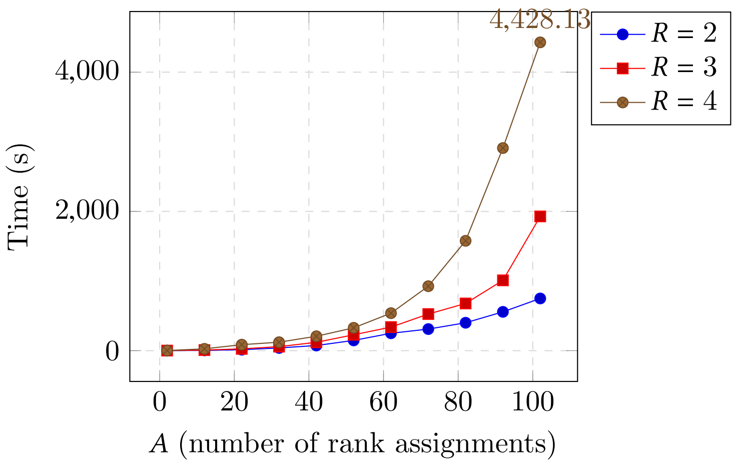
It is worth noting that one can tolerate a long running time for learning rank assignments, if we consider the learning phase to be executed offline, in order to later use its results in the context of decision-making or planning framework, such as the one presented in the last chapters. Moreover, the set of all the possible rank assignments, which correspond to the mappings of ethical features to ranks, can be very high (R^F to be exact). However, as we shall discuss in Section 6.6, the possible rank assignments can be greatly reduced by only considering a reasonable subset of all the possibilities, for instance, by introducing constraints.

Experiment 2 Then, regarding the second batch of experiments, we can see their results in Figure 6.4. Here, the Y axis denotes the running time of the parameter learning algorithm and the X axis measures the number of expert opinions, i.e: the size of the dataset. We observe a different trend than in the previous experiments. The running time was tested using different amounts of ethical features $F = 4, 6, 8, 10$, but across all of them, the running time increased very slowly. This in turn suggests, like in the

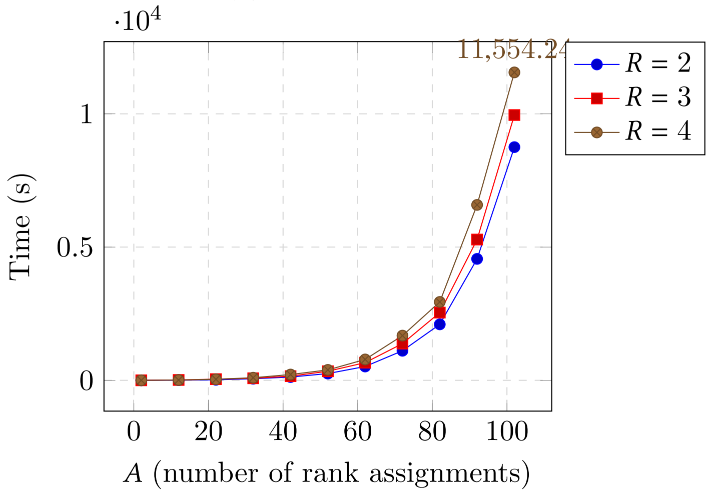
[2]https://github.com/ML-KULeuven/problog/tree/master/problog/learning
[3]https://docs.python.org/3/library/os.html



(a) $F = 5$



(b) $F = 7$



(c) $F = 10$

Figure 6.3: Parameter learning runtime according to the number of possible rank assignments A with varying number of ethical features F.

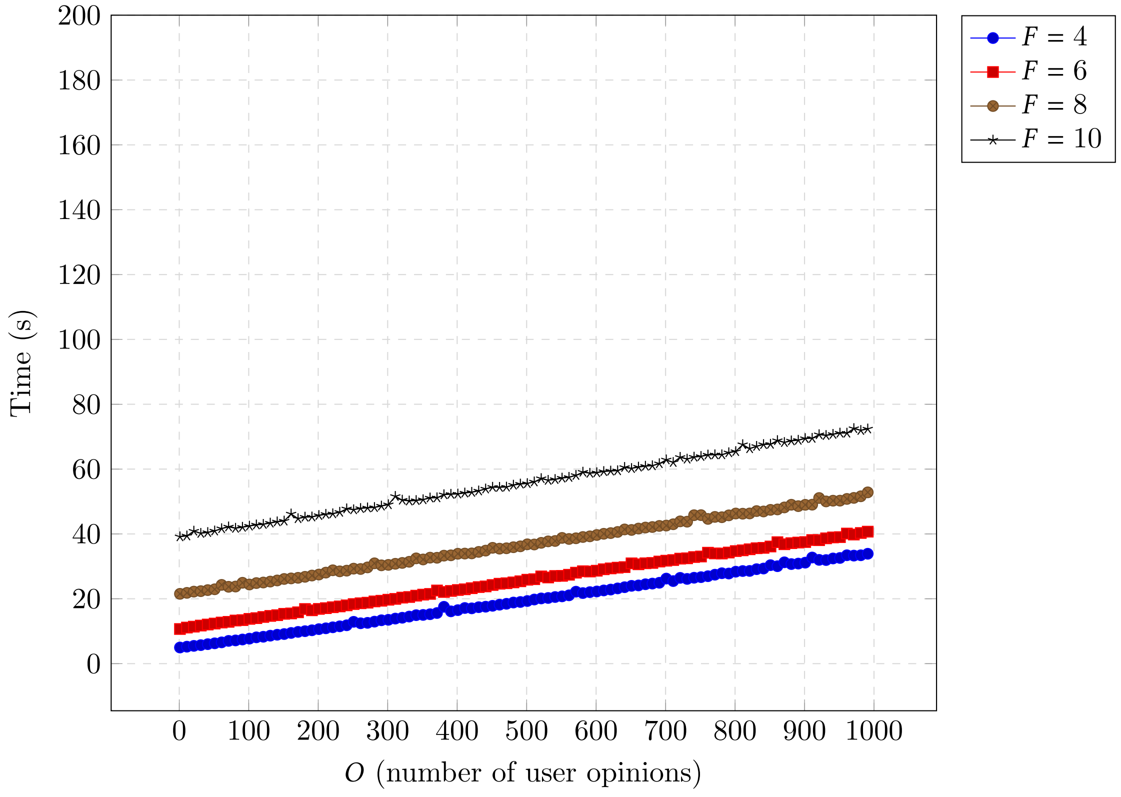


Figure 6.4: Parameter learning runtime according to the dataset size O for varying ethical features F.

previous batch of experiments, that the running time depends more on the number of ethical features and possible rank assignments, instead of the dataset or complexity of the situations, plans and which ethical features are assigned to each plan.

To answer the questions posed before:

Q1: The number of rank assignments impacts the parameter learning time greatly, the trend in this increase of running time appears to be above linear, and in addition, increasing the number of ethical features or possible ranks also had a noticeable impact, as the running time escalated more quickly in Figure 6.3 for higher F and R values.

Q2: The size of the dataset, given by the number of expert opinions, did not seem to affect the performance of the algorithm as much as any of the tested variables in the first batch of experiments. In fact, even when testing for large datasets, the running time increased very slowly. In conjunction with the first set of experiments, this suggests that the complexity of the task depends more on the total amount of rank assignments, ethical features and ranks than anything else.

6.5 Related work

Our work in this chapter described a method for learning ethical preferences from a corpus of elicited opinions, in the form of ranks, to be used in conjunction with the framework we described in the last chapters.

Previously, in Chapter 3, we described most of the well-known ethical decision-making and automated planning systems that utilize learning algorithms, namely bottom-up and hybrid approaches.

Bottom-up systems, by definition, focus on inferring which combinations of properties make a decision ethical, or more ethical than other alternatives, from a corpus of data. Previous literature in the context of decision-making has done this by learning numerical utilities for actions [Armstrong, 2015], learning to summarize the overall preferences of a large number of people through pairwise comparisons between alternatives [Noothigattu et al., 2018] using a voting-based mechanism, or learning to balance personal gain with showing good character to other agents through simulation [Hegde et al., 2020]. Regarding planning-based approaches, [Abel et al., 2016, Wu and Lin, 2018, Rodriguez-Soto et al., 2021] have shown how to infer which actions are unethical through the reward function of a Markov Decision Process [Puterman, 1990], by assigning penalties when agents execute actions that violate ethical behaviors. However, bottom-up approaches differ from hybrid systems, such as ours, in one main way: they do not model ethical principles or try to characterize what is ethical following some higher notion of rules of conduct that can determine what is right. Using the framework presented in the last chapters, we can indeed model ethical principles coming from well-known theories to represent rules of conduct, as shown in Chapter 4. In turn, we use a learning algorithm to infer which ethical features or principles (as they get reduced to features) are more important than others through their ethical ranks.

Another related work is that of the Moral Machine experiment [Awad et al., 2018]. There, the authors crowdsourced 40 million decisions regarding moral dilemmas in which an agent drives an autonomous vehicle and has to decide whether to keep going straight or turning, where both decisions lead to the death of different people and/or animals. Their work deals with learning ethical preferences and analyzing which ethical features are prioritized by different people according to their culture, gender, age, and other characteristics. They use a voting-based method to aggregate the crowdsourced dataset and learn which features (such as sparing more lives or sparing the young) are more important to certain kinds of users. As a result, they produce different preference scores over the ethical features of the problem at hand, which can essentially be seen as an order relation over these features. In comparison, our approach focuses on providing a combination of ranks for the features, which can be used as a whole to determine the best alternative in each situation. By doing this, our method can be used in the context of a framework such as the one described in the last chapters.

Turning to hybrid ethical approaches, some literature has used Inductive Logic Programming (ILP) [Muggleton, 1991] to infer logical rules to determine which decisions are ethical. Namely, in [Anderson et al., 2005a] the authors also present the W.D. system, which uses ILP to learn when one decision is ethically preferred to another with respect to different ethical properties based on prima facie duties [Ross, 1930] and how each decision conforms to (or violates) these properties. Then in [Anderson et al., 2005b, Anderson et al., 2006] they continued their work by developing the MedEthEx, which applies this same idea to medical cases. Also, in [Anderson and Anderson, 2018] the same authors presented the GenEth system, which applied this same mechanism to the autonomous driving domain. Concretely, they represent ethical principles as first-order logic-based rules which characterize when one action is ethically better than another with respect to how much they conform to or go against different ethical properties. If two cases of the dataset disagree, ILP systems such as these will fail in their inference, while our system is resistant to this kind of problem. Indeed, our method in this chapter leverages the power of probabilistic logic to handle noise in the event different cases of the dataset disagree on which choice is more ethical than another. In addition, our approach can be used to learn in the context of planning instead of only decision-making, by utilizing a framework such as the one described in the previous chapters.

Furthermore, [Dyoub et al., 2020] also uses ILP for ethical decision-making, although to learn which decisions are ethical or unethical, instead of determining which decisions are more ethical than others, as the previously discussed systems. Our framework differs from theirs in that we learn to assign ranks to ethical features and use them to compare decisions on ethical terms, instead of determining if decisions are ethical or not. Also, as mentioned before, our model differs from this research in that it can be used to learn in the context of planning with ethical elements, in addition to decision-making.

In summary, the system described in this chapter differs from previous ones in that: (i) it leverages learning algorithms to elicit preferences for planning problems (as opposed to only decision-making), (ii) it can handle noisy datasets in contrast with ILP-based approaches, by using a probabilistic encoding (Problog), and (iii) it learns rank assignments for ethical features instead of a set of logical rules that determine if a decision is ethical or not.

6.6 Discussion

In this chapter, we have developed a method for ethical preference elicitation which can handle noisy datasets and be used in conjunction with the model presented in the previous chapters. By reducing the problem of inferring ethical preferences to choosing ranks for ethical features using the parameter learning technique, we showcased how our model can be used as a hybrid ethical system. In addition, we provided a Problog-based implementation[4] of this part of our system, described a case study to exemplify the approach, and provided a preliminary analysis of some experimental results.

As mentioned in the Introduction, explainability is a highly desirable characteristic for automated agents in ethically-nuanced domains. One of the many advantages of using probabilistic logic programming is that by coupling logic rules with probabilistic annotations, the deductions it produces are transparent, i.e: humans can understand the reasoning of the machine and its decisions through its encoding. By using the parameter learning approach, we produce a rank assignment that can be combined with a Problog program to compare plans, in such a want that a user can understand and interpret. Thus, from an Explainable AI (XAI) perspective, the method described in this chapter is a contribution to *transparent* systems [Arrieta et al., 2020], by making it easy for the user to understand the reasoning process by using well-defined logic rules, and inferring a probability distribution over a set of rank assignments.

By leveraging the noise-handling capabilities of Problog [De Raedt et al., 2007] and parameter learning [Gutmann et al., 2011], our system can handle many different opinions from a dataset and aggregate them. This, in turn, shows the adaptability and generality of the probabilistic logic setting, which has been previously used to develop explainable AI systems in other subfields, such as decision-making [Van den Broeck et al., 2010] and recommender systems [Catherine and Cohen, 2016]. Moreover, the ability of handling noise in this way separates our approach from much previous work that uses inductive logic programming and cannot handle differing views on which the most ethical choice is in a particular situation.

The result we obtain from this procedure is a rank assignment with the highest probabilistic annotation. Admittedly, aggregating the opinions of the whole dataset does not guarantee that the resulting ranks will match all of the decisions for a majority of the users, but in turn, it means that the particular rank assignment matches the most amount of decisions if we combine all of the decisions of every user. As such, this result can be used as a first approximation of what the most pertinent rank assignment is in the domain at hand.

Regarding the elicitation of user/expert opinions, we chose to consider the setting in which they submit their opinion on which choice they considered the most ethical. One could ask why we chose this form of input instead of asking them which rank should be assigned to each feature in their opinion. First of all, in the presence of a large number of ranks and ethical features, it would be very hard for the users to provide precise answers that would lead the system to select the best plan in every case according to their views. And also, it would be difficult for the users to predict potentially dangerous uses for the system when an ethical feature is assigned a particular rank. Indeed, it would be very easy for users to choose approximate ranks for some features not knowing in which possible scenarios they could be misused.

[4]https://github.com/martinjedwabny/lfi-ethical

In the experimentation, we could analyze how the running time of the parameter learning algorithm degrades when the different sizes of the input, in terms of rank assignments and dataset, are increased. As mentioned, the system could handle up to $A = 100$ rank assignments in most cases for up to $F = 10$ ethical features and $R = 4$ possible ranks, and up to $O = 1000$ opinions, with very slow degradation as the size of the dataset increased. In turn, the system proved to be more sensitive to the number of possible rank assignments, ethical features and ranks, which was not surprising considering that adding ethical features and ranks made the Problog program slower and that the number of rank assignments matches the amount of probabilistic annotations to learn.

It is necessary to specify the rank assignments to compare as part of the input of the learning procedure, as considering every possible rank assignment leads to R^F probabilistic annotations to learn. Taking $F = 10$ and $R = 4$ would lead to more than 1 million probabilistic annotations, which would not be feasible using this method. As a reference, in the Moral Machine experiment [Awad et al., 2018], the authors consider a total of 9 ethical features to decide which people an autonomous vehicle should save, namely: sparing humans, inaction, sparing passengers (versus pedestrians), sparing more lives, sparing men (versus women), sparing the young (versus the elderly), sparing pedestrians who cross legally, sparing the fit, and sparing those with higher social status. In cases such as this, it could be the case that some features are more important than others, for instance, society could consider that sparing more lives is more important than sparing the fit. Thus, not every possible rank assignment has to be considered, as we can remove all those in which sparing the fit has a higher rank than sparing more lives. In this way, by introducing more constraints, the possible rank assignments may be reduced.

Future work In the future, we envision different ways of expanding the work presented in this chapter. First of all, it would be interesting to compare the performance of different parameter learning encodings for the inference of preferences in terms of computation time. In this sense, it would be beneficial to study different ways to reduce the total number of possible rank assignments, which would decrease the number of probabilities to infer through the parameter learning algorithm.

The approach we took in this chapter was to produce a probability distribution over the considered rank assignments. Then, we suggested picking the rank assignment with the highest probabilistic annotation. Yet, it would also be possible to calculate the probability (or error) of each rank assignment with respect to the Problog program and the dataset separately, instead of producing this distribution that depends on all of the rank assignments at the same time. We leave this task for future work.

Also in terms of efficiency, we would like to test out other machine-learning techniques to infer the best rank assignment. Essentially, each problem is composed of discrete values representing the ethical features of each plan and situation and the dataset is composed of opinions of users stating which actions, characterized by their set of ethical features, are to be preferred to others. As such, supervised machine-learning algorithms for preference learning [Mohri et al., 2018], a special type of classification, can be used to infer which sets of ethical features are more important than others, by adapting the algorithms to find the optimal rank assignments so that its ranks prioritize the right features according to the dataset.

In terms of experimentation, we need to (i) scale up the size of the experiments, (ii) compare the computation time of this approach to other machine learning methods, and (iii) provide complex test cases for ethical planning problems and make the agents plans be assessed by human observers.

Lastly, we would like to test the inference mechanism in conjunction with the ethical framework presented in the last chapters in a real-life scenario for an integral evaluation of our system.

7 Conclusion

As we explained in the Introduction, the mainstream employment of automated agents in various domains to aid people in daily tasks or operate without any human intervention has sparked serious concerns related to our trust in these systems. For this reason, past research has pointed out the necessity to build machines that can be ethically aligned to societal values and explain their decisions [Tolmeijer et al., 2020]. Thus, we set out to provide a framework that automated agents could utilize to represent multiple ethical principles and combine them through preferences, along with a mechanism to learn priorities between them. We formalized and explained the main concepts and background notions surrounding this thesis in Chapter 2. Chapter 3 described the state-of-the-art of machine ethics implementations and various overarching topics concerning ethics as a field. In Chapter 4, we introduced an extension of PDDL that assigns a set of ethical features to plans, demonstrated how it captures some well-known ethical principles and showed how these features can be combined using preferences. Then in Chapter 5, we focused on demonstrating that ethical problems modeled using our framework can be translated into utilities by using soft goals. We also developed two different implementations of this translation that allow state-of-the-art planners which are compatible with soft goals and actions costs, to be used in our framework, and provided several experiments to test the effectiveness of our approach. Lastly, in Chapter 6, we described and implemented a method to compute priorities between ethical values for our framework using probabilistic logic and the parameter learning approach. All of this work built upon and extended previous work published in [Jedwabny et al., 2021a] and [Jedwabny et al., 2021b].

This conclusion discusses how the work of this thesis addresses the research problem and questions described in the Introduction (Section 7.1), highlights the scope and impact of our contributions to the field of machine ethics (Section 7.2), and finishes by presenting some interesting future directions (Section 7.3).

7.1 Research questions

Previously in Chapter 1, we posed ourselves a set of questions to answer throughout this thesis. In what follows, we will go over how our research addressed these questions.

Question 1: How can we model and combine well-known ethical principles through preferences in a deterministic planning setting? We developed an extension of PDDL, a well-known deterministic planning language, composed of different ethical constructs in Chapter 4, that allows the modeling of various ethical principles and preferences between them. In doing so, we provided a transparent method to align machines with a given societal view of ethics.

Ethical principles were modeled through ethical features and rules, while the preferences were described with ethical ranked bases, which assign a level of priority to features. Ethical features represent properties with ethical meaning that are assigned to plans if certain conditions are met. These conditions are specified using ethical rules and depend on states and optionally on the execution of certain operators, according to whether they are defined via fluents or operators. Practically, ethical features can seem similar to fluents but, as discussed in Section 4.5, they are used to characterize plans instead of world states and it is useful to keep them separated from fluents for modularity. Then, ethical rules are used to assign ethical features to plans depending on fluents, actions and other ethical features.

Finally, the ranked bases can be used both to define priorities, and implicitly, to combine ethical principles by defining priorities between ethical features. This preference representation model makes it possible to translate our ethical planning problems to classical planning problems with utilities, as we saw in Chapter 5, and is compatible with learning techniques through the ethical ranked bases, as specified in Chapter 6.

Question 2: Can we provide an efficient way of computing ethically optimal plans using our model? In Chapter 5, we described and implemented a translation scheme that reduces ethical planning, as we defined it, to classical planning with utilities.

We demonstrated the correctness of this translation procedure by proving that if an ethically optimal plan exists, a planner that can handle utilities will eventually find it in the translated version of the problem. Two implementations of this idea were developed and made publicly available[1]. While the first implementation enables planners compatible with soft goals to solve ethical problems, the second compiles these soft goals into action costs, following the ideas of [Keyder and Geffner, 2009]. Thus, we demonstrated that state-of-the-art planners such as those presented in the International Planning Competitions (IPC) can be used to solve ethical planning problems modeled using our framework.

[1]https://github.com/martinjedwabny/pddl-ethical

Lastly, we conducted some experiments showing that indeed, these planners can be used along our model and translation procedures to calculate ethically optimal plans in a reasonable time when considering various medium-sized instances of some very hard problems from the IPCs.

Question 3: In which way can we elicit preferences from a corpus of data to determine priorities between ethical values inside our model? To answer the last section, we showed in Chapter 6 how we can use Problog and the parameter learning technique to learn priorities between ethical features in the form of ranks, as specified by our model in the previous chapters.

We introduced a Problog encoding of situations, plans, ethical features and ranks that, in conjunction with the parameter learning method, can be used to aggregate the choices of one or many data sources to determine the ranks, as defined in Chapter 4, of the ethical features. In this way, our framework can be used as a hybrid one, by both representing ethical principles and learning the priorities between them, which was required for our goal of developing agents ethically aligned to the values of society. Lastly, we implemented these ideas using state-of-the-art probabilistic logic programming libraries and discussed the results of some experiments showing the feasibility of our approach.

7.2 Scope and impact

Here, we will discuss the scope of the contributions of our work and its impact on the field of machine ethics in light of previous research. In order to do this, we will provide an overview of how this thesis compares and extends the current state-of-the-art.

To begin with, the first contribution of this thesis was the development of an extension of PDDL to represent the ethical elements of a problem and a mechanism to compare plans, introduced in Chapter 4. By itself, this framework constitutes an advancement and an interesting alternative to past approaches for top-down ethical reasoning models in several different accounts:

1. It contributes an alternative view of ‘machine ethics’ as a matter of preferences and not absolute *right* and *wrong*, as many previous models do. Most past literature in ethical reasoning for automated planning and decision-making we reviewed in Chapter 3 representated ethical theories by classifying plans as either right or wrong [Berreby et al., 2015, Ganascia, 2007, Bourgne et al., 2021, Arkin et al., 2011, Govindarajulu and Bringsjord, 2017, Lindner et al., 2019]. In this sense, our work can be seen as a refinement or an alternative to past approaches, because instead of classifying plans as one of these two classes, by considering preferences we allow a plurality of classes of ethical correctness, ranging from more to less desirable. In addition, by working with preferences, we can help decide between multiple

plans even when none is perfectly ethical, which as we have discussed in Chapter 4, can be considered an advantage in certain cases, such as when not doing anything is considered unethical.

2. The second way in which our treatment of ethics departs from past literature is the way in which we separate it from the rest of the domain elements, namely the fluents, actions, states and goals, which we call the operational part of a planning problem. We argue that by keeping the fluents and ethical features separated from one another, we improve the modularity between the operational and ethical aspects of the framework. This is useful for two reasons: (i) the problem description remains unaffected and we can add ethical terms (features, rules and preferences) to an existing problem seamlessly, which is useful if ethical considerations change according to societal views, but the planning problem remains the same, and (ii) we can reuse the ethical features and ranks in many different problems by adapting the ethical rules to each of them, therefore allowing the reuse of ethical preferences and principles. This distinction is not the norm in past literature, as we saw in Chapter 3, except for belief-desire-intention (BDI) models. BDI approaches [Arkin et al., 2009, Dennis et al., 2016, Cointe et al., 2016] typically embed an extra layer within the agent. They use precomputed plans obtained via external planning modules and compare them on ethical terms. Our framework differs from BDI approaches by using ethical features and preferences when searching for the plan itself, and not after the planning phase, which allows for specific optimization according to ethical considerations.

3. Third, we provide a new adaptation of several ethical theories and concepts using preferences and a seamless way to combine them. In Chapter 3, we demonstrate how our modelization of ethics can model consequentialism, deontological ethics, virtue ethics, the do-no-harm principle and the doctrine of double effect (with some reasonable modifications in the latter case). Although all of these theories had been researched in the past in the context of machine ethics, the separation of their elements via ethical features, rules and preferences forced us to rethink them to fit our approach, so that we could maintain modularity between the ethical and operational aspects of a problem. A preference-based model of ethics by itself offers a useful alternative perspective of machine ethics, and more importantly, this pluralist view of ethics permits to take into account the opinions of various information sources and combining them, thus conciliating the agent's behavior with the numerous viewpoints of society on ethics.

Then, in Chapter 5 we present an implementation and experimentation of our framework described in the previous chapter, which further shows the usefulness of our approach:

1. We contributed an implementation of the PDDL extension described in Chapter 4 in the actual programming language, that anyone can

use as a basis to investigate ethical reasoning in planning domains and made it publicly available. From the standpoint of knowledge representation for automated planning, our formal model offers an extension of the PDDL language, one of the most widely used representations for classical planning, which is compatible with the large corpus of research on PDDL-based representations (such as STRIPS [Fikes and Nilsson, 1971], SAS+ [Bäckström and Nebel, 1995] and PDDL itself [Fox and Long, 2003]). Much of the previous research reviewed in Chapter 3 represented planning using logic-based frameworks, BDI frameworks, or more specific models, but few used PDDL-based representations. In comparison, by using PDDL, our approach can be is compatible with much previous research using this language.

2. It was shown both in theory and practice how utility-based planners can be used to solve these ethically-extended PDDL problems with two different translation routines, which we also made publicly available. By leveraging all the past research regarding PDDL-based planning from the IPCs, our framework can directly use existing state-of-the-art planners to find ethically optimal plans, which is a great advantage compared to existing research, especially those approaches relying on logic for planning as a deduction, in terms of computation speed. We demonstrated the practicality of our approach by solving ethical planning problems using our translation routines and some state-of-the-art planners. Also, we analyzed some practical results regarding the number of ethical constructs these planners can handle when they are added to some medium-sized instances of very hard problems from the International Planning Competitions.

3. Some authors [Tolmeijer et al., 2020] have pointed out the necessity of developing example planning domains to test ethical theories and their implementations. Our work contributed to this in two ways. First, by providing an extension of PDDL to capture ethical concepts, we provided a basis to easily create benchmarks for planning domains. And second, we developed a PDDL domain based on our overarching example (Autonomous driver) to test various encodings of ethical theories.

Finally, Chapter 6 shows the adaptability of the preference-based mechanism to be used as a hybrid framework:

1. We showed how the rank-based preferences of the model presented in the previous two chapters can be inferred using parameter learning. The simplicity of the numerical ranks made it straightforward to provide an encoding of the learning problem in such a way that we can use a corpus of preferences between plans to retrieve the best-fitting rank assignment. Compared to previous research, our approach relies on ranks to determine which option is more ethical, instead of comparing them using their ethical features directly, as in the Moral Machine experiment [Awad et al., 2018]. By doing this, the resulting

rank assignments are compatible with the framework presented in the previous two chapters.

2. Then, we presented one case study showcasing the ranks learned by our approach to show the applicability of our hybrid framework.

3. Finally, we performed some experiments and analyzed their results to demonstrate the practicality and computational cost of using our approach.

All in all, the development of our hybrid ethical framework is a novel work on ethical theory representation, which takes advantage of diverse techniques from various AI fields. We provided a framework that is largely compatible with many widely-used representations and algorithms and a useful alternative view on machine ethics as a preference-based problem.

7.3 Perspectives and future work

In this final section, we will describe different areas of extension and improvement we left out for future work.

Starting with Chapter 4:

- As an initial step into modeling ethics as preferences, we chose the classical planning setting to develop our research. However, much current research in autonomous agents deals with domains containing uncertainty or other complex characteristics, and logically, ethical reasoning should be adapted for these domains as well. As such, in the future, we plan to consider other planning models that allow multiple agents, exogenous actions, or uncertainty (belief-state or probabilistic planning) would be a logical next step to further show the usefulness of our ideas.

- Regarding ethical features, in the current definitions of our framework, we considered the ethical features assigned to a plan to be a set, but there is no reason why they cannot contain repetitions. Considering the set of features assigned to a plan a multiset could help to model cases in which activating an ethical feature multiple times makes a difference. For instance, if an agent could activate an ethical feature many times in a problem, it could be useful to add this ethical feature repeatedly if this fact would impact the ethical desirability of the plan compared to others. The usefulness of such an extension should, of course, be assessed in terms of ethical theories and potential dilemmas.

- Also, in the interest of representation compactness, we could introduce a way of representing many ethical features with the same object (or rather, aggregated feature). Adding an extra value number in the ethical rank can also help refine the method of comparing plans. In other words, we can change the ethical ranked based $b(e) = \langle t, r \rangle$ in

Definition 4.10, to the more precise $b(e) = \langle t, r, n \rangle$, where $t \in \{+, -\}$, $r \in \mathbb{N}$ is the ethical rank and $n \in \mathbb{N}$. This value n indicating that the feature e is worth n features at the rank r. For instance, if the agent could save X number of lives by executing an action, instead of having five ethical features $save(1), \dots, save(5)$ with a specific rank r, we could have a single feature $saved5()$ with rank r and $n = 5$.

- Then, concerning ethical rules, instead of defining the activation conditions via sets of features and one action, we could extend our work by providing a more general language to express preferences. For example, by activating ethical rules when a certain condition holds for every state, or a certain number of states, in the style of PDDL3 preferences [Gerevini et al., 2009]. This could be useful for defining conditions that the agent should always preserve to remain ethically correct.

Turning to the framework's implementation in Chapter 5:

- An interesting challenge regarding the implementation of our framework would be providing a planner specific to our ethical planning problems, that does not need the translation routines described in Chapter 5, but rather parse and solve the problems directly. This would allow the development of an algorithm specifically tuned for ethical planning problems that might achieve better performance than the tested state-of-the-art planners.
- Then, we would like to find complexity results for ethical planning problems. By doing this, we could determine whether or not providing an algorithm specific to our ethical planning problems could be much more efficient than translating the problems to classical planning with utilities and using general-purpose state-of-the-art planners. We have shown that they can be translated into classical planning problems with utilities, which are well-studied regarding temporal and spatial complexity. However, it could be useful to characterize the computational cost in terms of the ethical features and rules introduced in a problem.
- Concerning the experimentation, we have left for future work the testing of more complex ethical rules, e.g: using parameters, complex activation conditions and a large number of activated and deactivated ethical features.

And finally, regarding Chapter 6:

- We would like to test the computational cost of the learning with different logical encodings to improve its performance. More precisely, in the mentioned chapter, we described one encoding that turned the task of finding the optimal ranks of an ethical problem into a parameter learning problem. Nevertheless, different such encodings are possible, and we would like to compare the possibilities.

- Then, instead of producing a probability distribution over the considered rank assignments and choosing the one with the highest probabilistic annotation, we would like to implement an algorithm that calculates the probability (or error) of each rank assignment with respect to the Problog program and the dataset, and determines the best rank assignment without computing this whole distribution. This task would require a modification of the parameter learning implementation and the Problog library, but will be much faster than our current approach for finding the best rank assignment.

- Also, we would like to try finding the best rank assignment through other machine-learning algorithms. As mentioned in the Discussion of the chapter, supervised machine-learning algorithms for preference learning [Mohri et al., 2018] may be utilized to learn the optimal rank assignments so that the assigned ranks prioritize the right features according to the dataset.

From a global point of view, we would like to test our hybrid ethical framework as a whole. That is, testing the performance of the system in a task that requires learning the ethical ranks and applying them in a planning setting simultaneously. Although the experimental results of Chapter 6 seemed promising, we would like to perform additional tests to check the quality of our approach's answers. Namely, testing if people agree on the learned behavior of an agent that uses ethical features and the ranks obtained through our proposed parameter learning approach. Although the system is transparent in the sense that it decides based on logic rules and probabilistic annotations, we could also test if people can understand the reasoning process of the agent. Lastly, by seeing if the users agree and understand the decisions, we would confirm whether our approach is effective and/or easily explainable.

A
Appendix: Computational complexity

This Appendix chapter describes some notions related to computational complexity that were left out of the main chapters of this thesis for being of general knowledge for the computer science community but were useful to include for reference.

Complexity classes indicate the inherent difficulty of a computational problem. Time complexity describes the number of elementary (fixed time) operations performed by an algorithm until completion given the size of its input. Similarly, space complexity measures the amount of space in computer memory used by an algorithm, including that reserved to store the input, the result and all the intermediate computation values.

Because the exact number of operations may vary between different inputs of the same size, complexity analysis is described in terms of the worst case, exact case, or average case for a given size. Here we will present complexity classes using the worst-case analysis, but they all apply similarly to the other cases. Time and space worst-case complexities are described using the O-notation. Briefly, given a problem with size of input n and time/space complexity represented by the function $f(n)$, we say that the complexity is in $O(g(n))$, also written $f(n) \in O(g(n))$ if there exist constants $c, n_o \in \mathbb{N}$ such that for all $n \in \mathbb{N}_{\geq n_0}$ it holds that $f(n) \leq c * g(n)$. In other words, this means that $f(n)$ grows no faster than $g(n)$. For more information about complexity theory, we refer the reader to [Papadimitriou, 2003].

A problem is in the time/space complexity class:

- **Constant**: if the time/space required for solving it does not depend on the size of the input. It is denoted $O(1)$ in big-O notation.

- **Linear**: if a deterministic Turing machine can solve it in time/space that increases linearly with the input size. It is denoted $O(n)$ in big-O notation, with n being the size of the input.

- **Polynomial** (PTime/PSpace): if a deterministic Turing machine can solve it in polynomial time/space for a given input size. It is denoted $O(n^c)$ in big-O notation, with n being the size of the input and c a constant.

- **NP**: if a non-deterministic Turing machine can solve it running in polynomial time/space for a given input size. Equivalently, a problem belongs to this class if checking whether a possible answer is valid can be solved in polynomial time with a deterministic Turing machine.

- **Exponential** (EXPTIME/EXPSPACE): if a deterministic Turing machine can solve it in simple exponential time/space: $O(2^{p(n)})$ where $p(n)$ is a polynomial function and n the size of the input.

Moreover, we say that a problem P_A is **hard** for a complexity class C if any instance of another problem from C can be reduced to an instance of P_A through an appropriate reduction (one that does not have a superior complexity). Also, a problem P_A is **complete** for a complexity class C if it belongs to C and every problem in C can be reduced to P_A using an appropriate reduction.

B
Appendix: PDDL code

In this appendix, we list the PDDL code used for our examples throughout this thesis. The code listed here is mostly written using the syntax and extensions of PDDL2.1. We remind the reader that the formalization presented in Chapter 2 is very similar to actual PDDL2.1, but with some reasonable simplifications. For more information about PDDL, we refer the reader to [Fox and Long, 2003].

Normally, PDDL files are divided into 'domain' and 'problem' files. The domain file serves to define the extensions (called requirements) of PDDL that are allowed (such as object types, negative preconditions, quantification in preconditions and effects, conditional effects and many others), constants that can be used in actions, the fluents of the domain, the actions, and the types of objects. Then in the problem file, the rest of the objects (constants that cannot be used in actions), the initial state and the goal state are described. We chose to use some extensions in this PDDL code that are not present in the formalization we described for reasons related to modeling problems concisely and the efficiency of planners. For instance, we used typing to specify which objects can substitute the variables in the operators.

B.1 Autonomous driver example original PDDL code

The following code refers to Example 4.1 and 4.2 in Chapter 4.

B.1.1 Domain file

```
(define (domain crash-d)
 (:requirements :strips :typing :equality
   ↪ :negative-preconditions :conditional-effects)

 (:types
  car xPos yPos direction - object
 )

 (:constants
  agent - car
  left straight right - direction
 )

```

```
(:predicates
 (hasPos ?C - car ?X1 - xPos ?Y1 - yPos)
 (hasDir ?C - car ?D - direction)
 (nextX ?D - direction ?X1 - xPos ?X2 - xPos)
 (nextY ?Y1 - yPos ?Y2 - yPos)
 (hasCrashed ?C1 - car)
 (hasBumped ?C1 - car)
 (updated)
)

(:action setDir
 :parameters (?D1 - direction)
 :precondition (updated)
 :effect (and
  (not (hasDir agent left))
  (not (hasDir agent straight))
  (not (hasDir agent right))
  (hasDir agent ?D1))
)

(:action setStop
 :parameters ()
 :precondition (updated)
 :effect (and
  (not (hasDir agent left))
  (not (hasDir agent straight))
  (not (hasDir agent right)))
)

(:action update
 :parameters ()
 :precondition (not (updated))
 :effect (and
  (updated)
  (forall
   (?C1 - car ?C2 - car ?Y1 - yPos ?X1 - xPos)
   (when
    (and
     (not (= ?C1 ?C2))
     (not (= ?C1 agent))
     (not (= ?C2 agent))
     (not (hasCrashed ?C1))
     (hasPos ?C1 ?X1 ?Y1)
     (hasPos ?C2 ?X1 ?Y1))
    (and
     (hasCrashed ?C1)
     (hasCrashed ?C2)
    )))
  (forall
   (?C1 - car ?Y1 - yPos ?X1 - xPos)
   (when
    (and
     (not (= ?C1 agent))
     (hasPos agent ?X1 ?Y1)
     (hasPos ?C1 ?X1 ?Y1))
    (and
     (hasBumped agent)
     (hasBumped ?C1)
```

```
      (not (hasDir ?C1 left))
      (not (hasDir ?C1 straight))
      (not (hasDir ?C1 right))
     ))))
 )

 (:action go
  :parameters ()
  :precondition (updated)
  :effect (and
   (not (updated))
   (forall
    (?C1 - car ?D1 - direction ?Y1 - yPos ?Y2 - yPos ?X1 -
   ↪ xPos ?X2 - xPos)
    (when
     (and
      (not (hasCrashed ?C1))
      (hasPos ?C1 ?X1 ?Y1)
      (hasDir ?C1 ?D1)
      (nextX ?D1 ?X1 ?X2)
      (nextY ?Y1 ?Y2))
     (and
      (not (hasPos ?C1 ?X1 ?Y1))
      (hasPos ?C1 ?X2 ?Y2))))
  )
 )
)
```

Listing B.1: Autonomous driver PDDL domain code.

B.1.2 Problem file

```
(define (problem crash-p-01)
 (:domain crash-d)
 (:objects
  c1 c2 - car
  x1 x2 - xPos
  y1 y2 y3 y4 - yPos
 )
 (:init
  (updated)

  (hasPos agent x1 y1)
  (hasPos c1 x2 y1)
  (hasPos c2 x2 y3)

  (hasDir agent straight)
  (hasDir c1 straight)

  (nextX straight x1 x1)
  (nextX straight x2 x2)
  (nextX right x1 x2)
  (nextX right x2 x2)
  (nextX left x1 x1)
  (nextX left x2 x1)

  (nextY y1 y2)
  (nextY y2 y3)
  (nextY y3 y4)
  (nextY y4 y4)
 )

 (:goal
  (and
   (updated)
   (hasPos agent x2 y4)
   (not (hasCrashed agent)))
 )
)
```

Listing B.2: Autonomous driver PDDL problem code.

B.2 Autonomous driver example PDDL code with ethical constructs

The following code shows how the previous one can be extended with the ethical features, rules and ranked bases described in Chapter 4.

B.2.1 Domain file

```
(define (domain crash-d)
 (:requirements :strips :typing :equality
   ↪ :negative-preconditions :conditional-effects :ethical)

 (:types
  car xPos yPos direction gravity - object
 )

 (:constants
  agent - car
  left straight right - direction
  low high - gravity
 )

 (:predicates
  (hasPos ?C - car ?X1 - xPos ?Y1 - yPos)
  (hasDir ?C - car ?D - direction)
  (nextX ?D - direction ?X1 - xPos ?X2 - xPos)
  (nextY ?Y1 - yPos ?Y2 - yPos)
  (hasCrashed ?C1 - car)
  (hasBumped ?C1 - car)
  (updated)
 )

 (:action setDir
  :parameters (?D1 - direction)
  :precondition (updated)
  :effect (and
   (not (hasDir agent left))
   (not (hasDir agent straight))
   (not (hasDir agent right))
   (hasDir agent ?D1))
 )

 (:action setStop
  :parameters ()
  :precondition (updated)
  :effect (and
   (not (hasDir agent left))
   (not (hasDir agent straight))
   (not (hasDir agent right)))
 )

 (:action update
  :parameters ()
  :precondition (not (updated))
  :effect (and
   (updated)
   (forall
    (?C1 - car ?C2 - car ?Y1 - yPos ?X1 - xPos)
    (when
     (and
      (not (= ?C1 ?C2))
      (not (= ?C1 agent))
      (not (= ?C2 agent))
      (not (hasCrashed ?C1))
      (hasPos ?C1 ?X1 ?Y1)
      (hasPos ?C2 ?X1 ?Y1))
     (and
      (hasCrashed ?C1)
      (hasCrashed ?C2)
     )))
   (forall
    (?C1 - car ?Y1 - yPos ?X1 - xPos)
    (when
     (and
      (not (= ?C1 agent))
      (hasPos agent ?X1 ?Y1)
      (hasPos ?C1 ?X1 ?Y1))
     (and
      (hasBumped agent)
      (hasBumped ?C1)
      (not (hasDir ?C1 left))
      (not (hasDir ?C1 straight))
      (not (hasDir ?C1 right))
     ))))
 )

 (:action go
  :parameters ()
  :precondition (updated)
  :effect (and
   (not (updated))
   (forall
    (?C1 - car ?D1 - direction ?Y1 - yPos ?Y2 - yPos ?X1 -
   ↪ xPos ?X2 - xPos)
    (when
     (and
      (not (hasCrashed ?C1))
      (hasPos ?C1 ?X1 ?Y1)
      (hasDir ?C1 ?D1)
      (nextX ?D1 ?X1 ?X2)
      (nextY ?Y1 ?Y2))
     (and
      (not (hasPos ?C1 ?X1 ?Y1))
      (hasPos ?C1 ?X2 ?Y2))))
 )
 )

 (:ethical-features
  (danger ?C1 - car ?G1 - gravity)
  (damageRail ?C1 - car)
  (responsibleAgent)
 )

 (:ethical-rank
  :feature
  (damageRail agent)
  :type -
  :rank 1
 )
 (:ethical-rank
  :feature
  (damageRail c1)
  :type -
  :rank 1
 )
 (:ethical-rank
  :feature
  (damageRail c2)
  :type -
  :rank 1
 )
 (:ethical-rank
  :feature
  (danger agent low)
  :type -
  :rank 2
 )
 (:ethical-rank
  :feature
  (danger c1 low)
  :type -
  :rank 2
 )
 (:ethical-rank
  :feature
  (danger c2 low)
  :type -
  :rank 2
 )
 (:ethical-rank
  :feature
  (danger agent high)
  :type -
  :rank 4
 )
 (:ethical-rank
  :feature
  (danger c1 high)
  :type -
  :rank 3
 )
 (:ethical-rank
  :feature
  (danger c2 high)
  :type -
  :rank 3
 )
 (:ethical-rank
  :feature
  (responsibleAgent)
  :type -
  :rank 4
 )

 (:ethical-rule crashRule
  :parameters (?C1 - car)
  :precondition (hasCrashed ?C1)
  :activation null
  :features (danger ?C1 high)
 )
 (:ethical-rule bumpRule
  :parameters (?C1 - car)
  :precondition (hasBumped ?C1)
  :activation null
  :features (danger ?C1 low)
 )
 (:ethical-rule responsibleCrashRule
  :parameters ()
  :precondition (hasCrashed agent)
  :activation null
  :features (responsibleAgent)
 )
 (:ethical-rule responsibleBumpRule
  :parameters ()
  :precondition (hasBumped agent)
  :activation null
  :features (responsibleAgent)
 )
 (:ethical-rule railLeftRule
  :parameters (?C1 - car ?X1 - xPos ?Y1 - yPos)
  :precondition (and
   (hasPos ?C1 ?X1 ?Y1)
   (hasDir ?C1 left)
   (nextX left ?X1 ?X1))
  :activation (go())
  :features (damageRail ?C1)
 )
 (:ethical-rule railRightRule
  :parameters (?C1 - car ?X1 - xPos ?Y1 - yPos)
  :precondition (and
   (hasPos ?C1 ?X1 ?Y1)
   (hasDir ?C1 right)
   (nextX right ?X1 ?X1))
  :activation (go())
  :features (damageRail ?C1)
 )
)
```

Listing B.3: Autonomous driver PDDL domain code.

B.2.2 Problem file

```
(define (problem crash-p-01)
 (:domain crash-d)
 (:objects
  c1 c2 - car
  x1 x2 - xPos
  y1 y2 y3 y4 - yPos
 )
 (:init
  (updated)

  (hasPos agent x1 y1)
  (hasPos c1 x2 y1)
  (hasPos c2 x2 y3)

  (hasDir agent straight)
  (hasDir c1 straight)

  (nextX straight x1 x1)
  (nextX straight x2 x2)
  (nextX right x1 x2)
  (nextX right x2 x2)
  (nextX left x1 x1)
  (nextX left x2 x1)

  (nextY y1 y2)
  (nextY y2 y3)
  (nextY y3 y4)
  (nextY y4 y4)
 )

 (:goal
  (and
   (updated)
   (hasPos agent x2 y4)
   (not (hasCrashed agent)))
 )
)
```

Listing B.4: Autonomous driver PDDL problem code.

B.3 Autonomous driver example translated PDDL code with soft goals

The following code is the resulting PDDL description of applying the transformation routine to PDDL code with soft goals of Chapter 5.2.

B.3.1 Domain file

```
(define (domain crash-d_GEN)
 (:requirements :preferences :strips :typing :equality
    ↪ :negative-preconditions :conditional-effects)

 (:types
  car xpos ypos direction gravity - object
```

```
)

(:constants
 agent - car
 left straight right - direction
 low high - gravity
)

(:predicates
 (haspos ?c - car ?x1 - xpos ?y1 - ypos)
 (hasdir ?c - car ?d - direction)
 (nextx ?d - direction ?x1 - xpos ?x2 - xpos)
 (nexty ?y1 - ypos ?y2 - ypos)
 (hascrashed ?c1 - car)
 (hasbumped ?c1 - car)
 (updated)
 (check)
 (normal-mode)
 (final-mode)
 (danger ?c1 - car ?g1 - gravity)
 (damagerail ?c1 - car)
 (responsibleagent)
)

(:action setdir
 :parameters (?d1 - direction)
 :precondition (and
  (normal-mode)
  (check)
  (updated)
 )
 :effect (and
  (not (check))
  (not (hasdir agent left))
  (not (hasdir agent straight))
  (not (hasdir agent right))
  (hasdir agent ?d1)
 )

)

(:action setstop
 :parameters ()
 :precondition (and
  (normal-mode)
  (check)
  (updated)
 )
 :effect (and
  (not (check))
  (not (hasdir agent left))
  (not (hasdir agent straight))
  (not (hasdir agent right))
 )

)

(:action update
 :parameters ()
 :precondition (and
  (normal-mode)
  (check)
  (not (updated))
 )
 :effect (and
  (not (check))
  (updated)
  (forall
   (?c1 - car ?c2 - car ?y1 - ypos ?x1 - xpos)
   (when
    (and
     (not (= ?c1 ?c2))
     (not (= ?c1 agent))
     (not (= ?c2 agent))
     (not (hascrashed ?c1))
     (haspos ?c1 ?x1 ?y1)
     (haspos ?c2 ?x1 ?y1)
    )
    (and
     (hascrashed ?c1)
     (hascrashed ?c2)
    )
   ))
  (forall
   (?c1 - car ?y1 - ypos ?x1 - xpos)
   (when
    (and
     (not (= ?c1 agent))
     (haspos agent ?x1 ?y1)
     (haspos ?c1 ?x1 ?y1)
    )
    (and
     (hasbumped agent)
     (hasbumped ?c1)
     (not (hasdir ?c1 left))
     (not (hasdir ?c1 straight))
     (not (hasdir ?c1 right))
    )
   ))
 )

)

(:action go
 :parameters ()
 :precondition (and
  (normal-mode)
  (check)
  (updated)
 )
 :effect (and
  (not (check))
  (not (updated))
  (forall
   (?c1 - car ?d1 - direction ?y1 - ypos ?y2 - ypos ?x1 -
   ↪ xpos ?x2 - xpos)
```

```
    (when
     (and
      (not (hascrashed ?c1))
      (haspos ?c1 ?x1 ?y1)
      (hasdir ?c1 ?d1)
      (nextx ?d1 ?x1 ?x2)
      (nexty ?y1 ?y2)
     )
     (and
      (not (haspos ?c1 ?x1 ?y1))
      (haspos ?c1 ?x2 ?y2)
     )
    ))
   (forall
    (?c1 - car ?x1 - xpos ?y1 - ypos)
    (when
     (and
      (haspos ?c1 ?x1 ?y1)
      (hasdir ?c1 left)
      (nextx left ?x1 ?x1)
     )
     (damagerail ?c1)))
   (forall
    (?c1 - car ?x1 - xpos ?y1 - ypos)
    (when
     (and
      (haspos ?c1 ?x1 ?y1)
      (hasdir ?c1 right)
      (nextx right ?x1 ?x1)
     )
     (damagerail ?c1)))
  )

 )

 (:action checkOp
  :parameters ()
  :precondition (not (check))
  :effect (and
   (check)
   (forall
    (?c1 - car)
    (when
     (hascrashed ?c1)
     (danger ?c1 high)))
   (forall
    (?c1 - car)
    (when
     (hasbumped ?c1)
     (danger ?c1 low)))
   (when
    (hascrashed agent)
    (responsibleagent))
   (when
    (hasbumped agent)
    (responsibleagent))
  )

 )

 (:action final-mode-start
  :parameters ()
  :precondition (and
   (check)
   (normal-mode)
   (not (final-mode))
  )
  :effect (and
   (not (normal-mode))
   (final-mode)
  )

 )

)
```

Listing B.5: Autonomous driver PDDL domain code with soft goals.

B.3.2 Problem file

```
(define (problem crash-p-01_GEN)
 (:domain crash-d_GEN)
 (:objects
  c1 c2 - car
  x1 x2 - xpos
  y1 y2 y3 y4 - ypos
 )

 (:init
  (updated)
  (haspos agent x1 y1)
  (haspos c1 x2 y1)
  (haspos c2 x2 y3)
  (hasdir agent straight)
  (hasdir c1 straight)
  (nextx straight x1 x1)
  (nextx straight x2 x2)
  (nextx right x1 x2)
  (nextx right x2 x2)
  (nextx left x1 x1)
  (nextx left x2 x1)
  (nexty y1 y2)
  (nexty y2 y3)
  (nexty y3 y4)
  (nexty y4 y4)
  (normal-mode)
 )

 (:goal
  (and
   (updated)
   (haspos agent x2 y4)
   (not (hascrashed agent))
   (check)
   (final-mode)
   (preference p_damagerail-agent
    (not (damagerail agent)))
   (preference p_damagerail-c1
    (not (damagerail c1)))
   (preference p_damagerail-c2
    (not (damagerail c2)))
   (preference p_danger-agent-low
    (not (danger agent low)))
   (preference p_danger-c1-low
    (not (danger c1 low)))
   (preference p_danger-c2-low
    (not (danger c2 low)))
   (preference p_danger-agent-high
    (not (danger agent high)))
   (preference p_danger-c1-high
    (not (danger c1 high)))
   (preference p_danger-c2-high
    (not (danger c2 high)))
   (preference p_responsibleagent
    (not (responsibleagent)))
  )
 )

 (:metric minimize
  (+
   (* (is-violated p_damagerail-agent) 1)
   (* (is-violated p_damagerail-c1) 1)
   (* (is-violated p_damagerail-c2) 1)
   (* (is-violated p_danger-agent-low) 4)
   (* (is-violated p_danger-c1-low) 4)
   (* (is-violated p_danger-c2-low) 4)
   (* (is-violated p_danger-agent-high) 48)
   (* (is-violated p_danger-c1-high) 16)
   (* (is-violated p_danger-c2-high) 16)
   (* (is-violated p_responsibleagent) 48)
  )
 )
)
```

Listing B.6: Autonomous driver PDDL problem code with soft goals.

B.4 Autonomous driver example translated PDDL code with action costs

The following code is the resulting PDDL description of applying the transformation routine to PDDL code with action costs of Chapter 5.2.

B.4.1 Domain file

```
(define (domain crash-d_GEN)
 (:requirements :action-costs :strips :typing :equality
    ↪ :negative-preconditions :conditional-effects)

 (:types
  car xpos ypos direction gravity - object
 )

 (:constants
  agent c1 c2 - car
  left straight right - direction
  low high - gravity
  x1 x2 - xpos
  y1 y2 y3 y4 - ypos
 )

 (:functions
  (total-cost)
 )

 (:predicates
  (haspos ?c - car ?x1 - xpos ?y1 - ypos)
  (hasdir ?c - car ?d - direction)
  (nextx ?d - direction ?x1 - xpos ?x2 - xpos)
  (nexty ?y1 - ypos ?y2 - ypos)
  (hascrashed ?c1 - car)
  (hasbumped ?c1 - car)
  (updated)
  (check)
  (danger ?c1 - car ?g1 - gravity)
  (damagerail ?c1 - car)
  (responsibleagent)
  (final-mode)
  (final-mode-check-damagerail-agent)
  (final-mode-check-damagerail-c1)
  (final-mode-check-damagerail-c2)
  (final-mode-check-danger-agent-low)
  (final-mode-check-danger-c1-low)
  (final-mode-check-danger-c2-low)
  (final-mode-check-danger-agent-high)
  (final-mode-check-danger-c1-high)
  (final-mode-check-danger-c2-high)
  (final-mode-check-responsibleagent)
 )

 (:action setdir
  :parameters (?d1 - direction)
  :precondition (and
   (not (final-mode))
   (check)
   (updated)
  )
  :effect (and
   (not (check))
   (not (hasdir agent left))
   (not (hasdir agent straight))
   (not (hasdir agent right))
   (hasdir agent ?d1)
  )
 )
 (:action setstop
  :parameters ()
  :precondition (and
   (not (final-mode))
   (check)
   (updated)
  )
  :effect (and
   (not (check))
   (not (hasdir agent left))
   (not (hasdir agent straight))
   (not (hasdir agent right))
  )
 )
 (:action update
  :parameters ()
  :precondition (and
   (not (final-mode))
   (check)
   (not (updated))
  )
  :effect (and
   (not (check))
   (updated)
   (forall
    (?c1 - car ?c2 - car ?y1 - ypos ?x1 - xpos)
    (when
     (and
      (not (= ?c1 ?c2))
      (not (= ?c1 agent))
      (not (= ?c2 agent))
      (not (hascrashed ?c1))
      (haspos ?c1 ?x1 ?y1)
      (haspos ?c2 ?x1 ?y1)
     )
     (and
      (hascrashed ?c1)
      (hascrashed ?c2)
     )
    ))
   (forall
    (?c1 - car ?y1 - ypos ?x1 - xpos)
    (when
     (and
      (not (= ?c1 agent))
      (haspos agent ?x1 ?y1)
      (haspos ?c1 ?x1 ?y1)
     )
     (and
      (hasbumped agent)
      (hasbumped ?c1)
      (not (hasdir ?c1 left))
      (not (hasdir ?c1 straight))
      (not (hasdir ?c1 right))
     )
    ))
  )
 )
 (:action go
  :parameters ()
  :precondition (and
   (not (final-mode))
   (check)
   (updated)
  )
  :effect (and
   (not (check))
   (not (updated))
   (forall
    (?c1 - car ?d1 - direction ?y1 - ypos ?y2 - ypos ?x1 -
   ↪ xpos ?x2 - xpos)
    (when
     (and
      (not (hascrashed ?c1))
      (haspos ?c1 ?x1 ?y1)
      (hasdir ?c1 ?d1)
      (nextx ?d1 ?x1 ?x2)
      (nexty ?y1 ?y2)
     )
     (and
      (not (haspos ?c1 ?x1 ?y1))
      (haspos ?c1 ?x2 ?y2)
     )
    ))
   (forall
    (?c1 - car ?x1 - xpos ?y1 - ypos)
    (when
     (and
      (haspos ?c1 ?x1 ?y1)
      (hasdir ?c1 left)
      (nextx left ?x1 ?x1)
     )
     (damagerail ?c1)))
   (forall
    (?c1 - car ?x1 - xpos ?y1 - ypos)
    (when
     (and
      (haspos ?c1 ?x1 ?y1)
      (hasdir ?c1 right)
      (nextx right ?x1 ?x1)
     )
     (damagerail ?c1)))
  )
 )
 (:action check-ethical-features
  :parameters ()
  :precondition (and
   (not (final-mode))
   (not (check))
  )
  :effect (and
   (check)
   (forall
    (?c1 - car)
    (when
     (hascrashed ?c1)
     (danger ?c1 high)))
   (forall
    (?c1 - car)
    (when
     (hasbumped ?c1)
     (danger ?c1 low)))
   (when
    (hascrashed agent)
    (responsibleagent))
   (when
    (hasbumped agent)
    (responsibleagent))
  )
 )
 (:action final-mode-start
  :parameters ()
  :precondition (not (final-mode))
  :effect (final-mode)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-damagerail-agent
  :parameters ()
  :precondition (and
   (final-mode)
   (damagerail agent)
   (not (final-mode-check-damagerail-agent))
  )
  :effect (and
   (final-mode-check-damagerail-agent)
   (increase (total-cost) 1)
  )
 )
 (:action
   ↪ final-mode-check-ethical-features-True-damagerail-agent
  :parameters ()
  :precondition (and
   (final-mode)
   (not (damagerail agent))
   (not (final-mode-check-damagerail-agent))
  )
  :effect (final-mode-check-damagerail-agent)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-damagerail-c1
  :parameters ()
  :precondition (and
   (final-mode)
   (damagerail c1)
   (not (final-mode-check-damagerail-c1))
  )
  :effect (and
   (final-mode-check-damagerail-c1)
   (increase (total-cost) 1)
  )
 )
 (:action final-mode-check-ethical-features-True-damagerail-c1
  :parameters ()
  :precondition (and
   (final-mode)
   (not (damagerail c1))
   (not (final-mode-check-damagerail-c1))
  )
  :effect (final-mode-check-damagerail-c1)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-damagerail-c2
  :parameters ()
  :precondition (and
   (final-mode)
   (damagerail c2)
   (not (final-mode-check-damagerail-c2))
  )
  :effect (and
   (final-mode-check-damagerail-c2)
   (increase (total-cost) 1)
  )
 )
 (:action final-mode-check-ethical-features-True-damagerail-c2
  :parameters ()
  :precondition (and
   (final-mode)
   (not (damagerail c2))
   (not (final-mode-check-damagerail-c2))
  )
  :effect (final-mode-check-damagerail-c2)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-danger-agent-low
  :parameters ()
  :precondition (and
   (final-mode)
   (danger agent low)
   (not (final-mode-check-danger-agent-low))
  )
  :effect (and
   (final-mode-check-danger-agent-low)
   (increase (total-cost) 4)
  )
 )
 (:action
   ↪ final-mode-check-ethical-features-True-danger-agent-low
  :parameters ()
  :precondition (and
   (final-mode)
   (not (danger agent low))
   (not (final-mode-check-danger-agent-low))
  )
  :effect (final-mode-check-danger-agent-low)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-danger-c1-low
  :parameters ()
  :precondition (and
   (final-mode)
   (danger c1 low)
   (not (final-mode-check-danger-c1-low))
  )
  :effect (and
   (final-mode-check-danger-c1-low)
   (increase (total-cost) 4)
  )
 )
 (:action final-mode-check-ethical-features-True-danger-c1-low
  :parameters ()
  :precondition (and
   (final-mode)
   (not (danger c1 low))
   (not (final-mode-check-danger-c1-low))
  )
  :effect (final-mode-check-danger-c1-low)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-danger-c2-low
  :parameters ()
  :precondition (and
   (final-mode)
   (danger c2 low)
   (not (final-mode-check-danger-c2-low))
  )
  :effect (and
   (final-mode-check-danger-c2-low)
   (increase (total-cost) 4)
  )
 )
 (:action final-mode-check-ethical-features-True-danger-c2-low
  :parameters ()
  :precondition (and
   (final-mode)
   (not (danger c2 low))
   (not (final-mode-check-danger-c2-low))
  )
  :effect (final-mode-check-danger-c2-low)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-danger-agent-high
   ↪
  :parameters ()
  :precondition (and
   (final-mode)
   (danger agent high)
   (not (final-mode-check-danger-agent-high))
  )
  :effect (and
   (final-mode-check-danger-agent-high)
   (increase (total-cost) 48)
  )
 )
 (:action
   ↪ final-mode-check-ethical-features-True-danger-agent-high
  :parameters ()
  :precondition (and
   (final-mode)
   (not (danger agent high))
   (not (final-mode-check-danger-agent-high))
  )
  :effect (final-mode-check-danger-agent-high)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-danger-c1-high
  :parameters ()
  :precondition (and
   (final-mode)
   (danger c1 high)
   (not (final-mode-check-danger-c1-high))
  )
  :effect (and
   (final-mode-check-danger-c1-high)
   (increase (total-cost) 16)
  )
 )
 (:action
   ↪ final-mode-check-ethical-features-True-danger-c1-high
  :parameters ()
  :precondition (and
   (final-mode)
   (not (danger c1 high))
   (not (final-mode-check-danger-c1-high))
  )
  :effect (final-mode-check-danger-c1-high)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-danger-c2-high
  :parameters ()
  :precondition (and
   (final-mode)
   (danger c2 high)
   (not (final-mode-check-danger-c2-high))
  )
  :effect (and
   (final-mode-check-danger-c2-high)
   (increase (total-cost) 16)
  )
 )
 (:action
   ↪ final-mode-check-ethical-features-True-danger-c2-high
  :parameters ()
  :precondition (and
   (final-mode)
   (not (danger c2 high))
   (not (final-mode-check-danger-c2-high))
  )
  :effect (final-mode-check-danger-c2-high)
 )
 (:action
   ↪ final-mode-check-ethical-features-False-responsibleagent
  :parameters ()
  :precondition (and
   (final-mode)
   (responsibleagent)
   (not (final-mode-check-responsibleagent))
  )
  :effect (and
   (final-mode-check-responsibleagent)
   (increase (total-cost) 48)
  )
 )
 (:action
   ↪ final-mode-check-ethical-features-True-responsibleagent
  :parameters ()
  :precondition (and
   (final-mode)
   (not (responsibleagent))
   (not (final-mode-check-responsibleagent))
  )
  :effect (final-mode-check-responsibleagent)
 )

)
```

Listing B.7: Autonomous driver PDDL domain code with action costs.

B.4.2 Problem file

```
(define (problem crash-p-01_GEN)
 (:domain crash-d_GEN)
 (:init
  (= (total-cost) 0)
  (updated)
  (haspos agent x1 y1)
  (haspos c1 x2 y1)
  (haspos c2 x2 y3)
  (hasdir agent straight)
  (hasdir c1 straight)
  (nextx straight x1 x1)
  (nextx straight x2 x2)
  (nextx right x1 x2)
  (nextx right x2 x2)
  (nextx left x1 x1)
  (nextx left x2 x1)
  (nexty y1 y2)
  (nexty y2 y3)
  (nexty y3 y4)
  (nexty y4 y4)
 )

 (:goal
  (and
   (updated)
   (haspos agent x2 y4)
   (not (hascrashed agent))
   (check)
   (final-mode)
   (final-mode-check-damagerail-agent)
   (final-mode-check-damagerail-c1)
   (final-mode-check-damagerail-c2)
   (final-mode-check-danger-agent-low)
   (final-mode-check-danger-c1-low)
   (final-mode-check-danger-c2-low)
   (final-mode-check-danger-agent-high)
   (final-mode-check-danger-c1-high)
   (final-mode-check-danger-c2-high)
   (final-mode-check-responsibleagent)
  )
 )

 (:metric minimize
  (total-cost)
 )
)
```

Listing B.8: Autonomous driver PDDL problem code with action costs.

C

Appendix: Ethical rank learning implementation

Here, we will list the full Problog code corresponding to implementations and examples described in Chapter 6. We will start by discussing different Problog language extensions which we used in Chapter 6. Then, we list the full code of the ethical ranks learning problem. And finally, we show the code of the case study described in the same chapter.

We remind the reader that in Section 6.1 we gave a high-level overview of the problem, and then explained the most relevant part of the implementation in Section 6.2. For more information about Problog, we refer the reader to [De Raedt et al., 2007] or its official website[1].

C.1 Logic language

As mentioned before, in this chapter we will represent knowledge using Problog [De Raedt et al., 2007], an extension of Prolog [Bratko, 2001] with probabilistic annotations described in Chapter 2. However, it is important to note that, throughout this chapter, will use certain extensions of Problog rules (provided by their common libraries) for the sake of representation simplicity, namely (i) negation as failure, (ii) numerical expressions, and (iii) lists.

Negation as failure allows expressions of the form not(B_i) in the body of rules, denoting that B_i was not possible to prove [Clark, 1978], i.e: the expression not(B_i) holds whenever B_i cannot be proven by the program. This extension has been well studied, is compatible with the semantics we described in Chapter 2, and is commonly used by Prolog. For more information about negation as failure, we refer the reader to [Naish, 1986] and [Ling, 1990].

Then, **numerical expressions** permit to represent:

- Comparisons as: `A>B, A<B, A=B, A\=B` (i.e: not equal operator in Prolog) where A, B are either variables or integers, denoting comparison between the terms, and

[1]https://dtai.cs.kuleuven.be/problog/

- Assignments as: `A is B+C, A is B-C, A is B*C, A is B/C`, where `A` is an un-instanced variable and `B,C` are either variables which can only be unified with numerical values, or constant numerical values, representing the unification of `A` with the result of the arithmetical operation at the right side of the '`is`' special construct.

It is worth noting that although Prolog and Problog offer these expressions out-of-the-box, one can also define them through predicates, so they do not extend the generality or semantics of the language.

And finally, **lists** are a built-in data type of Prolog (and Problog), represented as:

- `[ ]` : a special constant symbol representing the empty list,
- `[` H_1`, ...,` H_n `]` : a list with elements H_1`, ...,` H_n, where each H_i is an term (variable, constant, or functor), and
- `[ H | L ]` : a list with first element `H`, an term known as the head, attached to another list `L`, called the tail.

In addition, by utilizing lists, we will also use the predicate `length(L, N)` provided by Problog itself, which unifies the variable `N` with the length of the list `L`.

For more information about the usage of these functionalities and some examples, we refer the reader to [Bramer, 2005].

C.2 Parameter learning code

In this section, we list the complete Problog code of the ethical ranks learning problem described in Chapter 6, including the domain and theory encodings.

C.2.1 Domain encoding P_s

```
situation(s_1).
    ⋮
situation(s_{n_s}).
plan(a_1).
    ⋮
plan(a_{n_a}).
ethical_feature(e_1).
    ⋮
ethical_feature(e_{n_e}).
type(e_1,t_1).
    ⋮
type(e_{n_e},t_{n_e}).
rank(r_1).
```

```
     ⋮
rank($r_{n_r}$).
has_plan($a_{1,1}, s_1$). ... has_plan($a_{1,m_1}, s_1$).
     ⋮
has_plan($a_{n,1}, s_n$). ... has_plan($a_{n,m_n}, s_n$).
has_feature($e_{1,1,1}, a_{1,1}, s_1$).
     ⋮
has_feature($e_{1,m_1,w_{1,m_1}}, a_{1,m_1}, s_1$).
     ⋮
has_feature($e_{n,1,1}, a_{n,1}, s_n$).
     ⋮
has_feature($e_{n,m_n,w_{n,m_n}}, a_{n,m_n}, s_n$).
```

Listing C.1: Problog encoding of situational background knowledge P_s

C.2.2 Theory encoding P_t

```
max_val(0,0).
max_val(R,V) :- rank(R), R > 0,
    amount_of_rank(R,N),
    val(R,V1),
    R1 is R-1,
    max_val(R1,V2),
    V is V1*N+V2.

val(0,0).
val(R,V) :-
    rank(R),
    R > 0,
    R1 is R-1,
    max_val(R1,V1),
    V is V1+1.

val_until_rank(A,S,0,0).
val_until_rank(A,S,R,N) :-
    rank(R),
    R > 0,
    amount_satisfied_of_rank(A,S,R,N1),
    val(R,V),
    R1 is R-1,
    val_until_rank(A,S,R1,N2),
    N is N1*V+N2.

val(A,S,N) :- val_until_rank(A,S,$n_r$,N).

worse(A,B,S) :-
```

```
    has_plan(A,S),
    has_plan(B,S),
    A \= B,
    val(A,S,R1),
    val(B,S,R2),
    R1 < R2.

best(A,S) :-
    has_plan(A,S),
    not(worse(A,B,S)).

amount_of_rank(R,N) :-
    rank(R),
    rank_assignment(RA),
    amount_of_rank(R,N,RA).
amount_of_rank(R,0,[]) :-
    rank(R).
amount_of_rank(R,N,[F,R1|RA]) :-
    rank(R),
    R = R1,
    amount_of_rank(R,N1,RA),
    N is N1+1.
amount_of_rank(R,N,[F,R1|RA]) :-
    rank(R),
    R \= R1,
    amount_of_rank(R,N,RA).

amount_satisfied_of_rank(A,S,R,N) :-
    has_plan(A,S),
    rank(R),
    rank_assignment(RA),
    amount_satisfied_of_rank(A,S,R,N,RA).
amount_satisfied_of_rank(A,S,R,0,[]) :-
    has_plan(A,S),
    rank(R).
amount_satisfied_of_rank(A,S,R,N,[F,R1|RA]) :-
    has_plan(A,S),
    rank(R),
    R = R1,
    satisfies(F,A,S),
    amount_satisfied_of_rank(A,S,R,N1,RA),
    N is N1+1.
amount_satisfied_of_rank(A,S,R,N,[F,R1|RA]) :-
    has_plan(A,S),
    rank(R),
    R = R1,
    not(satisfies(F,A,S)),
    amount_satisfied_of_rank(A,S,R,N,RA).
amount_satisfied_of_rank(A,S,R,N,[F,R1|RA]) :-
    has_plan(A,S),
    rank(R),
    R \= R1,
    amount_satisfied_of_rank(A,S,R,N,RA).

satisfies(F,A,S) :-
    has_feature(F,A,S),
    type(F,'+').
satisfies(F,A,S) :-
    type(F,'-'),
    not(has_feature(F,A,S)).
```

Listing C.2: Full Problog encoding of *val* and *best* background knowledge P_t

C.3 Example full code

Here, we list the complete Problog code of the case study of Chapter 6.

C.3.1 Domain encoding P_s

```
situation(s1).
    ⋮
situation(s6).

plan(go(left)).
plan(go(right)).

ethical_feature(danger(agent)).
ethical_feature(danger(c1)).
ethical_feature(danger(c2)).
ethical_feature(responsible(agent)).

type(danger(agent), '-').
type(danger(c1), '-').
type(danger(c2), '-').
type(responsible(agent), '-').

rank(1).
rank(2).

has_plan(go(right), s1).
has_plan(go(left), s1).
has_feature(danger(c1), go(left), s1).
has_feature(danger(c2), go(left), s1).
has_feature(danger(agent), go(left), s1).

has_plan(go(right), s2).
has_plan(go(left), s2).
has_feature(danger(c1), go(right), s2).
has_feature(danger(c2), go(right), s2).
has_feature(danger(agent), go(left), s2).

has_plan(go(right), s3).
has_plan(go(left), s3).
has_feature(danger(c1), go(right), s3).
has_feature(danger(c2), go(left), s3).

has_plan(go(right), s4).
has_plan(go(left), s4).
has_feature(danger(agent), go(right), s4).
has_feature(responsible(agent), go(right), s4).
has_feature(danger(c1), go(right), s4).
has_feature(danger(c1), go(left), s4).
has_feature(danger(c2), go(left), s4).

has_plan(go(right), s5).
has_plan(go(left), s5).
has_feature(danger(agent), go(right), s5).
has_feature(responsible(agent), go(right), s5).
has_feature(danger(agent), go(left), s5).

has_plan(go(right), s6).
has_plan(go(left), s6).
has_feature(danger(c1), go(right), s6).
has_feature(danger(c2), go(right), s6).
has_feature(danger(c1), go(left), s6).
```

Listing C.3: Case study Problog encoding of situational background knowledge P_s

C.3.2 Interpretations I_s

$$\begin{aligned}
I_1^+ = \{&\texttt{best(go(right),s1)}, \texttt{best(go(right),s2)},\\
&\texttt{best(go(left),s3)}, \texttt{best(go(left),s4)},\\
&\texttt{best(go(left),s5)}, \texttt{best(go(left),s6)}\}\\
I_1^- = \{&\texttt{best(go(left),s1)}, \texttt{best(go(left),s2)},\\
&\texttt{best(go(right),s3)}, \texttt{best(go(right),s4)},\\
&\texttt{best(go(right),s5)}, \texttt{best(go(right),s6)}\}\\
I_2^+ = \{&\texttt{best(go(right),s1)}, \texttt{best(go(right),s2)},\\
&\texttt{best(go(left),s3)}, \texttt{best(go(left),s4)},
\end{aligned}$$

best(go(left),s5), best(go(left),s6)}

I_2^- = {best(go(left),s1), best(go(left),s2),
best(go(right),s3), best(go(right),s4),
best(go(right),s5), best(go(right),s6)}

I_3^+ = {best(go(right),s1), best(go(right),s2),
best(go(left),s3), best(go(left),s4),
best(go(left),s5), best(go(left),s6)}

I_3^- = {best(go(left),s1), best(go(left),s2),
best(go(right),s3), best(go(right),s4),
best(go(right),s5), best(go(right),s6)}

I_4^+ = {best(go(right),s1), best(go(right),s2),
best(go(right),s3), best(go(left),s4),
best(go(left),s5), best(go(left),s6)}

I_4^- = {best(go(left),s1), best(go(left),s2),
best(go(left),s3), best(go(right),s4),
best(go(right),s5), best(go(right),s6)}

I_5^+ = {best(go(right),s1), best(go(left),s2),
best(go(right),s3), best(go(left),s4),
best(go(left),s5), best(go(left),s6)}

I_5^- = {best(go(left),s1), best(go(right),s2),
best(go(left),s3), best(go(right),s4),
best(go(right),s5), best(go(right),s6)}

Bibliography

[Abel et al., 2016] Abel, D., MacGlashan, J., and Littman, M. L. (2016). Reinforcement learning as a framework for ethical decision making. In *Workshops at the thirtieth AAAI conference on artificial intelligence*.

[Adadi and Berrada, 2018] Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160.

[Aghighi and Bäckström, 2015] Aghighi, M. and Bäckström, C. (2015). Cost-optimal and net-benefit planning—a parameterised complexity view. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Alexander and Moore, 2021] Alexander, L. and Moore, M. (2021). Deontological Ethics. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2021 edition.

[Allen et al., 2005] Allen, C., Smit, I., and Wallach, W. (2005). Artificial morality: Top-down, bottom-up, and hybrid approaches. *Ethics and information technology*, 7(3):149–155.

[Anderson et al., 2005a] Anderson, M., Anderson, S., and Armen, C. (2005a). Towards machine ethics: Implementing two action-based ethical theories. In *Proceedings of the AAAI 2005 fall symposium on machine ethics*, pages 1–7.

[Anderson and Anderson, 2007] Anderson, M. and Anderson, S. L. (2007). The status of machine ethics: a report from the aaai symposium. *Minds and Machines*, 17(1):1–10.

[Anderson and Anderson, 2018] Anderson, M. and Anderson, S. L. (2018). Geneth: A general ethical dilemma analyzer. *Paladyn, Journal of Behavioral Robotics*, 9(1):337–357.

[Anderson et al., 2005b] Anderson, M., Anderson, S. L., and Armen, C. (2005b). Medethex: Toward a medical ethics advisor. In *AAAI Fall Symposium: Caring Machines*, pages 9–16.

[Anderson et al., 2006] Anderson, M., Anderson, S. L., and Armen, C. (2006). An approach to computing ethics. *IEEE Intelligent Systems*, 21(4):56–63.

[Anderson, 2008] Anderson, S. L. (2008). Asimov's "three laws of robotics" and machine metaethics. *Ai & Society*, 22(4):477–493.

[Apt, 1990] Apt, K. R. (1990). Logic programming. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990:493–574.

[Arkin et al., 2011] Arkin, R. C., Ulam, P., and Wagner, A. R. (2011). Moral decision making in autonomous systems: Enforcement, moral emotions, dignity, trust, and deception. *Proceedings of the IEEE*, 100(3):571–589.

[Arkin et al., 2009] Arkin, R. C., Ulam, P. D., and Duncan, B. (2009). An ethical governor for constraining lethal action in an autonomous system. Technical report, Georgia Institute of Technology.

[Armstrong, 2015] Armstrong, S. (2015). Motivated value selection for artificial agents. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[Arrieta et al., 2020] Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115.

[Arrow et al., 2010] Arrow, K. J., Sen, A., and Suzumura, K. (2010). *Handbook of social choice and welfare*, volume 2. Elsevier.

[Asimov, 1950] Asimov, I. (1950). *I, robot*, volume 1. Gnome press.

[Attfield, 2012] Attfield, R. (2012). Ethics: an overview.

[Awad et al., 2018] Awad, E., Dsouza, S., Kim, R., Schulz, J., Henrich, J., Shariff, A., Bonnefon, J.-F., and Rahwan, I. (2018). The moral machine experiment. *Nature*, 563(7729):59–64.

[Awad et al., 2020] Awad, E., Levine, S., Loreggia, A., Mattei, N., Rahwan, I., Rossi, F., Talamadupula, K., Tenenbaum, J., and Kleiman-Weiner, M. (2020). When is it morally acceptable to break the rules? a preference-based approach. In *12th multidisciplinary workshop on advances in preference handling (MPREF 2020)*.

[Bäckström and Nebel, 1995] Bäckström, C. and Nebel, B. (1995). Complexity results for sas+ planning. *Computational Intelligence*, 11(4):625–655.

[Baum, 2020] Baum, S. D. (2020). Social choice ethics in artificial intelligence. *AI & SOCIETY*, 35(1):165–176.

[Bemelmans et al., 2012] Bemelmans, R., Gelderblom, G. J., Jonker, P., and De Witte, L. (2012). Socially assistive robots in elderly care: a systematic review into effects and effectiveness. *Journal of the American Medical Directors Association*, 13(2):114–120.

[Berreby et al., 2015] Berreby, F., Bourgne, G., and Ganascia, J.-G. (2015). Modelling moral reasoning and ethical responsibility with logic programming. In *Logic for programming, artificial intelligence, and reasoning*, pages 532–548. Springer.

[Berreby et al., 2017] Berreby, F., Bourgne, G., and Ganascia, J.-G. (2017). A declarative modular framework for representing and applying ethical principles. In *IFAAMAS 2017*.

[Berreby et al., 2018] Berreby, F., Bourgne, G., and Ganascia, J.-G. (2018). Event-based and scenario-based causality for computational ethics. In *AAMAS 2018-17th International Conference on Autonomous Agents and Multiagent Systems*, pages 147–155. International Foundation for Autonomous Agents and Multiagent Systems.

[Bongard et al., 2006] Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121.

[Bonnemains et al., 2016] Bonnemains, V., Saurel, C., and Tessier, C. (2016). How Ethical Frameworks Answer to Ethical Dilemmas: Towards a Formal Model. In *EDIA@ ECAI 2016*, pages 44–51.

[Bourgne et al., 2021] Bourgne, G., Sarmiento, C., and Ganascia, J.-G. (2021). Ace modular framework for computational ethics: dealing with multiple actions, concurrency and omission. In *1st International Workshop on Computational Machine Ethics, Online event.*

[Boutilier et al., 1999] Boutilier, C., Brafman, R. I., Hoos, H. H., and Poole, D. (1999). Reasoning with conditional ceteris paribus preference statements. In *UAI*, volume 99, pages 71–80.

[Bramer, 2005] Bramer, M. (2005). *Logic programming with Prolog*, volume 9. Springer.

[Bratko, 2001] Bratko, I. (2001). *Prolog programming for artificial intelligence.* Pearson education.

[Bremner et al., 2019] Bremner, P., Dennis, L. A., Fisher, M., and Winfield, A. F. (2019). On proactive, transparent, and verifiable ethical reasoning for robots. *Proceedings of the IEEE*, 107(3):541–561.

[Brewka, 2004] Brewka, G. (2004). A rank based description language for qualitative preferences. In *ECAI 2004*, volume 16, page 303.

[Bringsjord et al., 2016] Bringsjord, S., Ghosh, R., and Payne-Joyce, J. (2016). Deontic counteridenticals. *Agents (EDIA)*, 2016:40–45.

[Bringsjord and Taylor, 2012] Bringsjord, S. and Taylor, J. (2012). Introducing divine-command robot ethics. *Robot ethics: the ethical and social implication of robotics*, pages 85–108.

[Broadie and Rowe, 2002] Broadie, S. and Rowe, C. (2002). Aristotle: Nicomachean ethics: Translation, introduction, commentary.

[Bruers and Braeckman, 2014] Bruers, S. and Braeckman, J. (2014). A review and systematization of the trolley problem. *Philosophia*, 42(2):251–269.

[Brundage, 2014] Brundage, M. (2014). Limitations and risks of machine ethics. *JEAIL*, 26(3):355–372.

[Bryant, 1986] Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691.

[Catherine and Cohen, 2016] Catherine, R. and Cohen, W. (2016). Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *Proceedings of the 10th ACM conference on recommender systems*, pages 325–332.

[Cave et al., 2018] Cave, S., Nyrup, R., Vold, K., and Weller, A. (2018). Motivations and risks of machine ethics. *Proceedings of the IEEE*, 107(3):562–574.

[Cenamor et al., 2014] Cenamor, I., De La Rosa, T., Fernández, F., et al. (2014). Ibacop and ibacop2 planner. *IPC 2014 planner abstracts*, pages 35–38.

[Chaput et al., 2021] Chaput, R., Duval, J., Boissier, O., Guillermin, M., and Hassas, S. (2021). A multi-agent approach to combine reasoning and learning for an ethical behavior. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 13–23.

[Charisi et al., 2017] Charisi, V., Dennis, L., Fisher, M., Lieck, R., Matthias, A., Slavkovik, M., Sombetzki, J., Winfield, A. F., and Yampolskiy, R. (2017). Towards moral autonomous systems. *arXiv preprint arXiv:1703.04741*.

[Chen et al., 2004] Chen, Y., Hsu, C.-W., and Wah, B. W. (2004). Sgplan: Subgoal partitioning and resolution in planning. *Edelkamp et al.(Edelkamp, Hoffmann, Littman, & Younes, 2004)*.

[Clark, 1978] Clark, K. L. (1978). Negation as failure. In *Logic and data bases*, pages 293–322. Springer.

[Clocksin and Mellish, 2003] Clocksin, W. F. and Mellish, C. S. (2003). *Programming in PROLOG*. Springer Science & Business Media.

[Cointe et al., 2016] Cointe, N., Bonnet, G., and Boissier, O. (2016). Ethical judgment of agents' behaviors in multi-agent systems. In *AAMAS 2016*, pages 1106–1114.

[Coles and Coles, 2011] Coles, A. and Coles, A. (2011). Lprpg-p: Relaxed plan heuristics for planning with preferences. In *ICAPS 2011*.

[Constant, 2013] Constant, B. (2013). *Des réactions politiques*. Presses Électroniques de France.

[Cranefield et al., 2017] Cranefield, S., Winikoff, M., Dignum, V., and Dignum, F. (2017). No pizza for you: Value-based plan selection in bdi agents. In *IJCAI*, pages 178–184.

[Craven and Slattery, 2001] Craven, M. and Slattery, S. (2001). Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1):97–119.

[Dantsin et al., 2001] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425.

[De Raedt and Kimmig, 2015] De Raedt, L. and Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47.

[De Raedt et al., 2007] De Raedt, L., Kimmig, A., and Toivonen, H. (2007). Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad.

[Dennis and Fisher, 2018] Dennis, L. and Fisher, M. (2018). Practical challenges in explicit ethical machine reasoning. *arXiv preprint arXiv:1801.01422*.

[Dennis et al., 2016] Dennis, L., Fisher, M., Slavkovik, M., and Webster, M. (2016). Formal verification of ethical choices in autonomous systems. *Robotics and Autonomous Systems*, 77:1–14.

[Devore, 2011] Devore, J. L. (2011). *Probability and Statistics for Engineering and the Sciences*. Cengage learning.

[Diana and Marescaux, 2015] Diana, M. and Marescaux, J. (2015). Robotic surgery. *Journal of British Surgery*, 102(2):e15–e28.

[Dimopoulos et al., 2006] Dimopoulos, Y., Gerevini, A., Haslum, P., and Saetti, A. (2006). The benchmark domains of the deterministic part of ipc-5. *Abstract Booklet of the competing planners of ICAPS-06*, pages 14–19.

[Dyoub et al., 2020] Dyoub, A., Costantini, S., Lisi, F. A., and Letteri, I. (2020). Logic-based machine learning for transparent ethical agents. In *CILC*.

[Edelkamp and Helmert, 2001] Edelkamp, S. and Helmert, M. (2001). Mips: The model-checking integrated planning system. *AI magazine*, 22(3):67–67.

[Edelkamp et al., 2006] Edelkamp, S., Jabbar, S., and Nazih, M. (2006). Large-scale optimal pddl3 planning with mips-xxl. *5th International Planning Competition Booklet (IPC-2006)*, pages 28–30.

[Edelkamp and Kissmann, 2009] Edelkamp, S. and Kissmann, P. (2009). Optimal symbolic planning with action costs and preferences. In *Twenty-First International Joint Conference on Artificial Intelligence*. Citeseer.

[Etienne, 2021] Etienne, H. (2021). The dark side of the 'moral machine'and the fallacy of computational ethical decision-making for autonomous vehicles. *Law, Innovation and Technology*, 13(1):85–107.

[Feldmann et al., 2006] Feldmann, R., Brewka, G., and Wenzel, S. (2006). Planning with prioritized goals. In *KR*, pages 503–514.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *AIJ*, 2(3-4):189–208.

[Foot, 1967] Foot, P. (1967). The problem of abortion and the doctrine of the double effect. *Oxford review*, 5.

[Fox and Long, 2003] Fox, M. and Long, D. (2003). Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124.

[Gamez et al., 2020] Gamez, P., Shank, D. B., Arnold, C., and North, M. (2020). Artificial virtue: The machine question and perceptions of moral character in artificial moral agents. *AI & SOCIETY*, 35(4):795–809.

[Ganascia, 2007] Ganascia, J.-G. (2007). Ethical system formalization using non-monotonic logics. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 29.

[Ganascia, 2015] Ganascia, J.-G. (2015). Non-monotonic resolution of conflicts for ethical reasoning. In *A Construction Manual for Robots' Ethical Systems*, pages 101–118. Springer.

[Geffner and Haslum, 2000] Geffner, P. H. H. and Haslum, P. (2000). Admissible heuristics for optimal planning. In *Proceedings of the 5th Internat. Conf. of AI Planning Systems (AIPS 2000)*, pages 140–149.

[Geißer et al., 2015] Geißer, F., Keller, T., and Mattmüller, R. (2015). Delete relaxations for planning with state-dependent action costs. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Gerevini and Long, 2005] Gerevini, A. and Long, D. (2005). Plan constraints and preferences in pddl3. Technical report, Technical Report 2005-08-07, Department of Electronics for Automation ....

[Gerevini et al., 2009] Gerevini, A. E., Haslum, P., Long, D., Saetti, A., and Dimopoulos, Y. (2009). Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *AIJ*, 173(5-6):619–668.

[Gert and Gert, 2020] Gert, B. and Gert, J. (2020). The Definition of Morality. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2020 edition.

[Ghallab et al., 2004] Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

[Govindarajulu and Bringsjord, 2017] Govindarajulu, N. S. and Bringsjord, S. (2017). On automating the doctrine of double effect. *arXiv preprint arXiv:1703.08922*.

[Gutmann, 2011] Gutmann, B. (2011). *On continuous distributions and parameter estimation in probabilistic logic programs.* PhD thesis, Ph. D thesis, KULeuven.

[Gutmann et al., 2011] Gutmann, B., Thon, I., and Raedt, L. D. (2011). Learning the parameters of probabilistic logic programs from interpretations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 581–596. Springer.

[Haines, 2006] Haines, W. (2006). Consequentialism. https://www.iep.utm.edu/conseque/.

[Halpern, 2016] Halpern, J. Y. (2016). *Actual causality.* MiT Press.

[Harper, 2009] Harper, S. J. (2009). Ethics versus morality: A problematic divide. *Philosophy & social criticism*, 35(9):1063–1077.

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107.

[Hashmi et al., 2014] Hashmi, M., Governatori, G., and Wynn, M. T. (2014). Modeling obligations with event-calculus. In *RuleML 2014*, pages 296–310. Springer.

[Hegde et al., 2020] Hegde, A., Agarwal, V., and Rao, S. (2020). Ethics, prosperity, and society: Moral evaluation using virtue ethics and utilitarianism. In *29th International Joint Conference on Artificial Intelligence (IJCAI 2020). doi*, volume 10.

[Helmert, 2003] Helmert, M. (2003). Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262.

[Helmert, 2006] Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

[Helmert et al., 2008] Helmert, M., Do, M., and Refanidis, I. (2008). IPC 2008 Website. https://ipc08.icaps-conference.org/deterministic/.

[Helmert and Domshlak, 2011] Helmert, M. and Domshlak, C. (2011). Lm-cut: Optimal planning with the landmark-cut heuristic. *Seventh international planning competition (IPC 2011), deterministic part*, pages 103–105.

[Helmert et al., 2011] Helmert, M., Röger, G., and Karpas, E. (2011). Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, pages 28–35.

[Hoffmann, 2003] Hoffmann, J. (2003). The metric-ff planning system: Translating“ignoring delete lists”to numeric state variables. *Journal of artificial intelligence research*, 20:291–341.

[Hoffmann and Edelkamp, 2005] Hoffmann, J. and Edelkamp, S. (2005). The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579.

[Honarvar and Ghasem-Aghaee, 2009] Honarvar, A. R. and Ghasem-Aghaee, N. (2009). Casuist bdi-agent: a new extended bdi architecture with the capability of ethical reasoning. In *International conference on artificial intelligence and computational intelligence*, pages 86–95. Springer.

[Horn, 1951] Horn, A. (1951). On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21.

[Howard and Muntean, 2017] Howard, D. and Muntean, I. (2017). Artificial moral cognition: moral functionalism and autonomous moral agency. In *Philosophy and computing*, pages 121–159. Springer.

[Hsu et al., 2006] Hsu, C.-W., Wah, B. W., Huang, R., and Chen, Y. (2006). New features in sgplan for handling preferences and constraints in pddl3. 0. In *Proceedings of the Fifth International Planning Competition*, pages 39–42. Citeseer.

[Hursthouse and Pettigrove, 2018] Hursthouse, R. and Pettigrove, G. (2018). Virtue Ethics. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2018 edition.

[Ivankovic et al., 2014] Ivankovic, F., Haslum, P., Thiébaux, S., Shivashankar, V., and Nau, D. (2014). Optimal planning with global numerical state constraints. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 24.

[Jaques, 2019] Jaques, A. E. (2019). Why the moral machine is a monster. *University of Miami School of Law*, 10:1–10.

[Jedwabny et al., 2021a] Jedwabny, M., Bisquert, P., and Croitoru, M. (2021a). Generating preferred plans with ethical features. In Bell, E. and Keshtkar, F., editors, *Proceedings of the Thirty-Fourth International Florida Artificial Intelligence Research Society Conference, North Miami Beach, Florida, USA, May 17-19, 2021*.

[Jedwabny et al., 2021b] Jedwabny, M., Bisquert, P., and Croitoru, M. (2021b). Probabilistic rule induction for transparent cbr under uncertainty. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 117–130. Springer.

[Karpas and Domshlak, 2009] Karpas, E. and Domshlak, C. (2009). Cost-optimal planning with landmarks. In *IJCAI*, pages 1728–1733. Pasadena, CA.

[Katz et al., 2018] Katz, M., Sohrabi, S., Samulowitz, H., and Sievers, S. (2018). Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*, pages 57–64.

[Kautz et al., 2006] Kautz, H., Selman, B., and Hoffmann, J. (2006). Satplan: Planning as satisfiability. In *5th international planning competition*, volume 20, page 156.

[Kersting and Raedt, 2001] Kersting, K. and Raedt, L. D. (2001). Towards combining inductive logic programming with bayesian networks. In *International Conference on Inductive Logic Programming*, pages 118–131. Springer.

[Keyder and Geffner, 2009] Keyder, E. and Geffner, H. (2009). Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556.

[Kimmig et al., 2011] Kimmig, A., Demoen, B., De Raedt, L., Costa, V. S., and Rocha, R. (2011). On the implementation of the probabilistic logic programming language problog. *Theory and Practice of Logic Programming*, 11(2-3):235–262.

[Kowalski and Kuehner, 1971] Kowalski, R. and Kuehner, D. (1971). Linear resolution with selection function. *Artificial Intelligence*, 2(3-4):227–260.

[Leśniak, 2012] Leśniak, K. (2012). Invariant sets and knaster-tarski principle. *Open Mathematics*, 10(6):2077–2087.

[Levinson et al., 2011] Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., et al. (2011). Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE.

[Lifschitz, 1999] Lifschitz, V. (1999). Answer set planning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 373–374. Springer.

[Lindner and Bentzen, 2017] Lindner, F. and Bentzen, M. M. (2017). The hybrid ethical reasoning agent immanuel. In *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 187–188.

[Lindner et al., 2017] Lindner, F., Bentzen, M. M., and Nebel, B. (2017). The HERA approach to morally competent robots. In *IROS 2017*, pages 6991–6997. IEEE.

[Lindner et al., 2019] Lindner, F., Mattmüller, R., and Nebel, B. (2019). Moral permissibility of action plans. In *AAAI 2019*, volume 33, pages 7635–7642.

[Ling, 1990] Ling, T. W. (1990). The prolog not-predicate and negation as failure rule. *New Generation Computing*, 8(1):5–31.

[Lloyd, 1994] Lloyd, J. W. (1994). Practical advtanages of declarative programming. In *GULP-PRODE (1)*, pages 18–30.

[López et al., 2015] López, C. L., Celorrio, S. J., and Olaya, Á. G. (2015). The deterministic part of the seventh international planning competition. *Artificial Intelligence*, 223:82–119.

[Loreggia et al., 2018] Loreggia, A., Mattei, N., Rossi, F., and Venable, K. B. (2018). Preferences and ethical principles in decision making. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 222–222.

[Malle and Scheutz, 2019] Malle, B. F. and Scheutz, M. (2019). Learning how to behave. In *Handbuch maschinenethik*, pages 255–278. Springer.

[Malle et al., 2017] Malle, B. F., Scheutz, M., and Austerweil, J. L. (2017). Networks of social and moral norms in human and robot agents. In *A world with robots*, pages 3–17. Springer.

[Marín and Sartor, 1999] Marín, R. H. and Sartor, G. (1999). Time and norms: a formalisation in the event-calculus. In *ICAIL 1999*, pages 90–99.

[McDermott and Doyle, 1980] McDermott, D. and Doyle, J. (1980). Non-monotonic logic i. *Artificial intelligence*, 13(1-2):41–72.

[McIntyre, 2019] McIntyre, A. (2019). Doctrine of Double Effect. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2019 edition.

[McLaren, 2003] McLaren, B. M. (2003). Extensionally defining principles and cases in ethics: An ai model. *Artificial Intelligence*, 150(1-2):145–181.

[Menkes Van Den Briel et al., 2004] Menkes Van Den Briel, R. S., Do, M. B., and Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 562–569.

[Mill and Bentham, 1987] Mill, J. S. and Bentham, J. (1987). *Utilitarianism and other essays*. Penguin UK.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning*, volume 1. McGraw-hill New York.

[Mohri et al., 2018] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

[Moor, 2011] Moor, J. H. (2011). The nature, importance, and difficulty of machine ethics. *Machine ethics*, pages 13–20.

[Muggleton, 1991] Muggleton, S. (1991). Inductive logic programming. *New generation computing*, 8(4):295–318.

[Naish, 1986] Naish, L. (1986). *Negation and control in Prolog*, volume 238. Springer Science & Business Media.

[Nilsson, 1984] Nilsson, N. (1984). Shakey the robot. Technical report, SRI International Technical Note 323.

[Noothigattu et al., 2018] Noothigattu, R., Gaikwad, S., Awad, E., Dsouza, S., Rahwan, I., Ravikumar, P., and Procaccia, A. (2018). A voting-based system for ethical decision making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Panagiotidi and Vázquez-Salceda, 2011] Panagiotidi, S. and Vázquez-Salceda, J. (2011). Towards practical normative agents: a framework and an implementation for norm-aware planning. In *COIN 2011*, pages 93–109. Springer.

[Papadimitriou, 2003] Papadimitriou, C. (2003). *Computational Complexity*, page 260–265. John Wiley and Sons Ltd.

[Poole, 1993] Poole, D. (1993). Logic programming, abduction and probability. *New Generation Computing*, 11(3):377–400.

[Poole, 1997] Poole, D. (1997). The independent choice logic for modelling multiple agents under uncertainty. *Artificial intelligence*, 94(1-2):7–56.

[Puterman, 1990] Puterman, M. L. (1990). Markov decision processes. *Handbooks in operations research and management science*, 2:331–434.

[Quinn, 2013] Quinn, P. L. (2013). Divine command theory. *The Blackwell guide to ethical theory*, pages 81–102.

[Rao et al., 1995] Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: from theory to practice. In *Icmas*, volume 95, pages 312–319.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Why should i trust you? explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

[Richardson and Domingos, 2006] Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine learning*, 62(1):107–136.

[Richter et al., 2011] Richter, S., Westphal, M., and Helmert, M. (2011). Lama 2008 and 2011. In *International Planning Competition*, pages 117–124.

[Riguzzi and Swift, 2018] Riguzzi, F. and Swift, T. (2018). A survey of probabilistic logic programming. In *Declarative Logic Programming: Theory, Systems, and Applications*, pages 185–228.

[Robinson, 1965] Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41.

[Rodriguez-Soto et al., 2021] Rodriguez-Soto, M., Lopez-Sanchez, M., and Rodriguez-Aguilar, J. A. (2021). Multi-objective reinforcement learning for designing ethical environments. In *IJCAI*, pages 545–551.

[Ross and Ross, 2002] Ross, D. and Ross, W. D. (2002). *The right and the good.* Oxford University Press.

[Ross, 1930] Ross, W. (1930). *The Right and the Good.* Oxford University Press.

[Russell, 2010] Russell, S. J. (2010). *Artificial intelligence a modern approach.* Pearson Education, Inc.

[Sacerdoti, 1974] Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135.

[Sato, 1995] Sato, T. (1995). A statistical learning method for logic programs with distribution semantics. In *ICLP*.

[Sato and Kameya, 1997] Sato, T. and Kameya, Y. (1997). Prism: a language for symbolic-statistical modeling. In *IJCAI*, volume 97, pages 1330–1339. Citeseer.

[Sato and Kameya, 2001] Sato, T. and Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454.

[Sayre-McCord, 2013] Sayre-McCord, G. (2013). Contractarianism. *The Blackwell Guide to Ethical Theory*, pages 332–353.

[Seipp and Röger, 2018] Seipp, J. and Röger, G. (2018). Fast downward stone soup 2018. *IPC2018–Classical Tracks*, pages 72–74.

[Sergot, 2004] Sergot, M. (2004). An action language for modelling norms and institutions. *Technical Report 2004/8.* Publisher: Imperial College London.

[Shahriari and Shahriari, 2017] Shahriari, K. and Shahriari, M. (2017). Ieee standard review — ethically aligned design: A vision for prioritizing human wellbeing with artificial intelligence and autonomous systems. In *IEEE*, pages 197–201.

[Shanahan, 1999] Shanahan, M. (1999). The event calculus explained. In *Artificial intelligence today*, pages 409–430. Springer.

[Shanahan, 2016] Shanahan, M. (2016). The Frame Problem. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, Spring 2016 edition.

[Sinnott-Armstrong, 2021] Sinnott-Armstrong, W. (2021). Consequentialism. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition.

[Tärnlund, 1977] Tärnlund, S.-Å. (1977). Horn clause computability. *BIT Numerical Mathematics*, 17(2):215–226.

[Tetlock et al., 2000] Tetlock, P. E., Kristel, O. V., Elson, S. B., Green, M. C., and Lerner, J. S. (2000). The psychology of the unthinkable: taboo trade-offs, forbidden base rates, and heretical counterfactuals. *Journal of personality and social psychology*, 78(5):853.

[Thornton et al., 2016] Thornton, S. M., Pan, S., Erlien, S. M., and Gerdes, J. C. (2016). Incorporating ethical considerations into automated vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 18(6):1429–1439.

[Tolmeijer et al., 2020] Tolmeijer, S., Kneer, M., Sarasua, C., Christen, M., and Bernstein, A. (2020). Implementations in machine ethics: a survey. *ACM Computing Surveys (CSUR)*, 53(6):1–38.

[Torralba et al., 2014] Torralba, A., Alcázar, V., Borrajo, D., Kissmann, P., and Edelkamp, S. (2014). Symba*: A symbolic bidirectional a* planner. In *International Planning Competition*, pages 105–108.

[Torralba and Pommerening, 2018] Torralba, A. and Pommerening, F. (2018). IPC 2018 Website. https://ipc2018-classical.bitbucket.io/.

[Vallati et al., 2018] Vallati, M., Chrpa, L., and Mccluskey, T. L. (2018). What you always wanted to know about the deterministic part of the international planning competition (ipc) 2014 (but were too afraid to ask). *The Knowledge Engineering Review*, 33.

[Van den Broeck et al., 2010] Van den Broeck, G., Thon, I., Van Otterlo, M., and De Raedt, L. (2010). Dtproblog: A decision-theoretic probabilistic prolog. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24.

[Van Emden and Kowalski, 1976] Van Emden, M. H. and Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742.

[Vanderelst and Winfield, 2018] Vanderelst, D. and Winfield, A. (2018). An architecture for ethical robots inspired by the simulation theory of cognition. *Cognitive Systems Research*, 48:56–66.

[Vaughan and Zuluaga, 2006] Vaughan, R. and Zuluaga, M. (2006). Use your illusion: Sensorimotor self-simulation allows complex agents to plan with incomplete self-knowledge. In *International Conference on Simulation of Adaptive Behavior*, pages 298–309. Springer.

[Von Wright, 1951] Von Wright, G. H. (1951). Deontic logic. *Mind*, 60(237):1–15.

[Von Wright, 1968] Von Wright, G. H. (1968). *An essay in deontic logic and the general theory of action*. Amsterdam: North-Holland Pub. Co.

[Wallach and Allen, 2008] Wallach, W. and Allen, C. (2008). *Moral machines: Teaching robots right from wrong*. Oxford University Press.

[Winfield et al., 2014] Winfield, A. F., Blum, C., and Liu, W. (2014). Towards an ethical robot: internal models, consequences and ethical action selection. In *Conference towards autonomous robotic systems*, pages 85–96. Springer.

[Wolfram, 1984] Wolfram, S. (1984). Cellular automata as models of complexity. *Nature*, 311(5985):419–424.

[Wu and Lin, 2018] Wu, Y.-H. and Lin, S.-D. (2018). A low-cost ethics shaping approach for designing reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[Xu et al., 2019] Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., and Zhu, J. (2019). Explainable ai: A brief survey on history, research areas, approaches and challenges. In *CCF international conference on natural language processing and Chinese computing*, pages 563–574. Springer.

[Yu et al., 2018] Yu, H., Shen, Z., Miao, C., Leung, C., Lesser, V. R., and Yang, Q. (2018). Building ethics into artificial intelligence. *arXiv preprint arXiv:1812.02953*.