# Modeling automated legal and ethical compliance for trustworthy AI

Yousef Taheri Sojasi

SORBONNE UNIVERSITÉ

École doctorale 130 : Informatique, Télécommunications et Électronique
Laboratoire de Recherche en Informatique (LIP6)

# THÈSE DE DOCTORAT

Pour obtenir le grade de Docteur en Informatique

En vue d'une soutenance publique
par

**Yousef TAHERI SOJASI**

# Modeling Automated Legal and Ethical Compliance for Trustworthy AI

Thèse sous la direction de Jean-Gabriel GANASCIA et Gauvain BOURGNE

Devant un jury composé de:

| | | |
|---|---|---|
| **Pr. MARIE-JEANNE LESOT** | Sorbonne Université | Présidente et Examinatrice |
| **Pr. MADALINA CROITORU** | Université de Montpellier | Rapporteuse |
| **Dr. HDR FABIEN TARISSAN** | ENS Parsi-Saclay | Rapporteur |
| **Dr. CATHERINE TESSIER** | ONERA | Examinatrice |
| **Pr. JEAN-GABRIEL GANASCIA** | Sorbonne Université | Directeur |
| **Dr. HDR GAUVAIN BOURGNE** | Sorbonne Université | Directeur |

# Abstract

The advancements in artificial intelligence have led to significant legal and ethical issues related to privacy, bias, accountability, etc. In recent years, many regulations have been put in place to limit or mitigate the risks associated with AI. Compliance with these regulations are necessary for the reliability of AI systems and to ensure that they are being used responsibly. In addition, reliable AI systems should also be ethical, ensuring alignment with ethical norms. Compliance with applicable laws and adherence to ethical principles are essential for most AI applications. We investigate this problem from the point of view of AI agents. In other words, how an agent can ensure the compliance of its actions with legal and ethical norms. We are interested in approaches based on logical reasoning to integrate legal and ethical compliance in the agent's planning process. The specific domain in which we pursue our objective is the processing of personal data. i.e., the agent's actions involve the use and processing of personal data. A regulation that applies in such a domain is the General Data Protection Regulations (GDPR). In addition, processing of personal data may entail certain ethical risks with respect to privacy or bias.

We address this issue through a series of contributions presented in this thesis. Starting by the GDPR compliance, we adopt Event Calculus with the Answer Set Programming(ASP) to model agents' actions and use it for planning and checking the compliance with GDPR. A policy language is used to represent the GDPR obligations and requirements. Then we investigate the issue of ethical compliance. A pluralistic ordinal utility model is proposed that allows one to evaluate actions based on moral values. This model is based on multiple criteria and uses voting systems to aggregate evaluations on an ordinal scale. We then integrate this utility model and the legal compliance framework in a Hierarchical Task Network(HTN) planner. In this contribution, legal norms are considered hard constraints and ethical norm as soft constraint. Finally, as a last step, we further explore the possible combinations of legal and ethical compliance with the planning agent and propose a unified framework. This framework captures the interaction and conflicts between legal and ethical norms and is tested in a use case with AI systems managing the delivery of health care items.

## Keywords

Computational Ethics; Answer Set Programming; Planning; Event Calculus; Hierarchical task networks; Voting systems; Legal compliance; Ethical compliance; Legal knowledge representation; Trustworthy AI.

# Résumé

Les avancées en intelligence artificielle ont conduit à des enjeux juridiques et éthiques significatifs liés à la vie privée, aux biais, à la responsabilité, etc. Ces dernières années, de nombreuses réglementations ont été mises en place pour limiter ou atténuer les risques associés à l'IA. Le respect de ces réglementations est nécessaire pour la fiabilité des systèmes d'IA et pour garantir une utilisation responsable. De plus, des systèmes d'IA fiables doivent également être éthiques, en assurant une conformité avec les normes éthiques. La conformité aux lois applicables et l'adhésion aux principes éthiques sont essentielles pour la plupart des applications de l'IA. Nous étudions ce problème du point de vue des agents d'IA. En d'autres termes, comment un agent peut-il garantir que ses actions respectent les normes juridiques et éthiques. Nous nous intéressons aux approches basées sur le raisonnement logique pour intégrer la conformité juridique et éthiques dans le processus de planification de l'agent. Le domaine spécifique dans lequel nous poursuivons notre objectif est le traitement des données personnelles, c'est-à-dire, les actions de l'agent impliquent l'utilisation et le traitement des données personnelles. Une réglementation applicable dans ce domaine est le Règlement Général sur la Protection des Données (RGPD). De plus, le traitement des données personnelles peut entraîner certains risques éthiques en matière de vie privée ou de biais.

Nous abordons cette question à travers une série de contributions présentées dans cette thèse. Nous commençons par la question de la conformité au RGPD. Nous adoptons le Calcul des Événements avec la Programmation par Ensembles de Réponses (ASP) pour modéliser les actions des agents et l'utiliser pour planifier et vérifier la conformité au RGPD. Un langage de policy est utilisé pour représenter les obligations et exigences du RGPD. Ensuite, nous examinons la question de la conformité éthique. Un modèle d'utilité ordinale pluraliste est proposé, permettant d'évaluer les actions en fonction des valeurs morales. Ce modèle est basé sur plusieurs critères et utilise des systèmes de vote pour agréger les évaluations sur une échelle ordinale. Nous intégrons ensuite ce modèle d'utilité et le cadre de conformité juridique dans un planificateur de Réseau de Tâches Hiérarchiques (HTN). Dans cette contribution, les normes juridiques sont considérées comme des contraintes "hard" et les normes éthiques comme des contraintes "soft". Enfin, en dernière étape, nous explorons davantage les combinaisons possibles de la conformité juridique et éthique avec l'agent de planification et proposons un cadre unifié. Ce cadre capture l'interaction

et les conflits entre les normes juridiques et éthiques et est testé dans un cas d'utilisation avec des systèmes d'IA gérant la livraison d'articles médicaux.

## Mots-clefs

Éthique Computationnelle; Answer Set Programming; Planification; Calcul des Événements; Réseaux de tâches hiérarchiques; systèmes de vote; Conformité Juridique; Conformité éthique; Représentation des connaissances juridiques; IA responsable; IA digne de confiance.

# Acknowledgments

I would like to warmly thank my supervisors Gauvain Bourgne, Jean-Gabriel Ganascia for their continuous support and invaluable knowledge.

I would also like to thank Ken Satoh, Hisashi Hayashi, Kanae Tsushima, as well as other members of the RECOMP project for their for their precious help, intellectual rigour and kindness in these trying times.

I would also like to express my sincere gratitude the members of the Jury for accepting to review my thesis. This work is part of the "Real-time Compliance Mechanism" RECOMP project, financed by the French Agence Nationale de la Recherche (ANR, French Research Agency) under the reference ANR-20-IADJ-0004.

# Contents

# List of Figures

# List of Tables

# Introduction

Artificial intelligence (AI) systems are increasingly being used in applications that directly impact human welfare and social functions, from healthcare and transportation to law enforcement and employment. The AI advancements in these applications have led to significant legal and ethical issues related to privacy, bias, accountability, surveillance, etc. In recent years, many regulations have been put in place to limit or mitigate the risks associated with AI. These regulations are designed to ensure that AI systems operate safely, ethically, and transparently. The primary goals of AI regulations include managing risks, complying with existing laws, and protecting individual rights while also protecting public interests and promoting innovation. Compliance with these regulations is necessary for the reliability of AI systems and to ensure that they are being used responsibly.

Laws may fall behind technological advances, occasionally conflict with ethical principles, or fail to address specific issues. Hence, reliable AI systems should also be ethical and ensure their alignment with ethical norms. According to the EU Ethics Guidelines for Trustworthy AI [95], it is essential for AI systems to be "lawful" and "ethical" to achieve trustworthiness. The former means complying with all applicable laws and regulations, and the latter means ensuring adherence to ethical principles and values. According to this document, the third requirement for a trustworthy AI is "robustness", that is out of the scope of this thesis. Ethical guidelines usually discuss the development, deployment and use of AI in a way that supports ethical principles and promotes good. These guidelines explore ethical issues with AI systems, ensuring that their benefits are maximized while minimizing potential harms and risks.

Compliance with the law and adherence to ethical principles apply to most AI applications and should be met throughout the system's entire life cycle. There are areas of research in AI, especially in the machine learning (ML) community, to deal with these issues, such as fairness [155], privacy [193], and explainability [121] in ML algorithms. Unlike these methods, we take a symbolic approach and are interested in mechanisms based on logical reasoning to address this issue. Our objective in this thesis is to study this issue in the context of AI agents, particularly agents capable of planning, i.e. deriving a series of actions automatically to achieve a certain objective in a deterministic environment. Planning is used to model a variety of domains including robotics, autonomous cars, etc. In addition, each domain concerns a different set of ap-

plicable regulations. The specific domain in which we pursue our objective is the processing of personal data. i.e., the agent's actions involve the use and processing of personal data.

This domain has received a lot of attention in recent years, especially from the point of view of regulatory bodies. A significant regulation that applies in this case is the General Data Protection Regulation (GDPR). GDPR imposes strict requirements on the processing of personal data, including data used in AI systems, and the data subject's rights. There are also ethical concerns that apply to the processing of personal data. They include the use of sensitive categories of personal data, the use of large-scale processing, algorithmic bias, etc.

According to the specified domain, the main research track in this thesis is to study GDPR compliance and ethical alignment of an AI system that handles personal data automatically. It is useful to note that the term ethical compliance may suggest that ethics is only a set of rules, as in the case of law. Although certain aspects of ethical reasoning involve rules, this is not the case in general. In our context, "ethically compliant" means that the agent is capable of morally evaluating the options and choosing the best one according to some (specified) principles.

As a motivating example, consider an agent who is confronted with one of the following decisions.

1. Choosing a data set to process for a certain purpose;
2. Choosing a path to transfer personal data to a certain destination; and,
3. Choosing a processing to apply on personal data.

The agent makes one or a series of such choices automatically in order to perform a processing on personal data or produce a specific output. A series of choices is a plan to reach a certain desired state. The agent uses a planning mechanism to automatically generate and select plans. The first interest is the design of such an AI agent, i.e. how to model this agent's actions and the planning mechanism to handle personal data automatically? How can legal and ethical evaluations be integrated into the agent's reasoning process?

There are also a number of questions regarding the legal compliance of the options. For example, what are the GDPR requirements in this example to check the compliance of each option? How can these requirements be integrated into the agent's reasoning process? An important obligation of GDPR is the data subject's consent for processing of their personal data. How can a data subject's consent and other regulations be represented according to the requirements of the GDPR? In terms of ethical evaluation, we might ask several questions that contribute to ethical evaluation of options. For example, in the first case, what categories of personal data entail higher risks? Which path is safe?, in terms of data protection to transfer personal data? or which processing is less biased?

GDPR compliance checking requires expert knowledge and a formalized representation of GDPR regulations. Legal knowledge representation is a field of

computational law that deals with formalizing legal knowledge. Languages and ontologies have been developed to represent different relations in legal documents and deontic rules like LegalRuleML[150, 14, 13]. In particular, a number of ontologies and policy languages have been created to represent GDPR requirements [52, 27]. These tools provide a formalization of GDPR concepts that can be used for compliance checking; however, they need to be adapted and integrated into the planning process.

The ethical evaluation of options is also challenging due to the lack of universal rules, the sensitivity to context, and the difficulty of representing ethical principles. In addition, the evaluation process has to be expressive and reasonable. In the computational ethics community, there are works that propose models of ethical evaluations [82, 122]. These works mainly model ethical theories[178], e.g. consequentialism and deontology, and evaluate their judgments in ethical dilemmas such as the trolley problem or comparing morality of action plans [48, 23]. These models are useful in showing the mechanism behind ethical theories and their conflicts. However, adopting these models for our specific domain is not simple. In the case of deontological theories, the derivation of rules and codes of conducts from fundamental principles is challenging. In case of utilitarianist theories such as act utilitarianism, the notion of utility is essential to evaluate options. But there is no expressive and consistent way to assign utility values without introducing arbitrary numbers into our domain. Finally, these ethical evaluation models do not consider legal compliance and possible interactions or conflicts with legal norms.

The other challenge is the design of such an agent, the choice of a suitable planning formalism, and integration with both the legal compliance and ethical evaluation in a single architecture. Law and ethics overlap in many areas, but the interaction between the two should be investigated, for example, which one has priority over the other? Can both legal compliance and ethical evaluations be combined as a single component?

The study of this problem is conducted in several steps. We start with the GDPR compliance checking problem. That is, modeling an AI agent that uses planning to handle personal data and checks the compliance of its actions with GDPR. We adopt Event Calculus with Answer Set Programming(ASP) to model agents' actions, their effects, and preconditions, and abductive reasoning to generate plans. The plans are then verified for compliance with GDPR. We use a policy language, called SPECIAL [27], to translate and represent GDPR requirements into ASP and use it to check the compliance of the plans. This model is also capable of explaining missing regulations in case a plan is not compliant. The proposed model is published in the JURISIN post-proceedings [176] as a first contribution.

In the next step, we deal with the ethical evaluation problem. A literature review was conducted to study the approaches for the ethical evaluation of the plan. We start by the question of "How several options can be compared ethically?" In other words, which action is aligned with moral values? This research led to the development of a pluralistic utility model [177]. This model takes into account multiple criteria to represent moral values in a hierarchical

structure. Measurements are modeled on an ordinal scale and aggregated using voting systems.

After addressing the legal compliance and ethical evaluation model separately, we integrate both components together with a planning mechanism. The planning formalism in this case is a version of the Hierarchical Task Network (HTN) planner implemented in Prolog. An advantage of this planner is that it supports execution and real-time replanning. Hence, the architecture enables compliance checking in real time. In this architecture, legal regulations are considered hard norms, and ethical norms are considered soft norms. This means that the satisfaction of legal norms is necessary, while ethical norms should be satisfied as much as possible. This is one possible way to integrate the legal compliance and ethical evaluation with the agent's planning mechanism.

Finally, as a last step, we further explore architectures for integrating legal compliance and ethical evaluation in the agent's planning process. In this architecture, legal and ethical evaluations are integrated in a single component. This enables the model to capture interactions or possible conflicts between legal and ethical norms. This architecture is adopted to model a system that that handles multiple agents for the delivery of health care items. In this use case, a planning heuristic based on the event calculus is used to model agents actions and plan a delivery route. This architecture allows modeling hard ethical norms and their trade-off with the legal norms.

This thesis is organized as follows. Part I presents the state-of-the-art and background required to read the contributions. In Chapters 1 and 2 we discuss the state of the art in the legal domain and ethics, respectively. Chapter 3 presents the tools and technical background used in the contributions. Part II contains the contributions made during the research in this thesis. In Chapter 4 the data processing with the GDPR compliance checking framework is presented. Chapter 5 describes the pluralistic ordinal utility model based on multiple hierarchical criteria. In Chapter 6, both legal and ethical components are integrated with an HTN planner. In Chapter 7 we explore other architectures to integrate legal and ethical components in the planning process and present our new use case. Finally, in part III we discuss the conclusions and future work.

This thesis can be read in either the presented order or in other orders considering the dependencies among the chapters. The dependencies are depicted in Figure 1. For example, one way to read this thesis is to start with Chapters 1 and 3.1.2 and 3.2.1, which are the prerequisites for reading Chapter 4. Another possible order is to start with Chapter 2, 3.1.2, and 3.3 which are required to read Chapter 5 of the contributions.

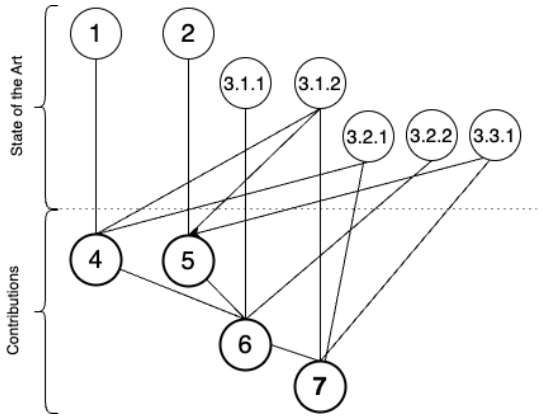



Figure 1: Dependencies among chapters

# Part I

# State of the Art

# Chapter 1

# State of the Art: Legal Compliance

As artificial intelligence continues to evolve towards greater autonomy, mechanisms to comprehend and adhere to applicable laws and regulations are becoming more essential. The law represents a multifaceted system of rules and regulations that govern society and influence its politics, economics, and social interactions [21].

AI, as a tool with huge impact and the potential to harm individuals or societies, should be used responsibly. Regulations serve as a crucial safeguard to ensure the safety of AI systems and prevent harm to individuals and societies. Compliance with these regulations in the design and deployment of AI systems can be challenging, especially in automated settings, which is of our interest. Computational law is a branch of legal informatics that focuses on the intersection of law and computer science. It is an approach to automate legal reasoning that focuses on semantically rich laws, regulations, contract terms, and business rules [126]. It plays a crucial role in areas such as document review, compliance, case prediction, and legal research [106, 26]. Current applications of computational laws include the automation of legal tasks and compliance check [52, 81, 90], understanding legal texts by applying Natural Language Processing (NLP) techniques [160, 56], detecting patterns in legal data and predicting case outcomes using machine learning algorithms [187, 124], etc.

An important area in computational law is the representation of legal knowledge for legal reasoning and compliance checking. In this thesis, we are interested in formalized approaches in knowledge representations that can be integrated in the agent's reasoning process and enable compliance checking.

This chapter presents a brief introduction to some of the significant AI regulations, in particular, GDPR, which we focus on in this thesis. We discuss representation of legal knowledge in computational law and explain LegalRuleML, a standard language to represent legal norms, regulations, guidelines, and policies in machine-readable format. In addition, we describe the SPECIAL policy language that is used in the thesis to represent GDPR requirements.

## 1.1 AI Regulations

AI regulations refer to legal frameworks and policies established by governments and regulatory bodies to govern the development, deployment, and use of artificial intelligence. The regulations aim to address various ethical concerns related to AI, including privacy, transparency, accountability, fairness, safety, and societal impact. Recently, there has been a trend in AI regulations. These regulations are rapidly evolving, with various initiatives being proposed and implemented globally. Each country or region may adopt its own approach to AI regulation. Some of the most significant AI regulations and initiatives include.

- **EU AI Act** [60]: In April 2021, the EU proposed the AI Act, which aims to regulate AI systems considered high-risk. It outlines requirements for AI developers, users, and regulators, including data governance, transparency, accountability, and human oversight.

- **EU Digital Services Act (DSA) [59]**: Adopted in 2022, significantly tightens regulations on digital platforms operating within the EU. It enforces greater accountability for removing illegal content, increases transparency around algorithmic decision-making, and strengthens user rights in content moderation. Additionally, it imposes stricter controls on targeted advertising and requires platforms to perform annual risk assessments.
- **EU Data Act [61]**: Introduced by the European Commission in February 2022, aims to regulate the access and use of data generated from connected devices across the EU. It facilitates data sharing, ensures fair access, protects privacy, and reduces dependency on major platform providers.
- **USA California Privacy Rights Act (CPRA)** [37]: CPRA regulate the collection and use of personal data, including data used in AI systems. These regulations give consumers more control over their personal information and impose obligations on businesses handling such data.
- **USA Algorithmic Accountability Act** [5]: Proposed in the United States, this act aims to regulate the use of automated decision-making systems by federal agencies. It requires impact assessments for high-risk AI systems, ensuring fairness, transparency, and accountability.

Another significant regulation is the General Data Protection Regulation (GDPR) [63]. Although not specific to AI, GDPR imposes strict requirements on the processing of personal data, including data used in AI systems. In this thesis, we limited our objective to domains that involve the processing of personal data. Since our use case domain concerns with GDPR regulations, we describe it in more detail in the next section.

## 1.2 GDPR

The GDPR is a comprehensive data protection regulation enacted by the European Union (EU) to enhance the protection of individuals' personal data and harmonize data privacy laws across the EU member states. GDPR applies to the processing of personal data of individuals within the EU, regardless of where the processing takes place. It also applies to organizations outside the EU that offer goods or services to individuals in the EU or monitor their behavior. Some of the most important provisions in the GDPR include:

- **Lawful Basis for Processing**: Data controllers or processors must have a lawful basis for processing personal data. Lawful bases include consent, contract necessity, legal obligation, vital interests, public task, and legitimate interests.
- **Consent**: Consent for data processing must be freely given, specific, informed, and unambiguous. Individuals have the right to withdraw consent at any time.

- **Rights of Data Subjects**: GDPR grants individuals several rights over their personal data, including the right to access, rectify, erase, restrict processing, data portability, and object to processing.
- **Data Protection by Design**: Data controllers must implement data protection principles into the design of their systems and processes. They should also adopt measures to ensure that only the necessary personal data is processed.
- **Data Transfers**: GDPR imposes restrictions on the transfer of personal data outside the EU to ensure an adequate level of protection. Adequacy decisions, standard contractual clauses, binding corporate rules, and derogation are mechanisms for lawful data transfers.
- **Accountability**: Organizations must demonstrate compliance with GDPR principles by maintaining records of data processing activities and implementing appropriate technical and organizational measures.

The GDPR provides a robust framework for protecting the privacy and personal data of individuals. This regulation emphasizes individuals' rights over their data, including the right to access, rectify, and erase their information, giving them greater control over their personal information. GDPR establishes clear guidelines and principles to ensure data are processed in a lawful, fair, and transparent way. It helps mitigate the risks associated with data processing and promote trust between consumers and organizations by mandating strict consent requirements and robust security measures. In the next sections, we introduce legal ontologies that are used to codify regulations in a machine-understandable manner. We focus particularly on ontologies that can be used to model GDPR regulation.

## 1.3 Legal Knowledge Representation

Representation of knowledge and reasoning is essential to solve problems in a given domain. This requires a structured framework for organizing and categorizing concepts like entities, rules, and relationships in a way that computers can understand and process. An *ontology* is a formal specification of a shared *conceptualization* in a given domain that is simplified and abstract [86]. A conceptualization can be viewed in simple terms [78] as a tuple $(D, \mathbf{R})$ where

- $D$ is a set called the *universe of discourse*
- $\mathbf{R}$ is a set of relations on $D$

According to this definition, the members of the set $\mathbf{R}$ are ordinary mathematical relations on $D$, that is, sets of ordered tuples of elements of $D$. So each element of $\mathbf{R}$ is an extensional relation, reflecting a specific world state involving the elements of $D$. Ontologies in AI provide a basic framework for

knowledge representation. They are used to model the semantics of a domain that allows reasoning, inference, and decision-making. Formal languages and representation formats are used to develop ontologies such as OWL (Web Ontology Language)[130, 10], RDF (Resource Description Framework)[53, 157], and logic-based languages such as Prolog [49].

An important application area of ontologies is the legal domain. Fundamental concepts and entities within a legal domain involve contracts, statutes, case law, legal roles, rights, obligations, and legal procedures. Ontologies capture the structure and content of legal knowledge, often used in legal information systems and knowledge management [139]. The logical structure and rules of the ontology can be used to perform tasks such as legal analysis, decision support, compliance check, and legal reasoning.

Legal ontologies can be categorized as general-purpose or domain-specific. The former consisted of core concepts that are common in law in different domains. In contrast, domain-specific ontologies capture concepts in a limited domain application. The domain of interest in our research is the compliance with GDPR. Here, we describe a core legal ontology for representing legal rules and a domain-specific ontology to represent GDPR policies.

### 1.3.1 LegalRuleML

LegalRuleML (Legal Rule Markup Language) [149, 150] is a standardized language used to represent legal norms, regulations, guidelines and policies in a machine-readable format. It extends RuleML (Rule Markup Language) [25], a general purpose language for representing rules and logical assertions in XML, incorporating features specific to legal documents. These features include legal concepts, norms, obligations, permissions, and prohibitions. LegalRuleML provides as a rule interchange language for the legal domain and rule-based systems. It is used in applications ranging from legal reasoning and decision support systems to automated compliance and analysis of legal text. In this section, we briefly describe some features of LegalRuleML. More detailed explanations can be found in [14, 13, 180].

#### Structure

LegalRuleML has a conceptual basis for representing, formalizing, and reasoning with legal provisions. It supports deontic operators (e.g. obligations, permissions, prohibitions, rights) and supports multiple semantics of negation. For example, consider the simple rule in the phrase "every man is obliged to walk". This phrase can be formulated in Standard Deontic Logic [131] as $\forall_x[man(x) \rightarrow \mathbf{OB}(walk(x))]$. This formula is encoded in LegalRuleML using the XML format.

```
<lrml:PrescriptiveStatement key="someuniquekey">
    <ruleml:Rule closure="universal">
        <ruleml:if>
            <ruleml: Atom >
```

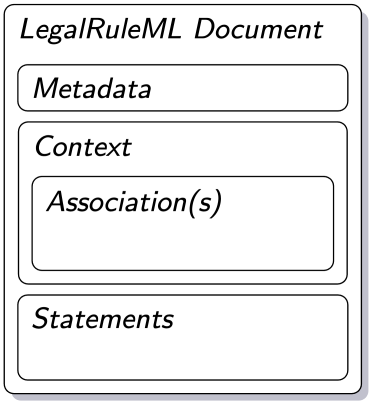



Figure 1.1: LegalRuleML structure [115]

```
                <ruleml:Rel iri="man" />
                <ruleml:Var key=":x">x</ruleml:Var>
            </ruleml:Atom>
        </ruleml:if>
        <ruleml:then>
            <lrml:Obligation>
                <ruleml:Atom>
                    <ruleml:Rel iri="walk" />
                    <ruleml:Var keyref=:x" />
                </ruleml:Atom>
            </lrml:Obligation>
        </ruleml:then>
    </ruleml:Rule>
</lrml:PrescriptiveStatement>
```

The tags `lrml` and `ruleml` refer to the ontological concepts in LegalRuleML and RuleML, respectively. The premise of the rule encapsulated by the `<ruleml:if>` tags and the conclusion using `<ruleml:then>`. The obligation $\mathbf{OB}(walk(x))$ is enclosed within the `</lrml:Obligation>` blocks. The predicates "man" and "walk" are connected to ontological concepts via the attribute `iri` of the block `< ruleml : Rel >`. The rule is encoded as a prescriptive statement using `<lrml:PrescriptiveStatement>` blocks. Prescriptive statements are used to model deontic rules, i.e. statements that involve obligations, permissions, or prohibitions.

Legal rules in general are more complex and concern contextual information. They need to be linked to this information to be interpreted correctly. LegalRuleMl provides a structure for appointing this information with legal rules. The general structure of a LegalRuleML document is illustrated in Figure 1.1. The metadata include components to link the formal rules and the legally binding textual statements. Textual normative provisions are considered a legal source and are referred to, using the block `<lrml:LegalSources>`. For example,

the following snippet is a legal source named `ref9` that refers to a law in United States Code [183]. The United States Code is the official codification of the general and permanent federal statutes of the United States.

```
<lrml:LegalSource key="ref9"
    sameAs="http://www.law.cornell.edu/wiki/lexcraft/
↪ section_identifiers_lii"
/>

```

Contextual information is essential for interpreting the legal rule. For example, the norms and their associated rules have validity within a specific time frame and with respect to three main legal axes: entry into force, efficacy, and applicability. Legal documents may frequently need to be revised as societal norms or judicial frameworks evolve. Hence, temporal information is crucial in modeling legal rules. Other contextual information related to legal rules may include agents and roles, authority, jurisdictions, and temporal information and the strength of the rule (defeasible, defeater, strict). For example, consider the rule `rule1` with the following properties.

- `rule1` has TemporalCharateristics `tblock1`
- `rule1` has Strength `defeasible`
- `rule1` has Author `aut1`
- `rule1` has Jurisdiction `US`
- `rule1` has Authority `congress`

The block `<Context>` contains the characteristics that are related to this rule. These contextual information are associated with `rule1` using the tag `<lrml:appliesCONTEXT>`.

```
<lrml:Context key="ruleInfo1" hasCreationDate="#t8">
    <lrml:appliesTemporalCharacteristics keyref="#tblock1"/>
    <lrml:appliesStrength iri="lrmlv:Defeasible"/>
    <lrml:appliesRole>
        <lrml:Role iri="lrmlv:Author">
            <lrml:filledBy keyref="#aut1"/>
        </lrml:Role>
    </lrml:appliesRole>
    <lrml:appliesAuthority keyref="#congress"/>
    <lrml:appliesJurisdiction keyref="jurisdictions:us"/>
    <lrml:toStatement keyref="#rule1"/>
</lrml:Context>

```

The block `<TemporalCharateristics>` captures the temporal dimension of the rules. In this example, it refers to the key `#tblock1`, that is, the period to enter into force or the period of efficacy of the corresponding rule. `#tblock1`

is represented as follows, and the block `<TimeInstants>` is used to indicate an specific time point.

```
<lrml:TimeInstants>
    <ruleml:Time key="t1">
        <ruleml:Data xsi:type="xs:dateTime">2012-07-21T00:00:00Z
        </ruleml:Data>
    </ruleml:Time>
</lrml:TimeInstants>
<lrml:TemporalCharacteristics key="tblock1">
    <lrml:TemporalCharacteristic key="nev1">
        <lrml:forRuleStatus iri="lrmlv:Efficacious"/>
        <lrml:hasStatusDevelopment iri="lrmlv:Starts"/>
        <lrml:atTimeInstant keyref="#t1"/>
    </lrml:TemporalCharacteristic>
</lrml:TemporalCharacteristics>

```

The author of the rule and the associated authority are also crucial parameters in modeling rules. The blocks `<Agent>` and `<Authority>` are used to define the author and the authority of the rules. They allow representing the provenance and authorial tracking of the rules. The following XML code indicates that the agent `y.taheri` is an author with label `aut1`. In addition, `congress` is the authority that has the type `Legislature`.

```
<lrml:Agents>
    <lrml:Agent key="aut1"
        sameAs="unibo:person.owl#y.taheri"/>
</lrml:Agents>
<lrml:Authorities>
    <lrml:Authority key="congress"
        sameAs="unibo:organization.owl#congress">
        <lrml:type iri="lrmlv:Legislature"/>
    </lrml:Authority>
</lrml:Authorities>

```

### Extensions

As mentioned earlier, LegalRuleML is a general-purpose language to represent legal rules. Hence, it may fail to represent certain concepts that are specific to a domain or regulation. GDPR is the domain of interest in this thesis. It discusses concepts such as data processing, purposes, legal bases, and entities such as controllers and processors that are specific to GDPR. LegalRuleML needs to be extended to support modeling GDPR provinces. A domain-specific ontology built on LegalRuleML to represent and formalize GDPR regulations is PrOnto [151]. PrOnto incorporates concepts like data types and documents, agents and roles, purposes and legal bases, and data processing operations in LegalRuleML. It has been adopted in the DAPRECO knowledge base [19] to formalize GDPR regulations using reified input / output logic [158].

### 1.3.2 The SPECIAL Policy Language

SPECIAL (Scalable Policy-awarE linked data arChitecture for prIvacy, trAnsparency and compLiance) [27] is a formal language based on OWL2 to express consent, business policies, and regulatory obligations. It is designed to automatically check if personal data processing complies with the obligations set forth in the GDPR. In SPECIAL data processing is encoded as business policies and the data subjects consent as usage policies. Policies are modeled as OWL2 classes that are combined by the operations `ObjectUnionOf` or `ObjectIntersectOf`. We briefly describe the approaches to represent personal data processing, data subject's consent, and regulatory norms in SPECIAL.

#### Personal Data Processing

In the SPECIAL policy language, personal data processing is modeled as a business policy. The business policy is a description of the controller's activity. A data processing description encodes a formalized business policy consisting of the following set of features:

- The data to be processed;
- The software that carries out the processing;
- The purpose of the processing;
- The entities that can access The results of the processing;
- The details of where the results are stored and for how long;
- The obligations that are fulfilled while (or before) carrying out the processing;
- The legal basis of the processing.

An example of a personal data processing encoded as OWL2 classes is presented in the following frame.

```
ObjectIntersectionOf(
    ObjectSomeValuesFrom(spl:hasData svd:purchasesAndSpendingHabit)
    ObjectSomeValuesFrom(spl:hasProcessing svpr:Analyze)
    ObjectSomeValuesFrom(spl:hasPurposes vpu:Marketing)
    ObjectSomeValuesFrom(spl:hasRecipient svr:aCompany)
    ObjectSomeValuesFrom(spl:hasStorage
        ObjectIntersectionOf(
        spl:hasLocation svl:EU
        spl:hasDuration svdu:Indefinitely))
    ObjectSomeValuesFrom(sbpl:hasDuty getValidConsent)
    ObjectSomeValuesFrom(sbpl:hasDuty getAccessReqs)
    ObjectSomeValuesFrom(sbpl:hasDuty getRectifyReqs)
    ObjectSomeValuesFrom(sbpl:hasDuty getDeleteReqs)
    ObjectSomeValuesFrom(sbpl:hasLegalBasis A6-1-a-consent)
```

```
15      )
```

Listing 1.1: A business policy in SPECIAL

The attributes of this business policy are described, respectively, as follows. (i) The category of the personal data used in the processing is *Purchases and spending habits*, (ii) the processing category is *analyze*, (iii) The purpose of the processing is *marketing*, (iv) the recipient of the processing result is *aCompany*, (v) the processing is carried out in a storage located in Europe and the duration of data storage is *Indefinite*, (vi) the duties defined for this processing, for example *getValidConsent* means that the specified software can read the data sources if consent has been given, (vii) the legal basis of the processing is *A6-1-a-consent*.

The business policy represents the class of all the operations that the controller may execute. SPECIAL uses the W3C's *Data Privacy Vocabularies and Controls Community Group*, (DPVCG) [152] to assign standard values for the attributes in the policy.

### Consent Representation

According to GDPR, the most important legal basis for the lawful processing of personal data is consent from the data subjects. Consent is represented as a usage policy in SPECIAL that is a description of the processing for which the data subject has given consent. Consent has the same attributes as a business policy, but, without legal basis and duties. For example, consider the following usage policy.

```
1 ObjectIntersectionOf(
2     ObjectSomeValueFrom(has_purpose createPersonalizedRecommendations )
3     ObjectSomeValueFrom(has_data serviceConsumptionBehavior)
4     ObjectSomeValueFrom(has_processing Transfer)
5     ObjectSomeValueFrom(has_recipient aCompany)
6     ObjectSomeValueFrom(has_storage
7         ObjectIntersectionOf(
8             ObjectSomeValueFrom( has_location:EU)
9             DataSomeValueFrom( has_duration DatatypeRestriction(
  ↪ xsd:integerxsd:minInclusive "365" xsd:integer))))
```

Listing 1.2: A consent in SPECIAL

This usage policy represents the consent of a data subject to transfer their *Service Consumption Behavior* data with the purpose *Create Personalized Recommendations*. The admissible recipient is *aCompany*, and the storage must be located in Europe. In addition, the consent is valid for a duration of 365 days.

This usage policy describes the class of all the operations allowed by the data subject. Data processing is compliant with the given consent if it is a subclass of the permitted operations. In other words, a business policy $BP$ is compliant with the given usage policy $P$ if `SubClassOf(BP  P)`.

### Regulatory Norms

SPECIAL offers a partial encoding of GDPR for automated compliance checking. The formalization covers the obligations that concern controllers and processors. The requirements in GDPR that cannot be derived or checked automatically are excluded. GDPR requirements in SPECIAL are encoded as classes and linked using `ObjectUnionOf`, `ObjectIntersectOf`, and `EquivalentClass` axioms. Each class corresponds to a chapter or a set of articles in GDPR. A schematic representation of the classes is shown in Figure 1.2. The encoded formalization part of GDPR is extensive; therefore, we only describe some relevant classes. The full version of the formalization can be accessed on the SPECIAL website [1].

The top-level class that formalizes the GDPR is `GDPR_Requirements`. It contains all the processing that is compliant with the encoded obligations. This class is equivalent to:

```
<!-- GDPR_Requirements: -->
ObjectUnionOf(
    ObjectSomeValuesFrom(spl:hasData ObjectComplementOf(PersonalData)
    )
    ObjectIntersectionOf(
        Chap2_LawfulProcessing
        Chap3_RightsOfDataSubjects
        Chap4_ControllerAndProcessorObligations
        Chap5_DataTransfer
    )
    Chap9_Derogations
)
```

The combination of `ObjectUnionOf` and `ObjectComplementOf` in this encoding implies that "if data are personal, then chapters 2–5 or 9 must be fulfilled". In other words, according to this expression, a data processing is compliant if it involves non-personal data, or the requirements of GDPR Chapters 2–5 are satisfied, or some of the derogation in `Chap9_Derogations` applies.

Business policies that satisfy the regulations in *Chapter 2* of the GDPR are represented by class `Chap2_LawfulProcessing`. The regulations correspond to *Articles 6, 9*, and *10*. According to Art. 6, a business policy should have a legal basis among those specified in this article. A processing that involves sensitive data requires another set of legal basis according to Art. 9. Moreover, Art. 10 specify additional restrictions for processing criminal records. According to these requirements, class `Chap2_LawfulProcessing` is equivalent to:

```
<!-- Chap2_LawfulProcessing: -->
ObjectUnionOf(
    Art6_LawfulProcessing
    Art9_SensitiveData
    Art10_CriminalData)
```

[1] https://specialprivacy.ercim.eu/platform/pilots-policies-and-the-formalization-of-the-gdpr


Figure 1.2: SPECIAL classes for regulatory obligations

A processing satisfies the requirements of Chapter 2 if the obligations of *Article 6 (Lawful Processing)*, *Article 9 (Sensitive Data)*, or *Article 10 (Criminal Data)* are fulfilled.

The next class is `Art6_LawfulProcessing`, which is defined as:

```
<!-- Art6_LawfulProcessing: -->
ObjectUnionOf(
    ObjectSomeValuesFrom(spl:hasData
        SensitiveData_as_per_Art9
    )
    ObjectSomeValuesFrom(spl:hasData
        CriminalConvictionData_as_per_Art10
    )
    Art6_1_LegalBasis
    Art6_4_CompatiblePurpose
    )
)
```

This expression means that, when the data involved in the processing are neither sensitive nor criminal conviction data, then the fundamental legal bases of *Art. 6(1)* applies, or the processing is compatible with the original purpose of collecting the data, as per *Art. 6(4)*.

The class `Art6_1_LegalBasis` is equivalent to:

```
<!-- Art6_1_LegalBasis: -->
ObjectSomeValuesFrom( hasLegalBasis
    ObjectUnionOf(
        Art6_1_a_Consent
        Art6_1_b_Contract
        Art6_1_c_LegalObligation
        Art6_1_d_VitalInterest
        Art6_1_e_PublicInterest
        Art6_1_f_LegitimateInterest
    )
)
```

This expression means that a business policy fulfills the requirements of *Art. 6(1)* if it contains a clause `ObjectSomeValueFrom( hasLegalBasis X )`. In this clause, $X$ corresponds to a class among points *a–f* of *Art. 6(1)*.

The next class that we describe here is `Chap5_DataTransfer`. This class contains business policies that meet the requirements of Chapter 5 of GDPR. According to these requirements, a transfer of personal data to a *third country*, i.e. a non-European country, is permissible if the third country guarantees an adequate level of protection (cf. *Article 45*). In the case where the storage or recipients are in a country, off the list in *Article 45*, then a transfer to a third country is lawful only if the controller or processor has provided *appropriate safeguards* (cf. *Article 46*). In addition, if *Art. 45* and *Art. 46* do not apply, a transfer is only permissible if one of the derogations listed in *Article 49* applies. These requirements are encoded in OWL2 in the following way:

```
<!-- Chap5_DataTransfer: -->
ObjectUnionOf(
    ObjectIntersectionOf(
        Art48_TransfersNotAuthorisedByUnionLaw
        ObjectUnionOf(
            AdequateLevelOfProtection_as_per_Art45
            AppropriateSafeguards_as_per_Art46
            Art49_Derogations
        )
    )
    ObjectComplementOf(
        ObjectIntersectionOf(
            ObjectSomeValuesFrom(spl:hasProcessing svpr:Transfer)
            ObjectSomeValuesFrom(spl:hasStorage
                ObjectSomeValuesFrom(spl:hasLocation svl:ThirdCountries)
            )
        )
    )
)
```

The class `AppropriateSafeguards_as_per_Art46` in this encoding formalizes the list of appropriate safeguards stated in *Article 46*.

```
<!-- AppropriateSafeguards_as_per_Art46 -->
Art46_2_a_PublicAuthorities
Art46_2_b_BindingCorporateRules_as_per_Art47
Art46_2_c_DataProtectionClausesAdoptedByEC
Art46_2_d_DataProtectionClausesAdoptedBySupervisoryAuthority
Art46_2_e_ApprovedCodeOfConduct
Art46_2_f_ApprovedCertificationMechanism
Art46_3_a_ContractualClauses
Art46_3_b_ProvisionsInAdministrativeArrangements
```

This definition means that a business policy satisfies the requirements of *Art. 46* if it contains a clause $X$ that represents one of the points a–f of *Art. 46(2)* or a-b of *Art. 46(3)*.

## 1.4 Conclusions

The main objective of this thesis was to introduce knowledge representation methods used in the legal domain. We also discussed the ontologies to represent GDPR requirements, in particular the SPECIAL policy language. LegalRuleML is a language that represents necessary concepts in regulations and uses it for legal reasoning. i.e., inferring what obligations, permissions, or prohibitions may hold in a specific situation. LegalRuleML covers a variety of concepts that are essential to properly represent the requirements in law text. However, not all of these concepts apply to automatic compliance checking. In contrast, SPECIAL is a simpler policy language for representing requirements that can be automatically checked.

Depending on the domain and the level of legal coverage, a representation approach can be adopted. Our specific domain concerns the legal compliance of plans that involve operations on personal data. We are interested in modeling constraints to check whether an action is allowed or not. Hence, a representation of rules in deontic format is not necessary in our domain. SPECIAL only models policies as operations that are allowed by GDPR instead of deontic rules. In addition, it covers the obligations of the data controllers and processors that we cover in our compliance checking model in Chapter 4.

# Chapter 2

# State of the Art: Ethical Compliance

In this chapter, we discuss the ethical aspects of our research. We are interested in (i) the ethical issues of AI particularly in relation to processing of personal data, (ii) moral judgment and reasoning from the point of view of ethics philosophy, and (iii) modeling and integrating of morality in AI.

Ethics is the study of what is right and wrong in human behavior and reasoning. It involves analyzing and proposing theories of ethics and morality, and examines how individuals should act and why. Ethics is essential to guide human behavior and influences how laws are formed and how society functions. *Applied ethics* is a practical application of ethics in specific contexts such as bioethics, environmental ethics, and AI ethics. For example, applied ethics might explore the implications of conducting medical research on humans or the ethics of animal rights. In case of AI, there are many guidelines that discuss the ethical issues specific to AI. They discuss the risks that AI poses to fundamental values and principles. Guidelines are necessary for the ethical development and deployment of AI systems. We explore these ethical guidelines and discuss the values and principles that are at risk from AI. We particularly emphasize on guidelines related to processing of personal data.

*Normative Ethics* is another branch of ethics that focuses on establishing the standards for right and wrong behavior. It includes the study of ethical theories such as utilitarianism, deontology, and virtue ethics. They propose different criteria for evaluating morality of actions. These theories are essential to understand moral judgment and reasoning. In addition, *Computational ethics* is the field that studies modelling and the integration of moral reasoning in AI agents. we explore this field and outline some of the challenges and general approaches in designing ethical AI.

This chapter is organized as follows. In Section 2.1 we discuss the ethical issues related to AI, particularly in relation to processing personal data. Section 2.2 presents the theories in normative ethics that are used for moral judgement and reasoning. Finally, in Section 2.3, we explain computational ethics and the challenges in the development of ethical AI.

## 2.1 Ethics of AI

The Ethics of Artificial Intelligence refers to moral considerations and implications in the development, deployment, and use of AI systems. As AI systems increasingly perform tasks traditionally done by humans, including decision-making in various contexts, ethical concerns about their impact on individuals and society become crucial. The ethics of AI include a wide range of issues, from individual rights and privacy to broader societal impacts and responsibilities. Many AI ethics guidelines [117] that address these risks have been developed by stakeholders, including governments, industry organizations, and research institutions. Guidelines ensure that AI systems are developed and used in a way that is ethical, fair, transparent, accountable, and aligned with societal values and human rights. We mention some of the important ethical issues of AI and outline the principles that are at risk according to the guidelines[102, 117].

AI systems can be vulnerable to attacks and exploitation, posing risks to the security and safety of individuals, such as adversarial attacks [42], data poisoning [98], and model inversion [65]. The guidelines state that AI should never cause foreseeable or unintentional harm [4, 80]. Harm is interpreted primarily as discrimination [45], violation of privacy [45], or bodily harm [181, 4]. Moreover, AI systems, particularly in the context of natural language processing [70] and content generation [39], can be used to create and propagate misinformation and manipulate public opinion, which endangers freedom and autonomy of users.

Furthermore, many AI algorithms, especially those based on deep learning and neural networks, operate with limited transparency and explainability. This can make it difficult to understand and trace the decisions made by AI algorithms by human operators, raising concerns about accountability and trust. Transparency is particularly important in applications that include data use, human-AI interaction [50, 100, 4], automated decisions, and the purpose of data use or the application of AI systems [110, 40]. In AI guidelines, transparency is presented mainly as a way to minimize harm and improve AI [182, 45, 189] to emphasize its benefit for legal reasons [110], or to improve trust [100, 95].

Although many of the principles mentioned are interrelated, they are not the only principles that are addressed in AI ethics guidelines [117]. Other values or principles that are recommended to develop trustworthy AI are beneficence [35, 51, 171, 194], trust [66, 35], sustainability [16, 51], and dignity [20, 194, 85].

The specific domain of our research is the processing of personal data. Using personal data may harm the privacy and harm of data subjects. In addition, certain processing has the risk of being biased toward a certain group. Hence, we explain the ethical issues related to privacy and bias in AI more specifically.

### 2.1.1 Privacy

AI algorithms such as deep learning often rely on large amounts of personal data for training and inference. They can be used to collect personal data, e.g. surveillance, or derive some personal data that are not authorized by data subject, e.g. profiling. These potential risks of AI raise concerns about privacy violations and data misuse. In AI ethics guidelines, privacy is often viewed as a value to promote [45, 125], and as a right to be protected [50, 182, 62]. Privacy concerns are also discussed in relation to data protection and data security [101, 80] and data security. Unauthorized access, data breaches, and inadequate data protection measures can compromise individual privacy rights.

According to Ethics and Data Protection [97], published by the European Commission, a processing operation may have higher ethical risks if it involves:

- Processing of 'special categories' of personal data. They include, racial or ethnic origin political opinions, religious or philosophical beliefs, genetic, biometric or health data, sex life or sexual orientation;
- Processing of personal data concerning children, vulnerable people or people who have not given their informed and explicit consent for processing;

- Complex processing operations and/or the processing of personal data on a large scale and/or systematic monitoring of a publicly accessible area on a large scale;
- Data processing techniques that are invasive e.g. surveillance, web crawling, profiling, that pose a risk to the rights and freedoms of research participants, or techniques that are vulnerable to misuse; and
- Collecting data outside the EU or transferring personal data collected in the EU to entities in non-EU countries.

In addition, this document discusses some of the measures to mitigate or avoid ethical risks and achieve data protection by design. For example, *Data Minimization* implies that data processing should involve only data that are necessary and proportionate to achieve the specific task or purpose for which they were collected (*Article 5(1)* GDPR). It applies not only to the amount of personal data collected, but also to the extent to which they may be accessed, further processed, and/or shared, the purposes for which they are used, and the period for which they are kept. Another way to mitigate the ethical concerns that arise from the use of personal data is to anonymize them so that they no longer relate to identifiable persons. *Anonymisation* involves techniques that can be used to convert personal data into anonymized data.

### 2.1.2 Fairness

AI algorithms, particularly those based on machine learning approaches, can exhibit biases based on the data on which they are trained, leading to discriminatory results. This can perpetuate or amplify existing societal biases and inequalities, particularly in areas such as hiring, lending, and law enforcement. The guidelines address concerns about algorithmic bias and discrimination, focusing on the need for diversity [62, 184], inclusion [15] and equality [15, 184] and fairness in AI systems. Fairness is often associated with justice [50, 181, 100, 133] and the prevention or mitigation of unwanted bias [50, 133] and discrimination [50, 45, 15, 62].

Fairness is a wide topic and applies to a variety of AI applications that include algorithmic decision-making processes. Fairness also depends on the context in which it is used. A significant area of application are the recommender systems [118]. For example, consider a job recommendation system. A fair system should not be biased toward any group of individuals, such as sex or race, when proposing job opportunities. This view of fairness is based on *User Fairness*, that implies, a fair recommender system for users should treat different predefined groups of users or similar individual users fairly. Fairness requirements in recommender systems are not limited to the user-side, i.e., users who receive recommendations. another view over fairness is the item-side, i.e. refers to the items to be ranked or recommended. This notion is called *Item Fairness*, implying that a fair recommender system for items should treat different predefined groups of items or similar individual items fairly. The fairness

demands from user-side are usually about the quality of the recommendations for them, while the fairness considerations from item-side usually focus on the exposure opportunity of items in the ranking lists. Hence in our job recommender example, the system is not also expected to be fair with respect to different racial or gender groups, it should also be fair with regard to jobs that are being recommended.

## 2.2 Normative Ethics

Normative ethics is a branch of ethical philosophy that investigates morally right and wrong behavior and provides established ethical theories for ethical reasoning and justifying moral choices. The 3 principal theories in normative ethics that propose different criteria for what makes actions morally right or wrong are Consequentialist Ethics, Deontology, and Virtue Ethics.

### 2.2.1 Consequentialist Ethics

Consequentialism is a class of normative ethical theories that states that the consequences of an action are the ultimate basis for judgement about its rightness or wrongness. Thus, from a consequentialist point of view, a morally right act is one that produces a good result. Consequentialists hold, in general, that an act is right if and only if the act produces a greater balance of good over bad than any available alternative. Different consequentialist theories differ in how they define moral goods.

*Hedonism*, proposed by Bentham [22], claims that pleasure is the only intrinsic good and that pain is the only intrinsic bad. *Hedonistic utilitarianism* holds that what matters is the aggregate happiness; the happiness of everyone. According to this theory, all action should be directed toward achieving the greatest total amount of happiness. Bentham believed that the value of a pleasure could be quantitatively understood and introduces a method of measuring the value of pleasures and pains, via *Hedonic Calculus*. This method, calculates the value of pleasure or pain according to variables like intensity, duration, uncertainty and remoteness. In contrast, John Stuart Mill [134], argue a qualitative approach. Mill proposed a hierarchy of pleasures, where higher quality pleasure is more valued than lower quality pleasure.

Some contemporary utilitarians are concerned with maximizing the satisfaction of preferences [31, 170, 88], hence *preference utilitarianism*. Preference utilitarianism holds that an individual's well-being depends on the level of satisfaction of his or her informed self-regarding preferences. It involves promoting actions that fulfill the preferences of those beings involved [170]. The concept of preference utilitarianism was first proposed by John Harsanyi [89] and is mainly advocated by concept and is more commonly associated with R. M. Hare, [88] Peter Singer, [170] and Richard Brandt.[31]

Furthermore, many consequentialists deny that all values can be reduced to any single ground, such as pleasure or desire satisfaction, so they instead adopt

| Theory | Promoted Value |
|---|---|
| Hedonism [22, 169, 134] | Pleasure |
| Preference Utilitarianism [89] | Satisfaction of Preference |
| Ideal Consequentialism [142] | Pluralistic Values |

Table 2.1: Promoted values in consequentialist theories

a pluralistic theory of value. Moore's *ideal utilitarianism* [142], for example, takes into account the values of beauty and truth (or knowledge) in addition to pleasure.

### 2.2.2 Deontological Ethics

Deontology argues that the morality of an action is determined by whether the action itself it complies with a set of duties or rules. The rightness or wrongness of an action does not depend on its consequences, but on whether it fulfills a duty. The outcomes or results of these actions are secondary to the importance of following one's moral duty. This principle contrasts sharply with consequentialist theories, such as utilitarianism, which prioritize the outcomes of actions in determining their moral worth. Deontological theories often adopt a form of moral absolutism, arguing that certain moral principles should be obeyed without exception. For example, if lying is wrong, deontological ethics would hold that one should not lie, even if lying would bring about bad consequences. For Kant and many deontologists, the moral value of an action is determined by the agent's intention and the good will behind it. Acting out of good will means doing something because it is the duty and for no other reason. The morality of an action, therefore, depends not on what happens as a result but on the moral intention behind it. Deontological theories may vary according to their foundational principles, the nature of the duties they emphasize, and the sources of those duties.

*Kantian Deontology* [105] holds that morality is grounded in reason and that certain *categorical imperatives* (universal moral laws) must be followed unconditionally. Kant's most famous imperative, the *Formula of Universal Law*, states that one should only act according to *maxims* that could be universally applied. Essentially, one should only act in a way that one would want everyone else to act in a similar situation. Another formulation of is the *Formula of Humanity*, which mandates treating individuals as ends in themselves, not merely as means to an end. This reflects a fundamental respect for the dignity of each person, which should not be violated even if doing so would benefit others.

Ross, in contrast, proposes that there is a plurality of *prima facie* duties that determine what is right [161]. Some duties originate from our own previous actions, like the duty of fidelity (to keep promises and to tell the truth), and the duty of reparation (to make amends for wrongful acts). The duty of gratitude (to return the kindness received) arises from the actions of others. Other duties include the duty of non-injury (not to hurt others), the duty of beneficence (to promote the maximum of aggregate good), the duty of self-improvement

(to improve one's own condition), and the duty of justice (to distribute benefits and burdens equably). Ross argues that these duties are not absolute but prima facie; they are binding unless they conflict with a stronger duty in a particular situation. One problem the deontological pluralist has to face is that cases can arise where the demands of one duty violate another duty, i.e. moral dilemmas. Some deontologists believe in the *Divine Command Theory*, which states that an action is right if God has commanded it to be right [192]. This type of deontology bases moral duty on religious texts and beliefs, making it distinct in that its moral laws are derived from a supernatural source.

### 2.2.3 Virtue Ethics

Virtue ethics, derived from the philosophical traditions of Aristotle and Plato, is a moral theory that treats virtue and character as the primary subjects of ethics, in contrast to rules (as in deontology) or consequences (as in consequentialism) [173]. A virtue is a characteristic disposition to think, feel, and act well in some domain of life [99]. Virtue ethics is concerned with the development of virtues that should be embodied by people to live and act morally, such as courage, temperance, wisdom, and justice. According to virtue ethics, ethical actions cannot be separated from the character of the individual who performs them.

Virtue ethics often aims at the achievement of *eudaimonia*, translated as flourishing or well-being. This concept is about living well and fulfilling one's potential. Aristotle proposes that living virtuously is necessary and central to achieving eudaimonia, which is considered the highest good for humans. Additionally, *Phronesis*, or *Practical Wisdom*, is an important virtue in Aristotelian ethics, which refers to the ability to make the right decision in complex situations. Practical wisdom aims at balancing different virtues in accordance with the right reasons and the appropriate emotions. Unlike deontology and consequentialism, virtue ethics is fundamentally agent-centered. It focuses on what kind of person one should be, rather than exclusively on what actions one should perform. Some modern versions of virtue ethics define virtues as traits that tend to promote some other good that is defined independently of virtues, thereby merging virtue ethics with consequentialist ethics [179].

### 2.2.4 Value Pluralism

In many cases, we refer to a set of values in order to discuss ethical issues. For example, AI ethics guidelines refer to certain values such as privacy, fairness, transparency, etc. that should be promoted and respected in the design and development of AI systems (cf. Section 2.1). In this section, we discuss how several values are incorporated into moral theories. The question about pluralism is whether different values are all reducible to one supervalue, or whether they are several distinct values. Pluralists argue that there are really several different irreducible values. Monism is the opposite view, which holds that there is only one ultimate value. Moral theories can be pluralistic according to their fundamental view of values. Consequentialists view values as what bring goods

in the world, such as friendship, knowledge, beauty, etc. Consequentialist theories can be pluralist if they hold that there are many fundamental goods, e.g. ideal utilitarianism. In contrast, monist utilitarians believe that there is only one fundamental value and that is well-being or pleasure or happiness, as in hedonism. They consider values such as friendship, knowledge, and so on as instrumental values that contribute to the foundational value. For deontologists, morality is based fundamentally on moral principles. Pluralist deontological theories presume that there are multiple fundamental principles. For example, Ross' theory is pluralist, as it presumes a plurality of prima facie duties. By contrast, Kant was a monist, as he believed that there is one universal principle from which all other principles are derived.

An issue in making a rational choice based on irreducibly plural values is *incommensurability* [84, 43]. Incommensurability is the lack of a common unit of value by which precise comparisons can be made [128, 43]. This implies that values cannot be represented on a cardinal scale, which poses challenges for normative theories. Especially utilitarian theories that aggregate values by weighing and summing utilities. However, incommonsurability of values does not imply that comparisons are impossible. Two items are "incomparable", if there is no possible relation of comparison, such as 'better than', or 'as good as' [43]. It is possible to say that one thing is better than the other, while it is impossible to measure how much better it is.

## 2.3 Ethical AI

Ethical AI refers to AI systems that act and behave ethically. It is associated with computational ethics that deals with the moral behaviors of Artificial Moral Agents (AMAs). Computational ethics involves the design and implementation of AI systems that can make ethical decisions. It is concerned with ensuring that automated systems act in ways that are beneficial, fair and just [141]. Ethical AI is distinguished from the ethics of AI, which involves the ethical principles, rules, guidelines, policies, and regulations related to AI. Ethical AI focuses on creating AI systems that inherently follow ethical principles, while ethics of AI emphasizes the moral obligations of AI creators and the impacts of AI on society. The distinction between the two concepts in interactions with AI, humanity and society is shown in Table 2.2.

### 2.3.1 Artificial Moral Agency

Traditional ethical theories often presuppose certain capabilities, such as consciousness, intentionality, and free will, which are disputed when applied to machines. Moral agency involves the ability of an agent (an entity that can instantiate intentional mental states capable of performing actions) to make free choices, deliberate about what one should do, and understand and apply moral rules correctly in paradigm cases [96]. AI systems, especially advanced ones like autonomous vehicles, operate with a degree of autonomy in the sense that

| | Ethics of AI | Ethical AI |
|---|---|---|
| AI | Principles of developing AI to interact with other AIs ethically | How AI should interact with other AIs ethically? |
| Human | Principles of developing AI to interact with humans ethically | How AI should interact with humans ethically? |
| Society | Principles of developing AI to function ethically in society | How AI should operate ethically in society? |

Table 2.2: The distinction between Ethics of AI and Ethical AI [168]

they can perform tasks, make decisions, and respond to environments without human intervention. However, this type of autonomy is largely constrained by their programming and the parameters set by their developers.

Himma claims that moral agents need to have conscious and intentionality, something that state-of-the-art systems do not seem to instantiate [96]. Sparrow points out that while AI can be programmed to follow certain ethical guidelines or make decisions based on predicted outcomes, these machines will never truly "be ethical" because they lack the capacity for personal moral judgment and the ability to understand or engage with ethical dilemmas genuinely [172].

In legal and ethical discussions, AI is generally not considered to have an agency similar to that of humans. The lack of agency in AI has significant implications for how responsibility and accountability are assigned in cases where AI systems cause harm or operate unpredictably. Legal systems attribute liability and responsibility to the humans behind AI systems (developers, users, and corporations) rather than to the AI systems themselves.

### 2.3.2 Classification of AMAs

Regardless of the debates on the possibility of artificial moral agents, a relevant question is to what extent the moral decision-making process can be understood and modeled or simulated by machines. Even the most complex artificial systems differ from human beings in important respects that are central to our understanding of moral agency[137]. It is therefore common in machine ethics to distinguish between different types of moral agents depending on how highly developed their moral capacities are. Two well-known taxonomies of AMAs have been proposed in the literature, Wallach and Moor [141].

Wallach and Allen introduce a framework for understanding the progression from the current state of AI to sophisticated artificial moral agents[185]. This framework considers two independent dimensions: autonomy and sensitivity to values. Figure 2.1 illustrates the positions of various systems based on their autonomy and ethical sensitivity:

- **High Autonomy, Low Sensitivity**: Systems such as autopilots operate autonomously within specific domains, but lack ethical reasoning

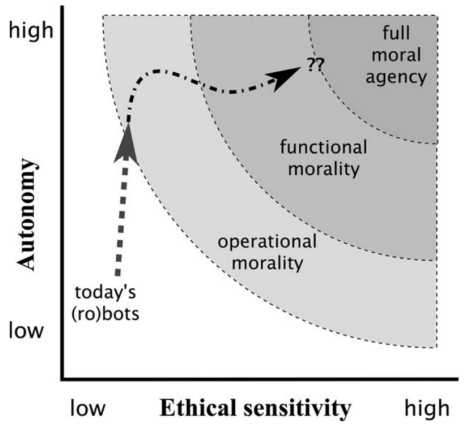



Figure 2.1: Two dimensions of AMA development [185]

capabilities. These systems have significant autonomy, but limited ethical sensitivity. They are engineered to respect values such as safety and passenger comfort, achieving these through precise monitoring and control mechanisms that limit their operational parameters to ensure ethical outcomes.

- **Low Autonomy, High Sensitivity**: Systems such as ethical decision support systems provide guidance based on ethical principles, but lack the autonomy to act independently. These systems provide decision-makers with access to morally relevant information, helping to make ethical decisions without autonomous action. An example is MedEthEx [9], a medical ethics expert system that helps clinicians choose ethically appropriate courses of action by analyzing specific cases through a series of questions.

The intersection of these dimensions illustrates the varying degrees of moral capability that machines might possess, ranging from basic operational morality to fully autonomous moral agents.

- **Operational Morality**: Operational morality refers to machines designed with specific ethical considerations in mind, despite lacking autonomy or ethical sensitivity. An example provided is a gun with a childproof safety mechanism. Such designs do not make ethical decisions but are created with values that enhance safety and prevent harm.

- **Functional Morality** Between the basic operational morality and fully realized moral agency, there lies a spectrum of functional morality. This encompasses systems that may have significant autonomy but limited ethical sensitivity, like autopilots, and those with minimal autonomy but enhanced capability to handle ethical considerations, like ethical decision support systems.

The other taxonomy of AMAs in the machine ethics community is proposed by Moor[141]. This taxonomy is based on the ethical impact and moral reasoning abilities of agents [141]. Different levels at which machines might function as moral agents are mentioned, ranging from simple ethical impact agents to sophisticated explicit and full ethical agents.

#### Ethical-Impact Agents

Machines as tools can impact ethical outcomes in significant ways. Moor argues that even without being fully ethical agents, machines as tools can have a substantial ethical impact by replacing harmful human practices.

#### Implicit Ethical Agents

Moor explains that machines can be implicit ethical agents by being designed to avoid unethical outcomes. This can be achieved not through an explicit understanding of ethics, but through constraints and design features that promote ethical actions. For example, banking software that handles transactions accurately and reliably can be seen to practice a form of "implicit ethics."

#### Explicit Ethical Agents

The concept of explicit ethical agents involves machines that can understand and process ethical principles similar to how they handle other types of data. Moor discusses the potential for machines to be programmed with complex ethical reasoning models, such as those integrating deontic logic (rules of duty) and epistemic logic (rules of knowledge).

#### Full Ethical Agents

Moor speculates on the possibility that machines become full ethical agents, which would involve not only ethical reasoning abilities, but also traits typically associated with human moral agents, such as consciousness and free will. He acknowledges that this is a controversial and challenging idea as it touches on deep philosophical questions about the nature of consciousness and morality.

### 2.3.3 Computational Ethics Challenges

According to Moor's terminology, the primary objective in machine ethics is to develop a machine that functions as an explicit ethical agent [6]. The complex nature of ethics makes it challenging to clearly specify what a moral behavior is and how it can be modeled computationally. James Gips proposed that creating an ethical robot should be considered a major computing Grand Challenge [79].

The difficulties in machine ethics can be divided into two main categories, namely philosophical and practical challenges [6]. The philosophical challenges primarily concern the nature of ethics itself and whether it can be adequately

represented and processed by machines. Ethics involves subjective judgment, cultural and situational nuances, and often a level of moral reasoning that might require consciousness or emotions, qualities traditionally believed to be exclusive to humans. On the practical side, the challenges relate to the actual implementation of ethical principles in AI systems.

### Philosophical Challenges

The first issue in developing machine ethics is to know "what constitutes a moral behavior?". In general, there is no consensus on what specifies moral behavior. The existence of universally acceptable ethical standards is the subject of debate between moral relativism and moral absolutism. Different cultures and societies have different moral values, and even within a single culture, there are multiple ethical theories (e.g., utilitarianism, deontology, virtue ethics) that can lead to different conclusions about what is ethical. Creating systems that incorporate this diversity and behave ethically in all cultural contexts is a significant challenge. In addition, the time-varying nature of values makes the development of moral agents even more challenging. This requires systems to be adaptable and update their ethical frameworks accordingly.

Ethics inherently involves judgments that are highly contextual and need to be interpreted in complex, often ambiguous situations. This subjectivity makes it challenging to create a set of fixed rules that can guide AI behavior in all possible scenarios. Even if such rules exist, automatic systems might encounter situations that were not anticipated by their designers, requiring them to make decisions without clear ethical guidelines. Moreover, different ethical frameworks can provide conflicting guidance in the same situation that leads to ethical dilemmas. Balancing these conflicts in a computational model is challenging. Sparrow, for example, argues that due to the deeply personal and contextual nature of ethics and its ties to the life history and moral character of individuals, ethics cannot be built into AI through programming or learning from ethical data [172].

### Practical Challenges

Formalizing abstract ethical principles into concrete computational algorithms is a major challenge. This involves defining ethical principles in a way that can be quantified and understood by machines. James H. Moor explores the complex relationship between ethical decision-making and computational methods [140]. He evaluates past attempts by philosophers like Jeremy Bentham to calculate ethical decisions based on utilitarian principles. Moor suggests that, while certain aspects of ethical decision making might be computable, the complexity of human values and the contextual nuances of ethical situations pose significant obstacles.

In addition, when it comes to moral behavior, the process of making a certain decision is as important as the decision itself. This implies that the underlying decision-making process must be transparent and understandable to

humans. Many advanced AI models operate as "black boxes" where their internal decision-making processes are not transparent or fully interpretable. This lack of transparency makes it difficult to understand and trust their ethical decisions. This is even more crucial when it comes to the development of ethical decision processes. Ensuring that AI systems can explain their ethical decisions in a way that humans can understand is crucial for accountability and trust. This poses a further challenge for moral machine development.

### 2.3.4 Implementation Approaches

There are several approaches to developing explicit ethical agents. These approaches vary depending on the domain to which they are applied and the challenges they address. Wallach et al. [186] propose a widely used categorization of approaches in machine ethics. They distinguish three types of implementation approaches, namely bottom-up, top-down, and hybrid.

#### Bottom-up

Bottom-up approaches focus on building systems that achieve specified goals or standards through experiential learning from samples. These systems do not rely on any ethical theory but can develop moral capacities through learning with their environment. Bottom-up systems can adapt to new and unforeseen situations, potentially developing sophisticated behaviors through continuous learning. This approach mirrors the way humans learn and refine moral behavior through experience. Some AI methods relevant to this approach are genetic algorithms, neural networks, and reinforcement learning. A shortcoming of this approach is that the decision process is often a "black box", i.e. not explainable for humans. as a result, there is a risk that the machine learns the wrong rules. Another difficulty is the lack of reasoning capacity to handle complex moral decisions.

#### Top-down

Top-down methods, which are also adopted in this thesis, involve the explicit implementation of ethical theories such as utilitarianism and deontology in AI systems. This approach relies on specific moral rules or principles for ethical evaluation and solving dilemmas. Top-down approaches rely on structured frameworks for moral reasoning. One of the significant issues is the inherent complexity and often conflicting nature of ethical rules. Implementing these rules in a way that allows for decision-making in diverse contexts is challenging.

#### Hybrid

A hybrid approach that combines elements of both top-down and bottom-up methodologies. Effective AI systems should be able to evaluate actions using top-down principles, while also being able to adapt and learn from bottom-up

experiences. In certain cases, this approach is necessary to cover all requirements of machine ethics.

## 2.4 Conclusions

In this section, we discuss the ethical aspects of our research objective. We briefly mentioned some of the ethical issues applied to AI according to AI guidelines. We discussed privacy and fairness concerns with regard to the processing of personal data. In addition, we explained moral theories for ethical judgments that are studied in normative ethics. These theories are essential in understanding moral reasoning. We then explored moral decision making or the ability to act morally from the point of view of an AI agent. We described some of the challenges, taxonomies, and approaches in computational ethics. our aim in this thesis is to develop an explicit agent capable of moral reasoning. We are interested in approaches top-down based on logical reasoning. We adopt ideas in normative ethics, in particular utiltarianist theories, to design a framework for moral evaluation in Chapter 5. We adopt a pluralistic view, use AI guidelines to represent values, and compare alternatives. This framework is demonstrated in comparing several processing on personal data that concern privacy and fairness.

# Chapter 3

# State of the Art: Modeling Tools

In this chapter, we describe the tools and methods that we adopted in our research. As mentioned earlier, we are interested in approaches based on logical reasoning. We represent and model our problem in logic programming. Throughout the research, we used ASP and Prolog as knowledge representation and reasoning tools. In addition, we used logical frameworks for planning and modeling agents of action. Aggregation methods are discussed in a wide variety of areas with different approaches and applications. In our research, we are interested in approaches for aggregating orders or preference relations. A suitable candidate that we adopted is voting systems.

We describe logic programming in Section 3.1, and discuss two well-known paradigms, namely Prolog and ASP. In Section 3.2 we explain the planning methods adopted in our research; Event Calculus and Hierarchical Task Network (HTN). Finally, Section 3.3, presents a brief overview of voting systems and their properties.

## 3.1 Logic programming

Logic programming is a programming paradigm based on formal logic, particularly first-order logic. In logic programming, programs are written as a set of logical statements, and computation is performed through logical inference. This approach allows for a high level of abstraction, making it suitable for problems involving complex relationships, rules, and symbolic reasoning.

Logic programming has a wide variety of applications across different domains, including Artificial Intelligence, database systems, and theorem proving. Specifically in Artificial Intelligence, logic programming is used for knowledge representation, reasoning, natural language processing, and expert systems. Due to the declarative nature, logic programming languages are highly expressive for problems involving complex relationships, rules, and facilitate reasoning about the program and proving properties about it. The major logic programming languages include Prolog and Answer Set Programming (ASP).

### 3.1.1 Prolog

Prolog was one of the first logic programming languages with its roots in first-order logic. It was developed and implemented in Marseille, France, in 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses [112]. The name originates from programmation en logique, French for programming in logic. In this section, we briefly introduce the basic concepts of programming in Prolog. A more detailed introduction to Prolog can be found in [47, 32]

**Terms** Syntactically, all data objects in Prolog are terms. They are inductively defined. A term is of one of the following forms:

- A variable in Prolog is a string of letters, digits, and underscores _, beginning either with an uppercase letter or with an underscore. As in

first-order logic, they are logic variables that are placeholders for arbitrary terms. The occurrences of the name of an ordinary variable stand for the same variable within one clause.

- An atom is a symbol name that starts with a lowercase letter or guarded by quotes and is used to denote predicates. After the initial lowercase letter, they can include digits and the underscore char `_`.
- A number. Fractional numbers use `.` as decimal point. Supported sizes and precisions depend on the Prolog system. Most of the major Prolog systems support arbitrary length integer numbers.
- A compound term has the form $a(t_1, \ldots, t_n)$, where $a$ is an atom called functor name, and $t_1, \ldots, t_n$ are the comma-separated list of argument terms, which are enclosed in parentheses. The number of arguments is called the term's arity. It is greater than or equal to 1, as in the case of $n = 0$ the compound term, collapses to the atom $a$. The functor of the compound term of arity $n$ is denoted as $a/n$. Lists and Strings are special cases of compound terms; A List is an ordered collection of terms. It is denoted by square brackets with the terms separated by commas, or in the case of the empty list, by `[]`. A string is a sequence of characters surrounded by quotes, that is equivalent to either a list of (numeric) character codes, a list of characters (atoms of length 1), or an atom.

**Clauses** A Prolog program consists of a finite set of definite clauses. Relations are defined by means of clauses in Prolog programs. A definite clause is characterized by a single element in the positive head. A clause can be either a rule or a fact. A rule in Prolog is of the following form:

```
Head :- Body.
```

That is read as "Head is true if Body is true". `Head` is a predicate, that is, just `p` in the case of a predicate with zero arguments or `p(t1,...,tn)` in the case of a predicate with a functor of `p/n`. The `Body` of a rule consists of calls to predicates, which are called the goals of the rule. It may contain the built-in logical operator `,/2` that denotes the conjunction of goals and `;/2` that denotes the disjunction. Conjunctions and disjunctions can only appear in the body, not in the head of a rule. The full stop `.` denotes the end of the clause.

A clause whose body is known to be always true is called a fact. In this case, the body of the clause is empty, and it is written as follows:

```
Head.
```

The collection of clauses with the same name and arity in the head defines a predicate that is denoted by `name/arity`. A logic program is a set of predicates. From a syntactical point of view, a predicate is just a term, either an atom in the case of zero arguments or a compound term.

In addition to facts and rules, Prolog programs can also contain directives. A directive is a rule with an empty Head, i.e., it is of the following form:

```
:- Body.
```

**Backtracking** The execution of a Prolog program begins when a single goal, known as the query, is posted. Prolog uses backtracking to find all possible solutions to a query. If a path of reasoning fails, Prolog will backtrack to the previous decision point and try another path. From a logical perspective, the Prolog engine attempts to find a resolution refutation of the negated query. The resolution method used by Prolog is called SLD resolution [68]. If the negated query can be refuted, then the query with the correct variable bindings is a logical consequence of the program. In that case, all generated variable bindings are reported to the user, and the query is said to have succeeded.

**Negation** The built-in Prolog predicate `\+/1` provides negation as failure, which allows for non-monotonic reasoning. In order to deal with negation as failure, Prolog uses the SLDNF resolution [11] which is an extension of SLD. The goal `\+ illegal(X)` in the rule

```
legal(X) :- \+ illegal(X).
```

is evaluated as follows: Prolog attempts to prove `illegal(X)`. If a proof for that goal can be found, the original goal (that is, `\+ illegal(X)`) fails. If no proof can be found, the original goal succeeds. Therefore, the `\+/1` prefix operator is called the "not provable" operator, since the query `?- \+ Goal.` succeeds if the `Goal` is not provable. This kind of negation is sound if its argument is "ground" (i.e., contains no variables). Soundness is lost if the argument contains variables and the proof procedure is complete. In particular, the query `?- legal(X).` now cannot be used to enumerate all things that are legal.

Currently, there are many implementations of the programming language Prolog [156]. A well-known Prolog system is SWI-Prolog [191], which is commonly used for knowledge representation, reasoning, and semantic web applications. Prolog has many applications in solving AI-related problems, particularly problems requiring symbolic representation and manipulation. It has been used for theorem proof [174], expert systems[132], and automated planning [165], as well as natural language processing[154, 114].

### 3.1.2 Answer Set Programming

Answer set programming (ASP) [119, 18, 75, 71] is a form of declarative programming that is used to solve complex combinatorial search problems. It is rooted in logic programming and non-monotonic reasoning, and it is particularly useful in knowledge-intensive applications and problems where the solution space needs to be explored systematically. ASP relies on the semantics of the stable model (answer set) of logic programming [76], which is used to examine the negation as failure. In ASP, search problems are reduced to computing stable models, and answer set solvers, which are designed to generate these stable

models, are used to perform the search.

We define the semantics of an answer set based on Gelfond [74] A rule of Answer Set Programming is a clause of the form

$$l_0 \; or \ldots \; or l_k \leftarrow l_{k+1}, \ldots, l_m, \; not \; l_{m+1}, \; \ldots, not \; l_n, \tag{3.1}$$

where $l_i$'s are literals. A literal is an atom $p$ or its negation $\neg p$. An atom is an expression of the form $p(t_1, \ldots, t_h)$ and all $t_i$ are terms composed of function symbols and variables.

ASP formalism contains two negations that need to be distinguished: a classical negation noted $\neg$ and a negation by failure or default negation noted *not*. $\neg p$ means that it can be proven that $p$ is false and *not* $p$ means that $p$ cannot be proven to be true in the absence of sufficient information. non-monotonic properties are mainly due to this "negation as failure" connector. Literals possibly preceded by *not* are called extended literals. The literal $p$ extended counterpart $\neg p$ is called the contrary. *or* is the other connector in the formalism, which is called epistemic disjunction and is different from the classical disjunction $\vee$. A formula $A \vee B$ of classical logic says that $A$ is true or $B$ is true while a rule $A$ *or* $B$, can be interpreted epistemically, which means that every possible set of reasoners' beliefs must satisfy $A$ or satisfy $B$.

If $r$ is a rule of type 3.1 then

- $head(r) = l_0, \ldots, l_k$
- $pos(r) = l_{k+1}, \ldots, l_m$
- $neg(r) = l_{m+1}, \ldots, l_n$
- $body(r) = l_{k+1}, \ldots, l_m, notl_{m+1}, \ldots, notl_n$

If $head(r) = \emptyset$ rule $r$ is called a constraint that is used to eliminate undesirable solutions by specifying conditions that must not hold. a constraint is written as

$$\leftarrow l_{k+1}, \ldots, l_m, notl_{m+1}, \ldots, notl_n. \tag{3.2}$$

A rule r such that $body(r) = \emptyset$ is called a fact and is written as

$$l_0 \; or \ldots or \; l_k. \tag{3.3}$$

**Definition** An ASP program is a pair $\{\sigma, \Pi\}$ where $\sigma$ is a signature and $\Pi$ is a collection of logic programming rules over $\sigma$.

To introduce the semantics of the Answer Set Programs, we need the following terminology. The terms, literals, and rules of the program $\Pi$ with signature $\sigma$ are called ground if they contain no variables and no symbols for arithmetic functions. A program is called ground if all its rules are ground.

Consistent sets of ground literals on $\sigma$, containing all arithmetic literals that are true under the standard interpretation of their symbols, are called partial interpretations of $\sigma$. A literal $l$ is true in a partial interpretation $S$ if $l \in S$; $l$ is false in $S$ if $\neg\, l \in S$; otherwise $l$ is unknown in $S$. An extended literal *not* $l$ is true in $S$ if $l \in S$; otherwise, it is false in $S$.

The semantics of the answer sets of a logic program $\Pi$ assigns to $\Pi$ a collection of answer sets that are partial interpretations of $\sigma(\Pi)$ corresponding to possible sets of beliefs which can be built by a reasoner on the basis of rules of $\Pi$. The precise definition of answer sets will be first given for programs whose rules do not contain default negation, i.e. $\Pi^+ = \{r \in \Pi | neg(r) = \emptyset\}$.

**Definition (Answer set of positive programs)** A partial interpretation $S$ of $\sigma(\Pi^+)$ is an answer set for $\Pi^+$ if $S$ is minimal (in the sense of a set-theoretic inclusion) among the partial interpretations satisfying the rules of $\Pi^+$.

To extend the definition of answer sets to arbitrary programs, take any program $\Pi$ , and let $S$ be a partial interpretation of $\sigma(\Pi)$. The **reduct**, $\Pi^S$ , of $\Pi$ relative to $S$ is the set of rules $head(r) \leftarrow pos(r)$ or for all rules in $\Pi$ such that $neg(r) \cap S = \emptyset$. Thus, $\Pi^S$ is a program without default negation.

**Definition (Answer set)** A partial interpretation $S$ of $\sigma(\Pi)$ is an answer set for $\Pi$ if $S$ is an answer set for $\Pi^S$.

ASP formalism has clear semantics with well-defined mathematical meaning, and there exist solvers that automate the computation of Answer Sets. ASP solvers compute all stable models for a given program in two steps. In the first step, the program variables are grounded, that is, they are instantiated by the terms in the program. Through the second step, a "sat" solver generates all the interpretations that satisfy the instantiated rules. The "sat" problem, that is, the computation of all the interpretations that satisfy a given proposition formula, is known to be an NP-complete problem. Clingo [104, 73] is a well-known ASP solver that combines "Clasp" [72] (the solver) and "Gringo" (the grounder). Clasp is an answer set solver for (extended) normal and disjunctive logic programs. It combines the high-level modeling capacities of ASP with state-of-the-art techniques from the area of Boolean constraint solving. Gringo is a grounder that, given an input program with first-order variables, computes an equivalent ground (variable-free) program. Its output can be further processed with Clasp. ASP has a variety of applications that include decision support systems for the space shuttle [146], metabolic network completion [67], and train scheduling [1]. Moreover, it enables the formalization of ethical theories and the verification of the validity of various statements in different scenarios for each theory[69].

## 3.2 Planning

Planning in AI involves the process of creating a sequence of actions or steps to achieve a specific goal or a set of objectives. AI planning is crucial for developing intelligent systems that can operate automatically, adapt to changes, and complete complex objectives in a structured manner. Planning has a wide variety of applications that include: Managing resources in health care, self-driving cars, optimizing routes and resource allocations in logistics, Navigating environments in robotics.

There are different planning methods that are used to model states and actions. Each action has preconditions (conditions that must be true before the action can be executed) and effects (the outcome or result of the action). Actions transform the state of the world from one state to another. A plan is a sequence of actions that leads from the initial state to the goal state. There are various types of planning, including classical, probabilistic, etc. The classical planning which we adopted in this thesis assumes a deterministic and fully observable environment, using algorithms such as STRIPS (Stanford Research Institute Problem Solver)[64] and GraphPlan[24]. Probabilistic planning deals with uncertainty in actions and outcomes, often using Markov Decision Processes (MDPs).

Planning, particularly used in computational ethics to model consequentialist and deontologist theories. In this section, we will present two planning formalisms, namely Event Calculus (EC) and Hierarchical Task Network (HTN). The former is a well-known classical formalism, and the latter is a rather different planning paradigm based on decomposing complex tasks into simpler subtasks.

### 3.2.1 Event Calculus

Event calculus is a formal language to represent and reason about events and their effects over time. The original version of the event calculus, introduced by Robert Kowalski and Marek Sergot in 1986,[113] was formulated as a logic program and designed to represent narratives and database updates [111]. The Event Calculus has been reformulated in various logic programming forms [135] [54, 103, 163] in classical logic [166, 136], in modal logic [41, 46] and as an "action description language" [103]. It has been extended and applied in many contexts, including planning, cognitive robotics, abductive reasoning, and legal reasoning [135].

The logical mechanism of the event calculus is summarized in simple terms by Shanahan [167] in Figure 3.1. This mechanism can infer what is true when given what happens when and what actions do. The part 'what happens when' is a narrative of events, and the part 'what actions do' describes the effects of actions.

Figure 3.1 illustrates how event calculus can provide a logical foundation for a number of reasoning tasks. These can generally be categorized into deductive

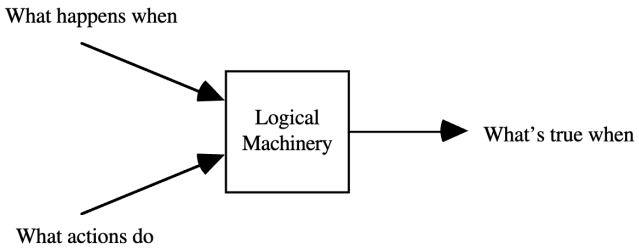



Figure 3.1: How the event calculus works [167]

tasks, abductive tasks, and inductive tasks.

- In a deductive task, the narrative of events and the effects of actions are given, and we seek to find out what is true at a particular point in time. In other words, "what happens when" and "what actions do" are given and "what's true when" is required. Deductive tasks include temporal projection or prediction, where we seek to find out the outcome of a known sequence of actions.

- Abduction is used to determine a sequence or narrative of events that need to occur, given the effects of actions and a set of fluents that hold at a specified point in time. This implies that "what actions do" and "what's true when" are supplied, and "what happens when" is required, i.e. a sequence of actions is sought that leads to a given outcome. Examples of such tasks include temporal explanation, certain types of diagnosis, and planning.

- Induction aims to derive the effects of actions from a given narrative of events and information about the fluents that hold at different points of time. In other words, in an inductive task, "what's true when" and "what happens when" are supplied, but "what actions do" is required. In this case, we seek a set of general rules or a theory of the effects of actions that explains observed data. Inductive tasks include scientific discovery, learning, and theory formation.

**The Frame Problem** In Event Calculus terms, the frame problem [34, 129] is the problem of expressing that in most cases a given action will not initiate or terminate a given fluent. The event calculus solves the frame problem by using a non-monotonic logic, such as first-order logic with circumscription [166] or, as a logic program, in Horn clause logic with negation as failure [113]. Circumscription allows us to assume by default that the events known to occur are the only events that occur. That is, there are no unexpected events. Circumscription is one of the several semantics that can be given to negation as failure [77] and coincides closely with the stable model semantics [116]. Thus, event calculus can be reformulated in the first-order stable model semantics that is

the mathematical basis of Answer Set Programming (ASP) [109, 116]. The ASP formalization allows efficient answer set solvers to be applied to event calculus reasoning.

Several versions of event calculus have been developed so far to deal with specific problems. These extensions have been used to formalize the conditional effects of events, nondeterministic events, concurrent events, events with delayed effects, gradual changes, events with duration, triggered events, events with indirect effects, continuous change, and the common sense law of inertia, [144, 135, 143]. In this work, we use a specific variant of the event calculus formulated in ASP and adapted for planning tasks.

**Axioms** We use a simplified version of the event calculus, introduced in [23]. This version of Event Calculus relies on a formal representation of events and states on a discrete set of time points. The time is represented in an explicit linear form that associates states and events when a change happens in the world. The axioms of the event calculus are presented below.

The transition between states occurs through events that take place at time $T$. A state is characterized by fluents that hold or not, depending on the occurrences of events. An event may initiate a certain fluent or terminate it. In the latter case, the fluent is marked as a negative effect. In addition, these events depend on preconditions that must be met for the event to occur.

```
negative(neg(F)) :- effect(E,neg(F)).
initiates(E,F,T) :- effect(E,F), occurs(E,T), not negative(F).
terminates(E,F,T) :- effect(E,neg(F)), occurs(E,T),time(T).
clipped(F,T) :- terminates(E,F,T).

:- occurs(E,T), prec(F,E), not holds(F,T), act(E), time(T).
```

A fluent holds at $T$ if it was initiated by an event occurrence at $T-1$. This is indicated by the predicate `hold/2`.. A fluent that is true at $T$ continues to hold until the occurrence of an event terminates it. These rules implicitly imply the common sense law of inertia which states that a fluent holds at a time $T$, if an event occurs and initiates it at an earlier time $T-1$ and there is no other event that occurs and terminates it at the time $T-1$.

```
holds(F,0) :- initially(F).
holds(F,T) :- initiates(E,F,T-1), time(T).
holds(F,T) :- holds(F,T-1), not clipped(F,T-1), time(T).
```

In order to use the ASP formulation of event calculus for abductive reasoning i.e. planning, we use the following rules.

```
0 {occurs(E, T):act(E)} 1 :- act(E), time(T),T<maxtime.
```

The choice rule `0 {occurs(E, T)} 1 :- act(E), time(T),T<maxtime.` is used to exempt the `occurs/2` predicate from minimization in ASP, this is the generator part that we use to solve a planning problem. Choice rules describe a set of "potential solutions," i.e. a simple superset of the set of solutions to the given search problem.

### 3.2.2 Hierarchical Task Network

Hierarchical Task Network (HTN) planning is an approach to automated planning that decomposes complex tasks into simpler and more manageable subtasks [145]. The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain in HTN usually includes a set of tasks and a set of decomposition rules. The decomposition rules specify how a task can be broken down into a set of subtasks. An example of HTN planning is the travel problem [17]. Consider the planning problem of arranging travel, in which one has to arrange accommodation and various forms of transportation. This problem can be viewed as a simple HTN planning problem, in which there is a single task, "arrange travel", which can be decomposed into arranging transportation, accommodations, and local transportation. Each of these more specific tasks can successively be decomposed on the basis of alternative modes of transportation and accommodations, eventually reducing to actions that can be really executed.

In this thesis, we use a special version of HTN called *Dynagent* [92], an online forward-chaining total-order HTN planning algorithm. Similarly to other planners, on task decomposition; however, Dynagent is an online planning algorithm, which means it supports planning, execution, and replanning. The online total-order forward chaining HTN planning Dynagent is formulated in Prolog. Here, we briefly describe the components of this planner.

**Definition.** *Fluents* are used to represent the states. A fluent is an atom of the form: $P(T_1, \ldots, T_n)$ where $n \geq 0$, $P$ is a $n - ary$ predicate, and each $T_i(1 \leq i \leq n)$ is a term.

**Definition.** A *belief rule* is of the following form: $belief(F, [F_1, \ldots, F_n])$ where $n \geq 0$, $F$ is a derived fluent called the head, each $F_i(1 \leq i \leq n)$ is a fluent, and the set of fluents $F_1, \ldots, F_n$ is called the body. When $n > 0$, $F$ is a derived fluent. When $n = 0$, the belief rule $belief(F, [\,])$ can be expressed as $belief(F)$ and $F$ is called a fact. A dynamic fluent $F$ is denoted by $dy(F)$. A dynamic fluent can be asserted or retracted from the belief set while performing an action or after an observation.

**Definition.** A *task* is a predicate of the form: $T(X_1, \ldots, X_n)$ where $n \geq 0$, $T$ is an $n-ary$ task symbol, and each $X_i(1 \leq i \leq n)$ is a term. When $T$ is a $0-ary$ task symbol, the task $T()$ can be abbreviated to $T$. A task is either *abstract* or *primitive*. The cost $C$ of the task $T$, where $C$ is a number (real number or integer), is represented as $cost(T, C)$. A *plan* is a list of tasks: $[T_1, \ldots, T_n]$ where $n \geq 0$ and each $T_i(1 \leq i \leq n)$ is a task, which is called the $i - th$ action of the plan. The cost of the plan $[T_1, \ldots, Tn]$ is the sum of each cost of $T_i(1 \leq i \leq n)$.

**Definition.** *Action rules* are used to represent the effect of an action. An action rule is of the following form: $action(A, C, E)$, where $A$ is an action, $C$ is

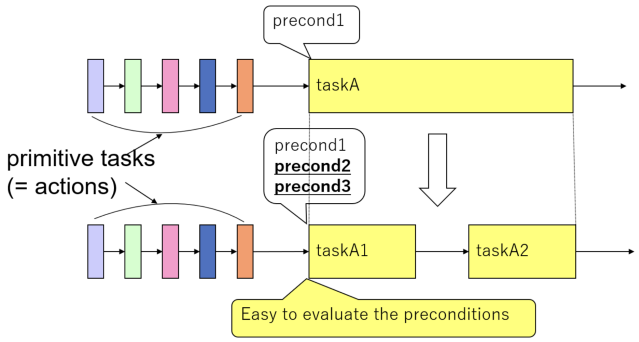



Figure 3.2: Task decomposition

a list of fluents called preconditions, $E$ is a list of effects, an effect is of the form $initiates(F)$ or $terminates(F)$, where $F$ is a fluent.

**Definition.** *Task-decomposition Rule* represents the method to decompose a task into subtasks. A task-decomposition rule is of the following form: $htn(H, C, B)$ where $H$ is an abstract task called the head, $C$ is a list of fluents called preconditions, and $B$ is a plan called the body.

**Definition.** A *planning domain* is composed of a set of beliefs and planning knowledge. The belief set represents the current state of the world and the knowledge of planning represents the effects of actions and the methods to decompose tasks into subtasks. A belief set is the pair $\langle D, S \rangle$ where $D$ is a set of dynamic fluents, and $S$ is a set of belief rules. Planning knowledge is represented by the tuple $\langle AR, TDR, COST \rangle$ where $AR$ is a set of action rules, TDR is a set of task-decomposition rules and $COST$ is a set of the costs of each task.

Here, we briefly describe the process used for planning in the online HTN planner. More detailed explanations can be found in [91, 92].

### Planning Process

Given a specification of a planning domain, the planner recursively decomposes the abstract tasks into primitive subtasks before execution. The HTN planning algorithm is forward-chaining and the task decomposition is conducted in the same order as task execution. As shown in Figure 3.2, when $taskA$ is decomposed into a plan, all previous tasks before $taskA$ are already decomposed into primitive subtasks. The preconditions (*precond2* and *precond3*) of the task decomposition, which are added to the preconditions of the first subtask ($taskA1$), must be satisfied before task execution.

The HTN planning search space is an or-search tree of plans; it means that a task can be decomposed to multiple plans. For example, in Figure 3.3, the

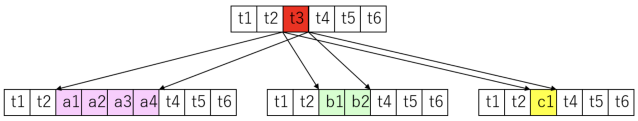



Figure 3.3: Multiple subplans

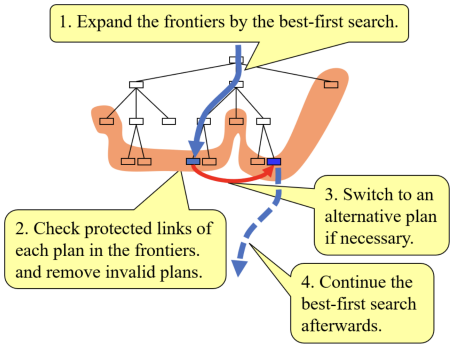



Figure 3.4: Switching to an alternative plan

task $t3$ in a plan is decomposed into three subplans $[a1, a2, a3, a4]$, $[b1, b2]$, and $[c1]$.

### Replanning Process

In the Dynagent planning algorithm, each dynamic precondition (a dynamic fluent) of a task in a plan is recorded. These preconditions serve as a protected link that must be true before the execution of the task. When a dynamic belief is updated and the protected link no longer holds, the corresponding plan is removed from the frontiers of the or-search tree. The planner then switches the current plan to the plan with the next-lowest cost and continues the best-first search using the frontiers of valid plans. This is illustrated in Figure 3.4.

After a dynamic belief update, if a protected link, which was previously false, becomes true, the corresponding plan becomes valid and is asserted to the frontiers of the search tree. Since frontier plans are always ordered, if the new plan has the lowest cost, the planner switches to the new plan and continues the best-first search. A similar switching process occurs when an action cost is updated. In this case, the plan's overall costs are evaluated in the frontiers and reordered in ascending order.

During execution of an action, the planner maintains all alternative plans and updates them in case they become more optimal. As shown in Figure 3.5, if an action is executed successfully, it will be removed from all the plans with that action in the beginning. If the execution fails, the plans that begin with that action are discarded. This is shown in Figure 3.6. In this case, the planning agent stops the execution of the plan and restarts the best-first search using the

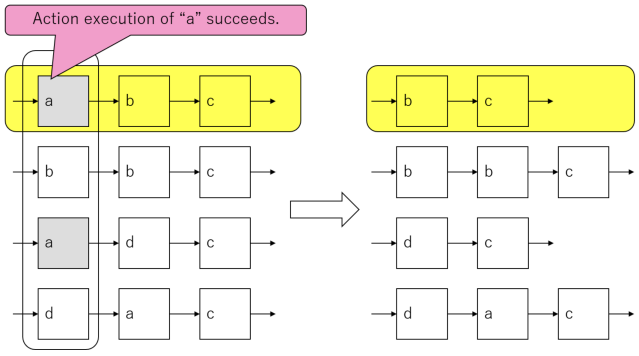



Figure 3.5: Plan update after action execution

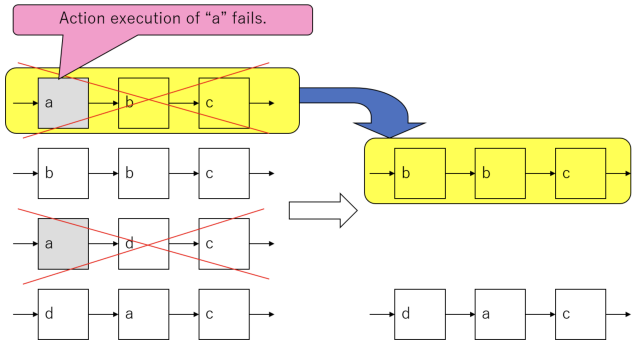



Figure 3.6: Replan after action failure

valid plans on the frontiers until it finds an optimal plan.

## 3.3 Voting Systems

Aggregation of several measurements or orders is the subject of research in many fields of AI, such as Multi-Criteria Decision Making(MCDM). These approaches are used to analyze and solve decision problems that involve multiple criteria [83, 12]. MCDM methods propose a way to combine the evaluation of criteria over alternatives. Alternatives can be evaluated in several ways, for example, pairwise comparison of alternatives [38], approval voting [28], ordinal ranking of alternatives [107], and classification of alternatives [153, 195].

Voting systems are ways of aggregating several preference orders into a single collective preference. They are used in conflict resolution, policy making, and reaching a collective decision in elections or contests. Alternative methodological techniques exist to address these issues, such as conflict analysis methods [108], group decision support systems [2], and multi-criteria decision analysis [164]. The procedures used to aggregate the preference orders are called voting rules.

The preference orders may represent different things depending on the domain in which they are used. In social choice theory, they represent the votes or individual preferences of the voters. In MCDM, they represent the preference order based on the criteria.

Voting rules are justified by an intuitive rationale and their properties are well studied in the social choice theory. Due to well-defined and expressive aggregators, voting rules are also applied in computational ethics problems, for example, in conflict resolution and judgment aggregation [57], moral uncertainty[127, 147], and solving dilemmas based on individual votes [36]. Here, we present a brief review of voting systems and their properties. More details can be found in [123, 30, 29].

A common voting rule is the *plurality* system, in which the winner is the candidate with the highest number of votes. The *plurality runoff* is similar, with the additional condition that the winner must be supported by more than half of the voters. If no candidate satisfies this condition, the two candidates with the highest number of votes proceed to the next round, and the plurality rule is applied again. *Borda's rule* is based on the points assigned to the alternatives according to the rank they obtain in individual preference orderings. Given $k$ alternatives; the lowest rank gets 0 points, next to the lowest 1 point, the next 2 points, and the highest rank $k-1$ points. The motivation behind the Borda's rule is to elect the alternative which on the average is positioned higher in the individual rankings than any other alternative.

The mentioned methods are positional rules, i.e. the candidates get some points based on the position in individual ranking. The aim is to choose a candidate that is better positioned in the voters' preferences, in some specific sense, than other candidates. Another intuitive way to determine the winner is by pairwise comparison, meaning that voters vote for their candidate in every pair that can be formed.There are several voting methods that are based on such pairwise comparisons of decision alternatives. They differ in how the winner is determined once the pairwise votes have been taken. A common property of most pairwise systems is the *Condorcet winner*, that is, they elect the candidate that beats every other candidate in pairwise voting, when such a candidate exists.

Examples of pairwise systems are Copeland's rule, Dodgson's method, and the max-min method. *Copeland's rule* is based on all $(k-1)/2$ majority comparisons of alternatives. For each comparison, the winning candidate receives 1 point and the non-winning one 0 points. The Copeland score of a candidate is the sum of his points in all pairwise comparisons. The winner is the candidate with the highest Copeland score. *Dodgson's method* aims to elect a Condorcet winner when one exists. Since this is not always the case, the method looks for the candidate that is closest to a Condorcet winner, in the sense that the number of pairwise preference changes needed for the candidate to become a Condorcet winner is smaller than the changes needed to make any other candidate a winner. The *Max-Min* method determines the minimum support of a candidate in all pairwise comparisons, i.e. the number of votes he receives when confronted with his toughest competitor. The candidate with the largest

| Voting Rules | Condocet winner | Condocet loser | Monotonicity | Pareto | Consistency |
|---|---|---|---|---|---|
| Plurality | No | No | Yes | Yes | Yes |
| Plurality run-off | No | Yes | No | Yes | No |
| Borda | No | Yes | No | No | No |
| Copeland's rule | Yes | Yes | Yes | Yes | No |
| Dodgson's rule | Yes | Yes | Yes | Yes | No |
| Max-min rule | Yes | Yes | Yes | Yes | No |

Table 3.1: Properties of voting systems [148]

minimum support is the max-min winner.

### 3.3.1 Properties

As mentioned previously, an important property in voting systems is the Condorcet winner. Lack of this property may cause non-transitive or cyclic preference orders, which is the case in the plurality rule. Another property is the *Condorcet loser*. It requires that an eventual Condorcet loser be excluded from the choice set. This property is generally accepted as a plausible constraint on social choices. A compelling property that can be found in the literature is *monotonicity*. This property implies that additional support should never hurt the chances of a candidate getting elected. *Pareto efficiency* is another important property in voting systems. It states that if every voter strictly prefers the alternative $A$ to the alternative $B$, then $B$ is not the collective choice. Another desirable property is *consistency*. It concerns choices made by subsets of voters. This means that if the subgroups elect the same alternatives, then these alternatives should also be chosen by the group at large. Despite its intuitive plausibility, consistency is not common among voting systems. The properties of each voting rule are summarized in Table 3.1.

## 3.4 Conclusions

The objective of this chapter was to discuss the technical methods used in research. For this purpose, we introduce Prolog and ASP as logic programming tools. Then the planning formalism used for modeling the agents' plans is explained. Finally, voting systems as an aggregation approach. Voting systems were only briefly introduced. In this thesis, we only made use of Copeland's rule as an aggregator in Chapter 5. Our contribution showed that voting systems the underlying expressivity and intuition of voting rules make them a suitable

candidate for aggregation problems in ethical evaluations. Further investigation of voting rules remains a future work.

# Part II

# Contributions

In this part, we present the contributions in this thesis. Our research problem concerns the legal and ethical compliance of AI agents' actions. We first investigated this problem from the point of view of legal compliance. We proposed a framework for planning based on event calculus and GDPR compliance check in Chapter 4. The ethical aspect of our research problem was then explored. In this part, our initial assumption was that legal constraint act as permissibility criteria and ethical constraints as optimization criteria. This means that illegal options are removed and that the best compliant option is chosen according to ethical constraints. This is the other interpretation for the terms hard and soft norms. This assumption implies that hard ethical norms are already reflected in the law; however, that often holds, it is not always the case. In addition, considering ethical norms only as soft norms neglects the cases where some ethical norms are in conflict with the legal norms. We adopted this view as a starting step in the development of an ethical evaluation model. Therefore, in the development of an ethical evaluation model, we made the assumption that all options are permissible and the question was which is ethically the best? We proposed a framework for ethical evaluation based on multiple values to address this issue. The evaluations in this model are on an ordinal scale and aggregated using voting systems. In the next contribution in Chapter 6 we integrate this model with compliance framework (cf. Chapter 4) with HTN planner. We then further investigated the issue of considering legal and ethical norms as hard and soft norms, respectively. In Chapter 7 we propose an approach in which both legal and ethical hard norms are integrated into a unified compliance model. contrary to the previous contributions where we applied our model on data processing scenarios; in our unified model is adopted and tested on a health care delivery system.

# Chapter 4

# Legal Compliance *Automated Data Processing with GDPR Compliance*

Automatic compliance checking is a key topic in the field of computational law. AI technologies pose many risks that have been subject to regularization in recent years. One of the areas where most of the AI applications are concerned is the processing of personal data. With the enforcement of the European General Data Protection Regulation (GDPR), such applications must guarantee compliance with the obligations set forth. The GDPR provides legal requirements for the processing of personal data. Organizations, corporations, and developers must take technical measures to assess the compliance of personal data processing with GDPR. GDPR applies to applications where personal data is used or processed. In this work, we are particularly interested in GDPR compliance checking and automated data processing applications. Designing such a system requires on the one hand a formalism to represent operations on personal data and automatically derive a sequence of operations to perform certain processing. On the other hand, we need a formalization of GDPR regulations that can be integrated as constraints with this formalism.

Since the adoption of the GDPR, several tools have been introduced to facilitate compliance assessment for data controllers and processors. Some of these methods are in the form of questionnaires or checklists that assess the compliance of companies or organizations, for example, Microsoft Trust Center [3]. However, these methods are not suitable for automated compliance checking. Others focus on building an ontological concept of GDPR, for example PrOnto [149], a privacy ontology, based on LegalRuleML (cf. Section 1.3.1) for legal reasoning and knowledge representation. These methods can be used to create a repository of rules based on regulative and constitutive norms (cf. [159]) or legal analysis. These methods focus on knowledge representation requirements and hardly adapt to an automated data processing environment. Another body of work in the legal domain is developing policy languages to represent regulatory norms that can be used to verify compliance of business processes [52, 27].

In this contribution, we propose a framework for automatic data processing and compliance checking. We model data operations as agents' actions. Data processing is viewed as a planning problem. The agent can deduct a series of operations to process personal data for a specific purpose. GDPR regulations are integrated as constraints to check for compliance. We use ASP (cf. Section 3.1.2) and Event Calculus (cf. Section 3.2.1) formalism to model the planning problem and adopt the SPECIAL policy language(cf. Section 1.3.2) to represent GDPR regulations. SPECIAL policy language is reformulated in ASP and integrated in the agent's, compliance checking process.

In other words, our framework has composed of two main components.

- **Planning**, Given a goal state, an initial state and a description of the domain, this component generates all possible plans regardless of their compliance.
- **Compliance Engine**, Given a plan, this component checks for its compliance against GDPR regulatory norms. In the case of a non-compliant plan, it explains the missing obligations.

This chapter is organized as follows. In Section 4.1 we explain our modular framework and describe how each component is constructed. In Section 4.2, we evaluate our framework in two simple scenarios and discuss the results.

## 4.1 Overall Model Architecture

We consider an agent who handles personal data processing. We are concerned about the compliance of the agents' actions with GDPR. In order to model both requirements of the planning domain and compliance checking, we adopt a modular framework with 2 components. The first is the planning module that contains the specification of storage, personal data in the system, and the agents' actions. Each action describes a transformation of the data or move it to another storage. Given an initial state and a goal state, this module generates all the possible plans which satisfy the goal. By plan, we refer to a sequence of processing on personal data. The second module (Compliance engine) checks if each plan is compliant with both regulatory norms and data subject's given consent, and can provide explanation of missing obligations in the case of non-compliance. Figure 4.1 illustrates the structure of the framework.

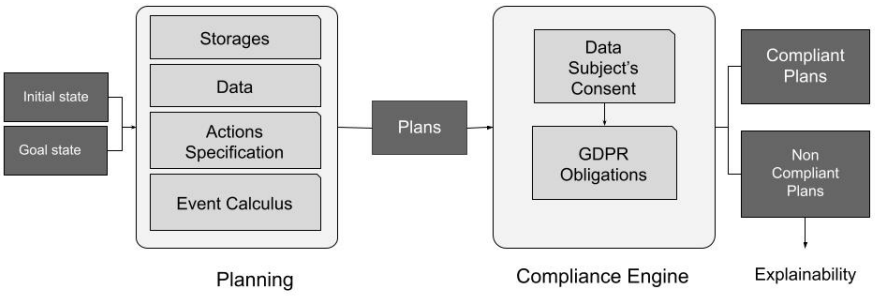



Figure 4.1: Modular structure

### 4.1.1 A Personal Data Handling Use Case

We describe our framework by implementing it in a use case model. An international company also operates in multiple European countries and the United States. The company has several sectors to provide services to customers. Each sector owns a server to store personal data. The servers are connected through an internal network and can transmit data between each other. One of the servers is a computing server in which the company analyzes customer data for various purposes. The company also has a partner as a data processor, which provides analytic services to the company. Figure 4.2 illustrates the connection between the company's servers and its partner processor, as well as their location.

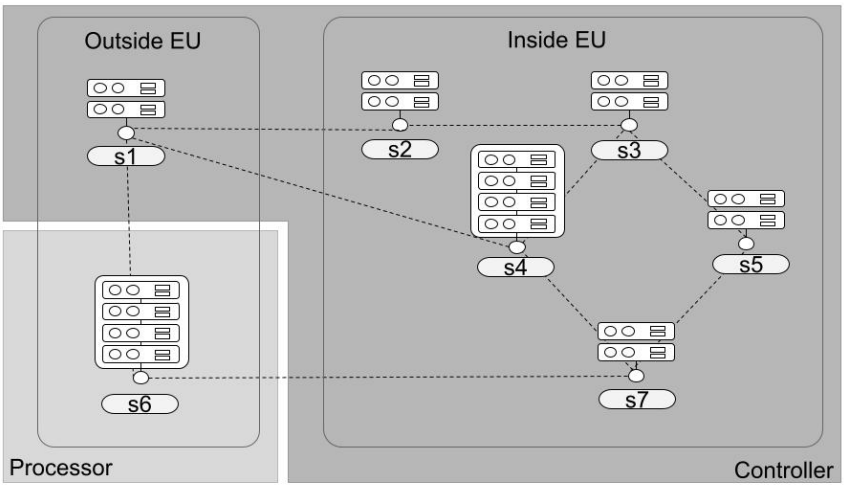



Figure 4.2: Connection among servers

In order to provide services, the company needs to analyze the customer data and use its result. When a sector requests the outcome of a particular analysis, a sequence of processing should be performed to provide the result to its corresponding server. We implement our framework in this scenario to design an agent to automatically generate a sequence of data processing to provide the requested output on a data and check for the compliance of generated sequences.

### 4.1.2 Planning Component

The key challenge in the design of the planning domain is formalizing actions, states, and domain knowledge in a way that allows for both planning and compliance checking. We assign GDPR-related attributes to domain objects to use them for compliance checking. We describe the main parts of the planning domain.

#### Storage

Storage in our framework basically represents anything capable of storing and processing data. *e.g.* servers, cloud space, etc. We use the `storage` to represent a server in our planning domain. We also use the `connected/2` predicate to represent that there is a connection between two storages. For example, below the formalization of ASP means that `s1`, `s2`, and `s3` are storage and there is a connection between `s1` and `s2`, and `s1` and `s3`, which means that they can transmit data.

```
1 storage(s1).
2 storage(s2).
```

```
storage(s3).
connected(s1,s2).
connected(s1,s3).
```

Knowledge about other servers and the connection between them has been formalized in the same way. In order to assess the compliance of these actions, we need information about the action itself, as well as other complementary information that is concerned with the GDPR. This information is represented by the predicate `has/3`. For example, the code below shows the controller of the storage `s6` and `s7` and the location of `s1` and `s2`.

```
has(s6, controller, aProcessor).
has(s7, controller, aCompany).
has(s1, location, us).
has(s2, location, eu).
```

`s4` and `s6` are computing servers capable of analyzing data for various purposes. Knowledge about the supported purposes of each one can be represented in the following way.

```
has(s4, analysisPurpose ,marketing).
has(s4, analysisPurpose ,personalisedAdvertising).
has(s6, analysisPurpose ,optimisationForController).
```

### Data

Personal data is represented by a `resource`. Each resource belongs to a data subject and has a category. Consider two resources `d1` and `d2` where they have categories *Purchases and Spending Habit* and *Service Consumption Behavior* and this is represented in the following way in ASP.

```
resource(d1).
resource(d2).
has(d1, dataCategory, purchasesAndSpendingHabit).
has(d2, dataCategory, serviceConsumptionBehavior).
```

When performing actions on resources, they either move to another storage or transform into a new data. We need to represent these data manipulations in our domain. We define the predicate *data* that represents any personal data, either a resource or the output of an analysis process on this resource with a certain purpose.

```
data(D):-
    resource(D).

data( analysisOutput(D,P) ):-
    resource(D),
    purpose(P).
```

If the attributes are static, we represent them as facts, and if they are dynamic, we represent them by fluents. Attributes like the storage of the data or the content of a storage are impacted by actions.

### Actions Specification

The agent action in our domain represents a data processing. It supports the transfer and analysis processing of personal data for several purposes. A transfer action in our domain is characterized by the data, its current location, the destination, and the purpose of the transfer. To perform an analysis action, we require the data and storage where the processing is taking place and the purpose of the analysis as well. We formalize the knowledge about agents actions as follows:

```
act(transfer(D,A,B,P)):-
    data(D),
    storage(A),
    storage(B),
    purpose(P),
    connected(A,B),
    A!=B.

act(analyse(D,A,P)):-
    data(D),
    storage(A),
    purpose(P) ,
    has(A, analysisPurpose ,P).
```

Each action should be specified by its preconditions and effects. A transfer action changes the storage of the data, or equivalently, it modifies the content of the origin and destination storage. This is captured by the fluent predicate `hasData(A,D)`, which represents that storage $A$ has data $D$.The analysis action transforms the personal data into new data, which is the result of this analysis. In the following, we represent the effects and preconditions of these actions.

```
prec( hasData(A,D), transfer(D,A,B,P)):-
    act(transfer(D,A,B,P)).
effect(transfer(D,A,B,P), hasData(B,D)):-
    act(transfer(D,A,B,P)).
effect(transfer(D,A,B,P), neg(hasData(A,D))):-
    act(transfer(D,A,B,P)).

prec( hasData(A,D), analyze(D,A,P)):-
    act(analyze(D,A,P)).
effect(analyze(D,A,P), hasData( A, analysisOutput(D,P)  )):-
    act(analyze(D,A,P)).
effect(analyze(D,A,P), neg( hasData(A,D) )):-
    act(analyse(D,A,P)).
```

As an example, we describe the effects and preconditions of an action analysis. `prec( hasData(A,D), analyze(D,A,P))` means that in order to perform the action, `analyze(D,A,P)` the fluent `hasData(S,D)` should hold, which means that the corresponding data should be present in the corresponding storage. When the action is performed, the data transform into the output and are represented

by `analysisOutput(D,P)` . The last line `effect(analyze(D,A,P), neg( hasData(A,D) ))` means that after the action is performed, the output data would be replaced by the input data. The predicate `neg( hasData(A,D) ))` indicates the negative effect of the action, which is the input data that is no longer stored in the corresponding storage.

### 4.1.3 Compliance Engine

This module contains the required elements for compliance checking against regulative norms and data subject's consent. For this purpose, it should be fed with legal specifications and formalization of the given consent. The legal specification contains organizational measures, the legal basis for the processing, and the duties defined for processing. The compliance engine has 3 main parts; the first part assigns legal information to actions based on the legal specifications. Then it checks compliance with the regulatory obligations in the next one. In the last part, we can check the compliance of these actions with the data subject's given consent. Each part is described in the following.

#### Actions as Business Policy

An action should be associated with legal information similar to the attributes of a business policy in SPECIAL (see the example of a business policy in the Listing 1.1). The legal information associated with an action must match the description of a business policy to be compatible with the underlying policy language. The following example shows how a transfer action is associated with legal information using the format `has(Action, GDPR_attribute, Value)`. For example, knowledge about the legal basis at line 6 or appropriate safeguards for a personal data transfer at lines 7 and 8. Notice that only a fragment of the associated attributes are shown below, you can find the complete list in the implementation codes in Appendix A, or in the code repository[1].

```
1  has(transfer(D,A,B,P), dataCategory, X) :-
2      act(transfer(D,A,B,P)),
3      has(D, dataCategory, X).
4
5  has(transfer(D,A,B,P), storage, B):-
6      act(transfer(D,A,B,P)).
7
8  has(transfer(D,A,B,P), purpose, P):-
9      act(transfer(D,A,B,P)).
10
11 has(transfer(D,A,B,P), recipient, X):-
12     act(transfer(D,A,B,P)),
13     has(B, controller, X).
14
15 system_legal_basis(art6_1_a_Consent).
```

[1] https://gitlab.lip6.fr/taheri/planning-compliance-mechanism-policies.git.

```
16
17 has(transfer(D,A,B,P), legalBasis, X):-
18     act(transfer(D,A,B,P)),
19     system_legal_basis(X).
20
21 transfer_safeguard(s4, s1, art46_2_e_ApprovedCodeOfConduct).
22
23 has(transfer(D,A,B,P), measures, X):-
24     act(transfer(D,A,B,P)),
25     transfer_safeguard(A,B, X).
```

### GDPR Regulatory Obligations

We define necessary predicates and axioms to produce a straightforward translation of the GDPR regulatory obligations encoded in SPECIAL policy language and support for explainability in the case of non-compliance. As an example, the obligation at the bottom level, Article 6-1 (legal processing) presented in Section 1.3.2, has been translated into ASP using the predicate `fulfills/2`. This rule means that an action $O$ fulfills the obligations of Article 6-1, if it has a legal basis as defined in the list. Note that in ASP `pred(a;b)` is equivalent to `pred(a)` and `pred(b)`.

```
1  art6_1_LegalBasis( art6_1_a_Consent;
2                     art6_1_b_Contract;
3                     art6_1_c_LegalObligation;
4                     art6_1_d_VitalInterest;
5                     art6_1_e_PublicInterest;
6                     art6_1_f_LegitimateInterest).
7
8  fulfills(O, art6_1_LegalBasis):-
9      has(O, legalBasis, X),
10     art6_1_LegalBasis(X),
11     act(O).
```

Each regulation is named after its reference in the GDPR text, part of the current supported obligations in this module are presented as a predicate `regulation`. Here `gdpr_Requirements` represents the obligations at the highest level.

```
1  regulation(art6_1_LegalBasis;
2      art6_lawfulProcessing;
3      art12_22_SubjectRights;
4      chap3_RightsOfDataSubjects;
5      chap2_LawfulProcessing;
6      art9_sensitiveData;
7      gdpr_Requirements).
```

In the SPECIAL policy language, the obligations are represented as equivalency between classes. The classes are combined using the operators `ObjectCompl` ↪ `ementOf`, `ObjectUnionOf` and `ObjectIntersectOf`. We translate these oper-

ations using the predicates `comp/1`, `inUniounOf/2`, and `inIntersectOf/2`. For example, the Listing 1.3.2 is translated as follows.

```
inUnionOf(art6_LawfulProcessing ,chap2_LawfulProcessing ).
inUnionOf(art9_SensitiveData, chap2_LawfulProcessing ).
inUnionOf(art10_CriminalData, chap2_LawfulProcessing ).
```

The predicate `comp/1` is used as follows.

```
fulfills(P , comp(R1)):-
    not fulfills(P , R1 ),
    regulation(R1),
    act(P).
```

The predicate `inUnionOf/2` is defined in the following way. The first rule means that if an action $P$ fulfills the set of obligations of $R2$, and $R2$ is in the set of unions of $R1$, then it also fulfills the set of obligations of $R1$.

```
fulfills(P,R1):-
    fulfills(P,R2),
    inUnionOf(R2,R1),
    act(P).

fulfills(P,R1):-
    not fulfills(P,R2),
    inUnionOf(comp(R2),R1),
    act(P).
```

The operations, are defined `inIntersectOf/2`, are defined in the following way.

```
incompleteRequirment(P,R1):-
    inIntersectOf(F2,R1) ,
    not fulfills(P,R2),
    act(P).

fulfills(P,R1):-
    not incompleteRequirment(P, R1),
    act(P),
    inIntersectOf(_,R1).
```

An action is compliant if it fulfills all obligations of the fraction of the GDPR at the top level. A plan contains several actions, and it is possible that only a certain operation violates the plan compliance. In this case we are interested to know which missing obligation caused the non-compliance, in order to do so, we use the predicate `missing/2` in the following ASP rule, it indicates that the obligations of a certain article are missed.

```
missing1(P,R,R):-
    not fulfills(P,R),
    regulation(R),
    act(P),
    occurs(P,_).
```

```
6
7  missing1(P,R1,R2):-
8      not fulfills(P,R2),
9      upperClass(R3,R2),
10     missing1(P,R1,R3),
11     regulation(R1),
12     regulation(R2).
13
14 missing(P,R):-
15     missing1(P,R,gdpr_Requirements),
16     not auxiliaryRegulation(R).
```

### Data Subject's Consent

Suppose that when collecting personal data, the user has given explicit consent to transfer his *Purchases and spending habits* data for the purpose of *marketing*. Based on this consent, data can only be disclosed to *aCompnay*, and it should be stored only in Europe. We translate this consent using the same format as Listing 1.2. Note that SPECIAL also supports time intervals for the validity of consent, but we do not support it here. In our scenario, the data subject has also given his consent to analyze processing with the same attributes.

```
1 has(c2, dataCategory, serviceConsumptionBehavior).
2 has(c2, processing, transfer).
3 has(c2, purpose, marketing).
4 has(c2, recipient, aCompany).
5 has(c2, storageLocation, eu).
```

In our modeling, a processing is compliant with the given consent if it has the same attributes as the action. We check the compliance of an action with the given consent using the following set of rules. It basically states that valid consent of an operation is satisfied if there is a coherent consent for it; and an action is coherent with a consent if there is no difference between the attributes of the consent and the operation. We capture it by the predicate `validConsentSatisfied` that is true when there is coherent consent for it.

```
1 non_coherent(P,C):- has(P,A, Z1) , has(C, A, Z2) ,Z1!=Z2, act(P), consent
      ↪ (C).
2 validConsentSatisfied(P):- not non_coherent(P,C), act(P), consent(C).
```

## 4.2 Evaluations

Once we have modeled our domain knowledge, the planning module can be used to generate plans by providing an initial state and a goal state. A plan is generated to deliver the result of the processing of personal data to the server asking for it. Consider that data $d1$ are initially stored on the server $s1$. We represent this initial state by `initially( hasData(s1,d1))`.

There is a request from the server $s4$ for the results of the analysis on the data $d1$ with the purpose *marketing*. We represent this request by `requestAnalysis (s4, d7, marketing)`. This request is then translated into the goal of the system that the output of this analysis should be stored on the storage that requests it.

```
holds(goal,T):- holds( hasData(A, analysisOutput(D,P)), T ),
    requestAnalysis(A,D,P).
:- not holds(goal, maxtime).
```

After providing the initial state and a goal state, all possible plans are generated to satisfy the given request. The plans are included in Table 4.1. Note that all these sequences of actions are generated regardless of their compliance. Each plan is a set of actions presented by the predicate `perform/2` that indicates the action and the time step in which it can be performed.

Having the plans generated by the previous module, the compliance engine can distinguish the compliant plans with the noncompliant ones and also provide a simple explanation for non-compliance by referring to the missing obligations. A plan is compliant if all the actions in that plan are compliant. Below, we show the compliance check result in two scenarios; Compliance checking with (i) data subject's consent and (ii) GDPR regulatory norms. In both cases, the initial state and the goal state are the same and the same plans are generated by the planning module (shown in Table 4.1), therefore the compliance engine assesses the compliance of the identical plans but with different legal restrictions.

### 4.2.1 Compliance Check for Consent

In this scenario, the customer has given a customized set of consent for various data processing. In particular, we assume that the data subject has given her consent only for internal transfers in EU, so the compliance engine distinguishes noncompliant plans if they are compatible with the data subject's consent. Table 4.2 presents the compliance of each plan and the explanation of the missing obligations.

| Plan | Time Step | Actions |
|---|---|---|
| 1 | 1 | transfer(d1,s2,s3,marketing) |
| | 2 | transfer(d1,s3,s4,marketing) |
| | 3 | analyse(d1,s4,marketing) |
| | 4 | transfer(analyseOut(d1,marketing),s4,s7, ↪ marketing) |
| | 5 | transfer(analyseOut(d1,marketing),s7,s5, ↪ marketing) |
| 2 | 1 | transfer(d1,s2,s1,marketing) |
| | 2 | transfer(d1,s1,s4,marketing) |
| | 3 | analyse(d1,s4,marketing) |
| | 4 | transfer(analyseOut(d1,marketing),s4,s7, ↪ marketing) |
| | 5 | transfer(analyseOut(d1,marketing),s7,s5, ↪ marketing) |
| 3 | 1 | transfer( d1, s2, s3, marketing) |
| | 2 | transfer( d1, s3, s4, marketing) |
| | 3 | analyse( d1, s4, marketing) |
| | 4 | transfer( analyseOut(d1, marketing), s4, ↪ s3, marketing) |
| | 5 | transfer( analyseOut(d7, marketing), s3, ↪ s5, marketing) |
| 4 | 1 | transfer(d1,s2,s1,marketing) |
| | 2 | transfer(d1,s1,s4,marketing) |
| | 3 | analyse(d1,s4,marketing) |
| | 4 | transfer(analyseOut(d1,marketing),s4,s3, ↪ marketing) |
| | 5 | transfer(analyseOut(d1,marketing),s3,s5, ↪ marketing) |

Table 4.1: Automatically generated plans.

| Plan | Compliant | Explanation |
|---|---|---|
| 1 and 3 | Yes | - |
| 2 and 4 | NO | missing(transfer(d1,s2,s1,marketing), ↪ art12_22_SubjectRights) missing(transfer(d1,s2,s1,marketing), ↪ chap3_RightsOfDataSubjects) missing(transfer(d1,s2,s1,marketing), ↪ exceptions_as_per_Art23) missing(transfer(d1,s2,s1,marketing), ↪ chap9_Derogations) missing(transfer(d1,s2,s1,marketing), ↪ gdpr_Requirements) |

Table 4.2: Automatic compliance check of plans (Consent)

All the plans are generated automatically by the personal data managing agent. The compliance check process is also done automatically in the second module. Plan 1 and 3 are compliant, since they fulfill all the obligations set forth in GDPR as well as the compliance with consent. Plan 2 and 4 are both noncompliant for the same reason. The action `transfer(d1,s1,s3,marketing ↪ )` lacks the obligation of `art12_22_SubjectRights` since the transfer action does not match the consent provided. When the obligations of a regulation are missed, it also causes that the obligations of the super-class regulations to fail. In this case, the action `transfer(d1,s2,s1,marketing)` misses the obligations of `chap3_RightsOfDataSubjects` and `gdpr_Requirements`, as they are the top classes of regulations in the policy formalization. Two other regulations have been reported as missed obligations, `exceptions_as_per_Art23` and `chap9_Derogations ↪` , this is because if the obligations of these regulations are fulfilled, it causes the transfer action to comply with GDPR.

### 4.2.2 Compliance Check for GDPR Regulatory Norms

In this scenario, we suppose that all the necessary consent is provided, so consent is no longer a restricting constraint. We check the compliance of plans against GDPR regulatory norms, in particular obligations of GDPR Chapter 5 (Transfers of personal data to third countries or international organizations).

According to the SPECIAL policy language[2] a transfer to a third country is only possible if it is not among the unauthorized transfers by the Union law Article 48 (Transfers or disclosures not authorized by the Union law) and is equipped with measures to ensure a secure data transfer; these measures could be one of the appropriate safeguards as in Article 46 (Transfers subject to appropriate safeguards). Again, our aim is to check the compliance of the plans shown in Table 4.1, under the assumption that all necessary consent is provided, but there are no safety measures between the servers outside the EU (third countries) and the servers located in the EU. The resulting compliance report is indicated in Table 4.3.

[2]link to the documentations: `https://specialprivacy.ercim.eu/platform/pilots-policies-and-the-formalization-of-the-gdpr`.

| Plan | Compliant | Explanation |
|---|---|---|
| 1 and 3 | Yes | - |
| 2 and 4 | No | missing(transfer(d1,s2,s1,marketing), ↪ chap5_DataTransferToThirdCountry) missing(transfer(d1,s2,s1,marketing), ↪ adequateLevelOfProtection_as_per_Art45) missing(transfer(d1,s2,s1,marketing), ↪ appropriateSafeguards_as_per_Art46) missing(transfer(d1,s2,s1,marketing), ↪ art49_Derogations) missing(transfer(d1,s2,s1,marketing), ↪ chap9_Derogations) missing(transfer(d1,s2,s1,marketing), ↪ gdpr_Requirements) |

Table 4.3: Automatic compliance check of plans (GDPR Chapter 5)

As shown in Table 4.3, plans 2 and 4 are not compliant, since the action `transfer(d1,s2,s1,marketing)` misses the required obligations of GDPR Chapter 5. The main missing elements are `adequateLevelOfProtection_as_per_Art45` ↪ , `appropriateSafeguards_as_per_Art46`, or `art49_Derogations`. The transfer action could be compliant if certain regulations among the missed ones are satisfied.

## 4.3 Discussions

The goal in this contribution was twofold, (i) designing an agent to handle personal data processing automatically, and (ii) compliance checking of these plans with GDPR. To achieve these goals, we presented a framework for planning and GDPR compliance checking. We used event calculus to formalize agent actions on personal data and time-varying properties of the system. In order to formalize GDPR obligation set and data subjects consent, we used SPECIAL policy language. We described how knowledge of the planning domain and legal knowledge for compliance checking can be represented and integrated. We presented two scenarios in Section 4.2 and showed how our framework can be used to for compliance checking.

The current work can be improved in several ways. We are dependent on the legal ontology or policy language used to represent the GDPR requirements. In the current contribution, we use SPECIAL as the underlying policy language, which is a simple policy language that does not support deontic operations. Using more comprehensive ontologies, such as LegalRuleML (cf. Section 1.3.1), can enrich the current work.

# Chapter 5

# Ethical Compliance
## *A Pluralistic Ordinal Utility Model to Evaluate Processing on Personal Data*

Our specific research problem was the legal and ethical compliance of a planning agent who processes personal data. In this chapter, we investigate it from the point of view of ethical compliance. We are interested in approaches that can morally evaluate and select the ethically best option. The question is, according to what criteria, two options can be morally evaluated? An answer could be using utilities as in utilitarianist theories, to compare options, however, utilities are often not quantifiable or measurable [33]. Moreover, using arbitrary values as utility can not be justified and may lead to contradictions. Another answer could be to use multiple values (cf. Section 2.2.4) as criteria for moral evaluation. Most of the values related to AI are discussed in AI ethics guidelines, e.g. privacy, fairness, transparency, etc. [102, 58]. We are interested in approaches to consider moral values in the evaluation process. However, there are some philosophical and computational challenges.

On the philosophical level, incommonsurability (cf. Section 2.2.4) of values, i.e. lack of a cardinal measure to compare values, poses challenges in modeling these values. Even if the values are not fully incommonsurable, it is not clear how to compare them rationally. On the computational side, moral values may represent abstract concepts or may have broad meanings [190], e.g., fairness, which makes their representation challenging. In addition, there is no agreement on a fixed list of AI values [162] or a universal interpretation of them. Moreover, obtaining the evaluative criteria from fundamental principles is not always feasible [138]. Some computational models allow the integration of multiple values in their modeling of utilitarian theories [8, 7]. However, they explore quantifiable cases and aggregate values by weighing and summing. In certain cases, it is possible to rationally compare options by quantification. However, it may add arbitrary variables to the process that can bias the decision or lead to counter-intuitive results. Other methods suppose utilities are given [122], elicited from a known source for specific cases [120], or use arbitrary values to explore ethical dilemmas without proposing a method to actually obtain them [48, 23].

In this contribution, we propose a model for evaluating alternatives based on their alignment with moral values. In many cases, intuition about which option is better than another is easier to determine than a precise quantification. Thus, we assume that values are incommonsurable but comparative in one of the following ways. Given two values, A and B:

1. A and B are neither better or worse than each other;
2. Any amount of A, no matter how small, is more valuable than any amount of B, no matter how large.

We model the evaluation of each value on the set of alternatives on an ordinal scale. An ordinal preference allows for the logical reasoning that is essential for ethical decision-making. In order to represent values in a disambiguate way, we adopt a hierarchical multi-criteria model. Each criterion represents a certain aspect of a moral value that can be used to order alternatives. The hierarchical criteria structure provides an explicit and explanatory representation of moral

values and increases expressivity of the model. Given the specified preference of two values on ordinal scale, in the *case 1*, we aggregate preferences using voting systems, in particular Copland's rule (cf. Section 3.3) and in *case 2*, using lexical ordering. We demonstrate this model in a use case of choosing the best ethically aligned recommendation system. To implement this use case, we use ASP (cf. Section 3.1.2), the full implementation codes are available in Appendix B.

This chapter is organized as follows. In Section 5.1 we introduce a formalization of our proposed model and discuss the aggregation process. We make use of a case study for ethical ordering of various recommendation systems. The examples run alongside formalizations to facilitate understanding of the abstract model. In Section 5.2 we discuss the conclusions and future perspectives.

## 5.1 Integration of AI Values

In this section, we introduce a formalization of the model along with a use case to facilitate understanding of the abstract model. We are interested in evaluating the adherence or alignment of alternatives to a set of moral values. To consider moral values in the decision process, we view them as criteria in a hierarchical structure that can be decomposed into more basic, comparable subcriteria. Evaluations are modeled as ordinal preference relationships over alternatives.

More precisely, our framework aims to order a set of given alternatives based on their adherence to a set of moral values. We indicate our model by A *Hierarchical Value Setting* $H$, represented by the following tuple:

$$\langle \mathcal{A},\ \langle N,\ \mathbf{R}\rangle,\ \rho,\ \mathbf{\Psi}\rangle \tag{5.1}$$

- $\mathcal{A}$ is the set of alternatives;
- $\langle N,\ \mathbf{R}\rangle$ represents the hierarchy of criteria;
- $\rho$ is a function that assigns an order to the criteria of the bottom level in the hierarchy;
- $\mathbf{\Psi}$ is the family of aggregation functions.

We describe each component of the model on a use case of ordering recommendation systems based on their alignment to moral values. This problem can be of interest to service providers such as on-line job platforms, especially in cases where they collaborate with several partners to provide job recommendations to their users. It is also related with several ethical concerns, users' personal data are used in order to obtain recommendations which concern privacy and fairness. The distribution of recommendations is also related to fairness and performance of the overall recommendations, which are related to user satisfaction and benefit of the system.

### 5.1.1 $\mathcal{A}$: The Set of Alternatives

$\mathcal{A}$, $|\mathcal{A}| \geq 2$ is the set of alternatives that will be ordered in the model based on their alignment to a set of moral values in $H$. Alternatives have different meanings depending on the context; however, they represent or refer to an action or a mixture of actions.

We show the evaluation process of our framework on a simplified problem of ordering 3 recommendation systems, namely `{sys1, sys2, sys3}`. Each alternative is represented in ASP using the predicate `alt/1`.

```
alt(sys1).
alt(sys2).
alt(sys3).
```

### 5.1.2 $\langle N, \mathbf{R} \rangle$: The Criteria Hierarchy

The criteria hierarchy is a tree-like graph represented by the tuple $\langle N, \mathbf{R} \rangle$ where $N$ is the set of nodes that represent criteria, $\mathbf{R} \subseteq N \times N$ is the child relation that assigns to each parent node its child nodes, in other words, this relation associates each criterion with its sub-criteria in a hierarchical structure. We denote the set of subcriterion or the children of a node $n$ by $r(n) = \{x | (n, x) \in \mathbf{R}, n \in N\}$. The root node in this structure is the top level criteria and is denoted by $\epsilon \in N$. The nodes at the bottom of the structure (leaves) represent evaluative criteria, i.e., criteria that can be evaluated based on the characteristics of the given alternatives. The set of leaf nodes or leaf criteria is represented by $L = \{n \in N | r(n) = \emptyset\}$.

The corresponding criteria hierarchy for the problem of ordering the recommender systems is specified by adopting the applying guidelines. We first describe what moral values are taken into account and by which criteria the alternatives are evaluated. Ethical evaluation in this context (cf. Section 2.1) concerns *i) privacy* of users, since these systems process personal data. *ii) Fairness* of the system with respect to jobs and users, and *iii) Performance* of the system to provide the best recommendations. Thus, the set of values to be integrated into the evaluation framework in this case is $\{Privacy, Fairness, Performance\}$. Next, we identify how we can compare the alternatives with regard to these moral values. In other words, what are the sub-criteria of the given moral values that can evaluate alternatives.

- **Privacy** Privacy is one of the most concerning issues. We use the guidelines mentioned in Section 2.1.1 as a criterion to compare privacy in different alternatives. The applying guidelines in this case are the following.
  - **Data Minimization** limiting access to data categories is an important and obvious criterion for preserving the privacy of data subjects. A system that uses fewer categories of personal data is ordered higher by this criterion.

  - **Data Sensitivity** Another important criterion is the number of sensitive categories of data, e.g. political, racial, etc. Processing these types of data entails an increased ethical risk to privacy.
  - **Scale and Complexity** This is another important criterion related to privacy. Using large-scale processing, that is, big data processing with multiple unknown sources of personal data, increases the risk of privacy [97]. According to this criterion, small-scale processing is ordered higher than large-scale big data processing.

- **Fairness** The other value concerning mainly the processing in our use case is fairness (cf, Section 2.1.2). In our context, fairness can be translated as (*Item Fairness*) or users (*User Fairness*)[188]. User fairness can also be decomposed into gender fairness ( *gender fairness*) and racial fairness ( *race fairness*). Bias in this case can be estimated *statistical parity difference* which is the difference in the ratio of favorable recommendations between two monitoring groups. A recommendation system that has a lower statistical parity difference among the monitored groups is ordered higher by fairness criteria. We do not enter into details of calculating parity or other scores. However, we suppose a score indicating the bias level is given.

- **Performance** represents the intrinsic value of the recommendation system that has been designed to serve. The notion of performance has various significations according to the context; here, it represents how fit the recommendations are and if the users are reacting positively to the generated recommendations. A system that has a better performance score has more adherence to this value. Here, we suppose that performance represents a single criterion without any sub criteria.

The value composition diagram in this use case is given by Figure 5.1. We use the predicate `child/2` in ASP to represent the hierarchy of criteria.

```
child(root,privacy).
child(root,fairness).
child(root,performance).

child(privacy, sensitivity).
child(privacy, minimization).
child(privacy, scaleComplexity).
child(fairness, item_fairness).
child(fairness, user_fairness).
child(user, racial_fairness).
child(user, gender_fairness).
```

### 5.1.3 $\rho$: Leaf Criteria Assessment

$\rho$ is the function of evaluating alternatives based on the bottom level or leaf criteria. This evaluation is modeled as an ordinal preference over alternatives

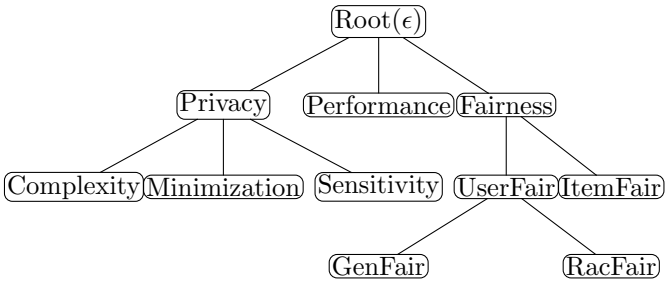



Figure 5.1: Criteria hierarchy diagram

| **Characteristics** | **Sys1** | **Sys2** | **Sys3** |
|---|---|---|---|
| Data Categories | dataHabit, activity, interests | interests, politicalBelief | activity, interests |
| Performance metric | 30% | 25% | 20% |
| Process Scale | large | large | small |
| Gender parity | 0,1 | 0,2 | 0,3 |
| Racial Parity | 0,2 | 0,4 | 0,1 |
| Item Parity | 0,3 | 0,6 | 0,4 |

Table 5.1: Recommendation algorithms' features

and is represented as the following function.

$$\rho : L \longmapsto 2^{\mathcal{A}\times\mathcal{A}} \tag{5.2}$$

Where $\forall l \in L,\ \rho(l)$ is a preference relation, given as a complete pre-order.

In order to apply the framework to our case study, we need to assess the preference of the leaf criteria over alternatives according to their characteristics. First, we must represent these characteristics or the knowledge about each recommendation system that can be used for ethical evaluations. These characteristics include i) knowledge about the required categories of data, ii) the performance metric of each system, and iii) the scale and complexity of the underlying algorithm, and the statistical parity for iv) gender fairness, v) racial equality, and vi) the statistical parity for item fairness. The knowledge of the available systems and their characteristics mentioned are shown in Table 5.1

This knowledge has been represented in the ASP using the predicate `has/3`

| **Leaf Criteria** | **Preference** |
|---|---|
| Minimization | sys2 $\sim$ sys3 $>$ sys1 |
| Sensitivity | sys1 $\sim$ sys3 $>$ sys2 |
| ScaleComplexity | sys3 $>$ sys1 $\sim$ sys2 |
| Performance | sys1 $>$ sys2 $>$ sys3 |
| Gender fairness | sys1 $>$ sys2 $>$ sys3 |
| Racial fairness | sys3 $>$ sys1 $>$ sys2 |
| Item fairness | sys1 $>$ sys3 $>$ sys2 |

Table 5.2: Leaf ordering of the recommendation systems

↪ . Note that since Clingo has difficulties with grounding float numbers, so we represent them using integers, for example:

```
    has(sys1,requiredData,clickHabit ).
    has(sys1,requiredData,activity ).
    has(sys2,requiredData,interests ).
    ...
    has(sys1, perfMetric, 30 ).
    has(sys2, perfMetric, 25 ).
    has(sys1, gender_parity, 1 ).
    has(sys2, racial_parity, 4 ).
    has(sys3, item_parity, 4 ).
```

Having these characteristics, the associated functions for the leaf criteria evaluate the alternatives in a pairwise manner based on the relevant characteristics of each alternative. The preference of the leaf criteria is represented by the predicate `pref/3`. We assume that the preferences are complete and transitive. These preferences are represented as orders in Table 5.2. The preference of the criteria are represented, by the relations $\sim$ and $>$ for simplicity. $\sim$ means two alternatives are equally preferred and $>$ implies strict preference.

```
    % Minimisation
has(Alt,nbData, N):-
    N = #count{ Data : has(Alt, requiredData, Data)},
    alt(Alt).

pref(minimization,Alt1,Alt2):-
    has(Alt1, nbData, N1),
    has(Alt2, nbData, N2),
    N2>=N1.

%Sensitivity
has(Alt, nbSensitiveData, N):-
```

```
    N = #count{ Data : has(Alt, requiredData, Data),
    has(Data, category, sensitiveData)},
    alt(Alt).
pref(sensitivity, Alt1, Alt2):-
    has(Alt1, nbSensitiveData, N1),
    has(Alt2,nbSensitiveData, N2),
    N2>=N1.

%Scale and Complexity
rankAux(largeScale, 2).
rankAux(smallScale, 1).
pref(scaleComplexity,Alt1,Alt2):-
    has(Alt1, processType, Type1),
     has(Alt2, processType, Type2),
     rankAux(Type1, R1),
     rankAux(Type2, R2),
     R2>=R1, alt(Alt1), alt(Alt2).

% Performance
pref(performance, Alt1, Alt2):-
    has(Alt1, perfMetric, P1),
    has(Alt2, perfMetric, P2),
    P1>=P2, alt(Alt1), alt(Alt2).

% Item Fairness
pref(item_fairness,Alt1,Alt2):-
    has(Alt1, item_parity,P1),
    has(Alt2, item_parity,P2),
    P1<=P2, alt(Alt1), alt(Alt2).

%Item Fairness
pref(racial_fairness,Alt1,Alt2):-
    has(Alt1, racial_parity,P1),
    has(Alt2, racial_parity,P2),
    P1<=P2, alt(Alt1), alt(Alt2).

%Gender Fairness
pref(gender_fairness,Alt1,Alt2):-
    has(Alt1, gender_parity,P1),
    has(Alt2, gender_parity,P2),
    P1<=P2, alt(Alt1), alt(Alt2).
```

### 5.1.4 Ψ: Aggregation functions

$\mathbf{\Psi} = \{\psi_n\}_{n \in N \setminus L}$ is a family of aggregators, used by each node to combine the preference of its children. To each non-leaf node $n$, we associate an aggregator $\psi_n$ defined as a function from a vector of preference relations (whose size is the

number of children of $n$) to a preference relation :

$$\psi_n : (2^{\mathcal{A}\times\mathcal{A}})^{|r(n)|} \longmapsto 2^{\mathcal{A}\times\mathcal{A}} \tag{5.3}$$

**Output of A Hierarchical Value Setting** . Given a hierarchical value setting $H$, using its aggregation function of the preference relation of all its children, we can define the preference relation for each node $n$. We denote by $\mathbb{P}^H = \{P_n^H\}_{n\in N}$ the final set of preference for each node. It is defined inductively as follows:

$$\begin{aligned} P_n^H &= \rho(n) && \text{if } n \in L \\ &= \psi_n(P_{i_1}^H, \ldots, P_{i_p}^H) && \text{if } r(n) = \{i_1, \ldots, i_p\} \end{aligned}$$

Therefore, given two alternatives $a$ and $b$ and a criterion $n$, $aP_n^Hb$ means that $a$ is preferred to $b$ according to criteria $n$. The final evaluation of the model is then given by the preference of the node at the highest level, that is, $P_\epsilon^H$. In the following, since we use a single hierarchical value setting, we remove the superscript $H$ from the $P_n^H$ notation.

We describe the aggregation process that can be used in our hierarchical model to obtain the preference of a parent node based on the preference of its children. To maintain the generality of the problem, we consider a finite set of criteria $C = \{1, \ldots, |C|\}$, such that each criterion $i \in C$ specifies a transitive and complete preference over the set of alternatives $P_i \subseteq D$, where $D = 2^{\mathcal{A}\times\mathcal{A}}$. We discuss different rules to aggregate the ensemble of these preferences, as noted by $\mathbb{P} = \langle P_1, \ldots, P_{|C|}\rangle \in D^{|C|}$, according to the available information on the importance and priorities of the criteria. The preference obtained by the aggregation rule is denoted by $P \subseteq D$.

### Equal Criteria

When a criterion $c_1$ is neither better nor worse than $c_2$, their corresponding preferences can be viewed as votes over alternatives. In such a case, the aggregation rules in the social choice and voting theory can be used to combine the preferences of the criteria. We denote this aggregator by $\psi_v : D^{|C|} \longmapsto D$ and the obtained preference by $P^v = \psi_v(\mathbb{P})$. We suppose that all criterion have equal importance. There are general properties that are desired in voting rules, like Pareto's efficiency, monotonicity, etc. An important property in our case is the Condorcet principle, the lack of which may cause a cyclic preference and lead to loss of transitivity. We limit our choice to the rules that satisfy these properties, e.g. Copeland's rule, max-min, etc. For example, the Copeland rule chooses the alternative that wins the most pairwise majority, for each $a, b \in \mathcal{A}$ let $r_{ab}$ be defined as follows:

$$r_{ab} = \begin{cases} 1 & |S_{ab}| > |S_{ba}| \\ \frac{1}{2} & |S_{ab}| = |S_{ba}| \\ 0 & |S_{ab}| < |S_{ba}| \end{cases} \tag{5.4}$$

Where $S_{xy} = \{i \in C | xP_iy \wedge \neg yP_ix\}$, thus, according to Copeland's rule, an alternative wins in pairwise comparison if its overall score defined by the following relation is higher.

$$aP^vb \Leftrightarrow \sum_{c\in\mathcal{A}} r_{a,c} \geq \sum_{c\in\mathcal{A}} r_{b,c} \tag{5.5}$$

Note that this is an example of a voting rule that have our desired properties, other rules that satisfy the Condorcet principle can be used interchangeably.

### Dominant Criteria

When any increase in a criterion $C1$ is not as valuable as that of the criteria $C2$. In this case $C2$ is a dominant criterion. There may be one or a class of such criteria that dominates the others, meaning that their preference has an absolute priority over other criteria. We suppose that there are $k \leq |C|$ classes of criteria such that $C = \cup_{i\in\{1,\ldots,k\}} C_i$, and the priority relation between the classes is given by a total strict order $\succ^s \subseteq 2^C \times 2^C$.

One way to aggregate criteria preferences in this case is to obtain the votes in each class and order them in lexicographic order. More precisely, consider the aggregation function $\psi_l : D^{|C|} \times C^2 \longmapsto D$ and its resulting preference $P^l = \psi_l(\mathbb{P}, \succ^s)$. The aggregation method in this case can be formulated as follows, that is, a combination of voting rule and lexicographic aggregation, and we call it the lexical aggregation function.

$$\begin{aligned} a\ P^l\ b \iff & \exists\ i \in \{1,\ldots,k\} \wedge a\ P^v_{C_i}\ b \wedge \\ & (\forall\ j \in \{1,\ldots,k\} \wedge C_j \succ^s C_i\ \Rightarrow a\ P^v_{C_j}\ b) \end{aligned} \tag{5.6}$$

where $a\ P^v_X\ b = \psi_v(\mathbb{P}^X)$, $\mathbb{P}^X \subseteq D^{|X|}$ is the vector of preference of the criteria in a set $X \subseteq C$, and $\psi_v$ is the aggregation function based on a voting rule as discussed in the previous section. When the only partition is $C$, then $P^l$ is equivalent to $P^v$.

We now describe the aggregation process for every parent node in our use case, given a preference over their children. We assume that these preferences are complete and use the lexical aggregation function to obtain the preference of the parent nodes. The root node is the top node that represents the final decision criteria. Here, we consider multiple priority settings for the root node to show the functionality of the aggregator and expressivity of our model, as in Table 5.3.

We represent the priority of the sub-criteria using the predicate `childPref/3` in ASP as shown below.

```
childPref(privacy, sensitivity, minimization).
childPref(privacy, minimization, scaleComplexity).

childPref(fairness, user_fairness, item_fairness).
```

| **Parent node** | **Children priorities** |
|---|---|
| Privacy | $\{Sensitivity\} \succ^s \{Minimization\} \succ^s \{Complexity\}$ |
| User Fairness | $\{genderFairness, racialFairness\}$ |
| Fairness | $\{userFairness\} \succ^s \{itemFairness\}$ |
| Root (1) | $\{Privacy,\ Fairness,\ Performance\}$ |
| Root (2) | $\{Fairness\} \succ^s \{Privacy\} \succ^s \{Performance\}$ |
| Root (3) | $\{Fairness, Privacy\} \succ^s \{Performance\}$ |
| Root (4) | $\{Fairness, Performance\} \succ^s \{Privacy\}$ |

Table 5.3: Given setting for the children nodes

```
childPref(user_fairness, racial_fairness, gender_fairness).
childPref(user_fairness, gender_fairness,  racial_fairness).

childPref(root, fairness, privacy).
childPref(root, privacy, performance).
```

The lexical aggregation function 5.6 has been implemented in the following way in ASP.

```
prefLex(Node, Alt1, Alt2):-
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2),
    is_dominant(Node, Class, Alt1, Alt2).

is_dominant(Node, Class,Alt1,Alt2):-
    not is_dominated(Node, Class,Alt1,Alt2),
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2).

is_dominated(Node, Class, Alt1,Alt2):-
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2),
    superiorThan(Node, Class1, Class),
    not pVote(Node, Class1, Alt1,Alt2).

is_dominated(Node, Class, Alt1,Alt2):-
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2),
    not superiorThan(Node,  _, Class),
    pVote(Node, Class, Alt2, Alt1).

superiorThan(Node, Class1, Class2):-
    childrenClass(Node, Class1),
```

```
    childrenClass(Node, Class2),
    not inferiorThan(Node, Class1, Class2).

inferiorThan(Node, Class1, Class2):-
    childrenClass(Node, Class1),
    childrenClass(Node, Class2),
    belongs(Node, Child1 , Class1),
    belongs(Node, Child2 , Class2),
    not childPref(Node, Child1, Child2).

childrenClass(Node, Class):-
    belongs(Node, _ , Class).

belongs(Node, Child , Class):-
    Class = #count{ Child1 :
        childPref(Node , Child, Child1)},
    child(Node, Child).
```

The predicate `pVote/4` in the code above corresponds to a voting rule. In our case, we have used Copeland's rule in order to aggregate votes in each class. The rule is translated in ASP in the following way, which is an implementation of 5.5:

```
pVote(Node, Class, Alt1, Alt2):-
    plural(Node, Class),
    copeland_score( Node, Class, Alt1, S1),
    copeland_score( Node, Class, Alt2, S2),
    S1>=S2.

copeland_score( Node, Class, Alt, S):-
    S= #sum{ S1:
        nb_pairwise_wins( Node, Class, Alt, Alt1, N1),
        nb_pairwise_ties( Node, Class, Alt, Alt1, N2),
        S1 = N1*2+N2,
        alt(Alt1)},
    childrenClass(Node, Class), alt(Alt) .

nb_pairwise_ties( Node, Class, Alt1, Alt2 , N):-
    N= #count{ N1 :
        nb_strict_voters(Node, Class, Alt1, Alt2 , N1),
        nb_strict_voters(Node, Class, Alt2, Alt1 , N2),
        belongs(Node, Child, Class), N1=N2},
    childrenClass(Node,Class), alt(Alt1), alt(Alt2).

nb_pairwise_wins( Node, Class, Alt1, Alt2, N):-
    N= #count{ N1 :
        nb_strict_voters(Node, Class, Alt1, Alt2 , N1),
        nb_strict_voters(Node, Class, Alt2, Alt1 , N2),
        belongs(Node, Child, Class), N1>N2},
    childrenClass(Node, Class), alt(Alt1), alt(Alt2).

```

| Parent Node | Preferences |
|---|---|
| Privacy | sys3 > sys1 > sys2 |
| User Fairness | sys1 > sys2 ∼ sys3 |
| Fairness | sys1 > sys3 > sys2 |
| Root(1) | sys1 ∼ sys3 > sys2 |
| Root(2) | sys1 > sys3 > sys2 |
| Root(3) | sys1 ∼ sys3 > sys2 |
| Root(4) | sys1 > sys2 ∼ sys3 |

Table 5.4: Preference of the parent nodes

```
29 nb_strict_voters(Node, Class, Alt1, Alt2, N):-
30     N = #count{ Child :
31         pref(Child, Alt1, Alt2),
32         not pref(Child , Alt2, Alt1),
33         belongs(Node, Child, Class) },
34     childrenClass(Node, Class), alt(Alt1), alt(Alt2).
```

The resulting orders based on the specified settings are shown in Table 5.4. As mentioned previously, we consider several cases for the root node in order to highlight the difference in the aggregation result. This shows our framework's ability to take into account contextual preferences on moral values.

When all root children are equally important as in the case of root (1) $Sys1$ or $Sys3$ are the best systems, and $Sys2$ is ranked lower. When fairness is superior to privacy and that is superior to performance, $Sys1$ is the first winner. In case of Root(1) the orders are obtained by collective voting of all moral values, but in case of Root(2) the orders of the dominant voter, i.e., fairness, are lexically preferred to privacy and performance. The other results can be interpreted in the same way.

Our framework can consider the contextual nature of ethical decision-making by considering various specifications as a priority relation among values. For example, in a healthcare scenario, we may wish to prioritize performance over privacy, but in a marketing context, we tend to prioritize privacy over performance.

## 5.2 Discussions

The proposed model for ethical evaluation provides an explicit representation of ethical considerations in terms of moral AI values, clarifies the interpretation of these values, and can take their importance into account by prioritizing

them. An advantage of our model is that orders are obtained by rules of logical aggregation through a reasoning process, which is essential in ethical decision making. Our framework can also serve as a utility function for modeling consequentialist theories. One of the future works is designing an ontology to collect the principles and prescriptions in AI ethics guidelines, which can cover a broad range of moral values.

# Chapter 6

# Legal and Ethical Compliance
## *A Data Processing Use Case with Real-time Execution*

In the previous contributions, we developed legal and ethical compliance models separately. In this chapter, we investigate the issue of both legal and ethical compliance together. An important question is how an agent's actions can be legal and ethical? Although law and ethics overlap on many norms, they are not always the same. There maybe actions that are considered ethical but not legal, or vice versa. In these cases, a conflict between legal and ethical norms occurs. Addressing these challenges in general is challenging. We need to specify the interaction between legal and ethical compliance to identify the possible conflicts in the compliance checking process. We limit our focus to cases where conflicts between legal and ethical norms do not occur. We suppose that the deontic ethical rules are already reflected in the law, and therefore a legal option is also ethically permissible. However, our concern is what alternative is the best among the compliant ones? Here, legal norms can be seen as admissibility criteria, and ethical norms as maximization criteria. We adopted this view in this contribution, i.e. legal norms are seen as hard constraints that must be satisfied, and ethical norms as soft constraints that should be satisfied as much as possible. However, integration of the two components with planning in a single architecture is a challenge.

In this chapter, we propose a model to integrate legal and ethical compliance in an online HTN planning (cf. Section 3.2.2) architecture. In this architecture, legal and ethical compliance is integrated in a modular way. HTN planning handles both planning and execution and reacts to exogenous changes in the environment while executing a plan. An advantage of this formalism is that it enables real-time compliance checking. This contribution was made in collaboration with other researchers, and the resulting work was published in RuleML challenge [93] and in the VECOMP workshop [94].

This chapter is organized as follows. Section 6.1 presents the proposed architecture. In Section 6.2 we explain the use case model that is used as a demonstration. Sections 6.3, 6.4, and 6.5 describe the planning, legal compliance, and ethical evaluation components, respectively. In Section 6.6 we explore some real-time execution scenarios. Section 6.7 discusses the findings and future research directions.

## 6.1 Architecture Review

The overall architecture is composed of three principal components, namely, planning and execution, legal compliance checking, and ethical evaluation. These components and the interaction between them are illustrated in Figure 6.1. The planning and execution component is based on an online HTN planner. Generate plans based on beliefs or facts in the specified domain and execute these plans. This component perceives and records changes in the operating environment and is able to update beliefs and perform replanning. The legal compliance component checks the validity of the generated plans and filters out illegal plans. The ethical checking component then receives legal plans as input, orders them according to some specified ethical setting, and selects the best one to be

executed.

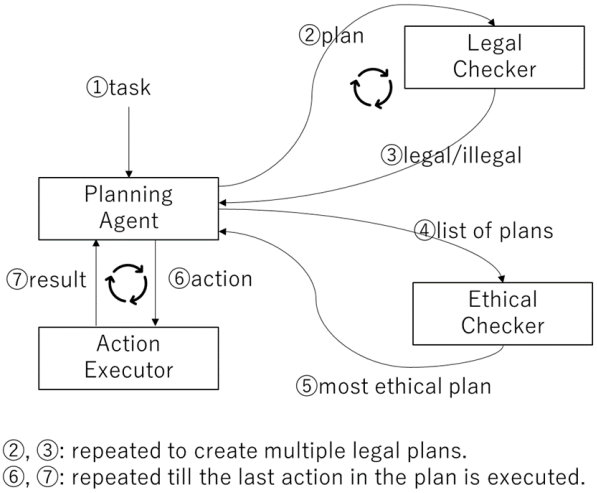



Figure 6.1: Overall system architecture

Given a task (①), the planning component generates a least-cost plan using the best-first search. The legal compliance component receives the plan as input (②) and checks it and sends the compliance state of the plan back to the planner (③). This process is repeated (②-③) until a maximum number of legal plans are obtained. These plans are then fed into the ethical evaluation component (④) where they are ordered, and the best plan is sent to the planning component (⑤). This component executes each action in the selected plan using the action executor (⑥), and communicates the effects or any unexpected changes to the planning component (⑦). The belief set in the planning component is then updated according to the communicated results. The belief updates might affect the possible plans or their legal/ethical evaluations, in this case a replanning is triggered (②-⑤).

## 6.2 Use case model

In order to show the characteristics and efficiency of our proposed approach, we apply it in a data transfer and processing situation. A similar use case model has been used in Section 4.1.1 as a demonstration of GDPR compliance of data processing. The model mainly includes multiple nodes that are used to transfer or process data and are connected as illustrated in Figure 6.2. Each node represents a section of a corporation that is located at a different location, which may be within the EU or outside the EU. Node 4 (marked as a square) is the central node that serves as a cloud server to process data for different purposes. Other nodes (marked as circles) are used to store and transfer data.

In this use case, users' personal data are stored in circle nodes. Different sections may ask to apply a processing on data and receive the output of the processing at their corresponding node.

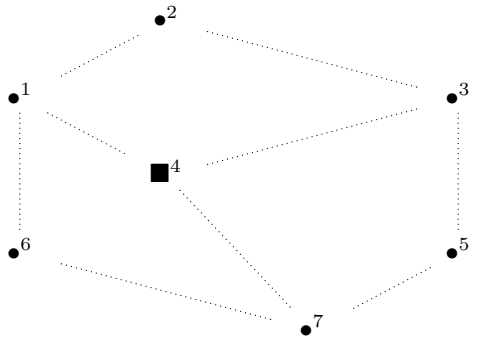



Figure 6.2: Nodes and connections in the network

In order to perform a task, the system locates the data, transfers them to the processing node, and applies a process with the corresponding purpose. After processing personal data, the system delivers the output to the requested node. The planner in our architecture generates possible plans to satisfy the given task, i.e. the possible paths to transfer data and process them in the network. Each possibility represents different behaviors of the system. According to this architecture, these behaviors are verified by the legal checker for any infringement of the (modeled) regulations. The legal checker rules out the illegal plans, and the remaining plans are ordered by the ethical checker based on their alignment with the ethical values (cf. Chapter 5).

There is additional information on this use case that enables testing our architecture in different scenarios. Table 6.1 shows the information on the nodes. The *region* is the location of each node. Since our focus is particularly on GDPR, the regions are categorized as *EU* and *NonEU*. The region of the node is used in the legal verification process. Transferring personal data outside the legislative zone may have ethical implications for data subjects; it is also used in the ethical verification process. The *safety level* corresponds to the safety protocols supported by each node that can be high, medium, or low. Transferring data through more secure nodes is necessary to avoid any possible breach that harms user privacy. Thus, it is important in ethical checking process. The *occupancy level* indicates whether a node is busy. It is used to minimize data management time and improve the overall efficiency of the system.

Table 6.2 shows the processing available to apply on personal data. It includes information on the *location* of processing that is node 4 and the *purpose* that is *recommendation* for all processing in this case. The *bias level* shows the extent to which a processing can be biased with respect to a certain group. We

| Node | Region | Safety Level | Occupancy Level |
|---|---|---|---|
| 1 | EU | medium | normal |
| 2 | non EU | medium | normal |
| 3 | EU | medium | busy |
| 4 | EU | high | busy |
| 5 | E | high | normal |
| 6 | EU | low | busy |
| 7 | non EU | high | normal |

Table 6.1: The attributes of each node

show this simply by positive integers. Each processing requires certain *categories* of data which are indicated by a list and the category name, e.g. c1, c2, etc. Some of the categories are considered sensitive.

| Processing | Location | Purpose | Bias Level | Required Categories |
|---|---|---|---|---|
| P1 | node 4 | recommendation | 2 | [c1,c2,c3,c4] |
| P2 | node 4 | recommendation | 1 | [c2,c3,c5] |
| P3 | node 4 | recommendation | 3 | [c1,c3,c6,c7,c8] |

Table 6.2: The information of available processing

Last but not least, Table 6.3 shows information on personal data. This includes their corresponding *category*, the node on which the data are *stored*, and the data subject who is the *owner* of the personal data.

| Data | Category | Storage Location | Owner |
|---|---|---|---|
| du11 | c1 | node 1 | user 1 |
| du12 | c2 | node 1 | user 1 |
| ... | ... | ... | ... |
| du27 | c1 | node 2 | user 2 |
| du28 | c2 | node 2 | user 2 |

Table 6.3: The information on personal data

## 6.3 Planning Component

The planning component uses the online HTN planning algorithm described in Section 3.2.2. Given a specification of the planning domain described in the previous section, this component generates a sequence of data operations to

perform a task. This component also monitors the execution of a plan and can modify it based on exogenous changes in the environment. The domain specification includes primitive and abstract tasks, as well as the belief set, which represents facts and fluents in the domain.

### 6.3.1 Belief Set

The belief set in our use case domain includes the information about nodes, processing, and data. part of the belief set used to represent this information in Prolog are discussed in this section.

**Nodes** The nodes and their connections are represented using the predicate `arc/2`. The predicate `connected/2` indicates that each arc among two nodes is bidirectional. Information about the region in which a node is located is also represented using the predicate `nodeRegion/2`. The region of a node indicates above all the legislative zones, as processing and transferring personal data to other regions may pose legal and ethical issues, therefore it needs to be checked.

```
1   belief(arc(node1,node2)).
2   ...
3   belief(arc(node2,node3)).
4   ...
5   belief(arc(node4,node7)).
6
7   belief(connected(Node1,Node2),[arc(Node1,Node2)]).
8   belief(connected(Node2,Node1),[arc(Node1,Node2)]).
9
10  belief(nodeRegion(node1,nonEu)).
11  ...
12  belief(nodeRegion(node3,eu)).
13  ...
14  belief(nodeRegion(node5,eu)).
```

The efficiency of data transfer depends on the occupancy and safety of the nodes that are shown, respectively, using the predicates `nodeOccupancy/2` and `nodeSafety/2`. Occupancy of a node could be *normal* or *busy*, and the safety of a node is a dynamic fluent that can take values *low*, *medium*, or *high* depending on the safety protocols and security conditions of a node.

```
1   belief(nodeOccupancy(node1, normal)).
2   ...
3   belief(nodeOccupancy(node6, busy)).
4
5
6   belief(nodeSafety(node1, med)).
7   ...
8   belief(nodeSafety(node4, high)).
9   ...
10  belief(nodeSafety(node6, low)).
```

**Personal Data** The company is using personal data in order to generate job recommendations for the users, we show a set of personal data as a list in Prolog. The node in which dataset are stored is represented by `dataSetAt/2`, for example, `dataSetAt([du11,du12,du13,du14], node_user1)` means that the dataset which contains personal data `du11,du12,du13`, and `du14` is located in `node_user1`. in addition to the storage location, we also need information on the data categories shown by `dataCategory` and whether a certain category of personal data is sensitive or not, shown by `categoryType`. This knowledge may be essential for legal and ethical evaluations in our architecture.

```
belief(dataSetAt([du11,du13,du16,du17,du18], node_user1)).
belief(dataSetAt([du11,du12,du13,du14], node_user1)).

belief(dataCategory(du11, c1)).
belief(dataCategory(du12, c2)).
...
belief(dataCategory(du18, c8)).
```

**Processing** Each processing in this use case is represented as an object located in a certain node with a particular purpose using predicates `processAt/2`, `processPurpose/2`. The bias measure for each algorithm is also recorded and is shown by the predicate `bias_metric/2`. This knowledge is used for ethical checking, for simplicity, we use positive integers as bias values.

```
belief(processAt(p1, node_cloud1)).
belief(processAt(p2, node_cloud1)).
...
belief(processPurpose(recommendation, p1)).
belief(processPurpose(recommendation, p2)).

belief(bias_metric(p2, 3)).
belief(bias_metric(p1, 2)).
```

### 6.3.2 Tasks

Two types of tasks can be defined in the planning component, primitive and compound tasks. Both are explained in this section.

**Primitive Tasks** The primitive tasks in our use case include transfer and run. These actions are formulated in HTN and described in this section. The action `transfer` transfers the data from one node to an adjacent node.

```
action(transfer(Dataset , NodeOrig, NodeDest, _),[
    dataSetAt(Dataset, NodeOrig),
    connected(NodeOrig, NodeDest),
    diff(NodeOrig, NodeDest)
],[
    initiates(dataSetAt(Dataset, NodeDest)),
```

```
        terminates(dataSetAt(Dataset, NodeOrig))
    ]).
```

The above rule specifies that the preconditions of the *transfer* action are `dataSetAt(Dataset, NodeOrig)`, `connected(NodeOrig, NodeDest)`. It also specifies that the effects of the action are to initiate `dataSetAt(Dataset, NodeDest)` and to terminate `dataSetAt(Dataset, NodeOrig)`.

The action `run` effectuates a certain process on the specified data set with the specified purpose. The data set must be stored at the same node as the process. The process output, i.e. recommendations, is obtained as new data set after the processing, and the original data is removed.

```
    action(run(Process, Dataset, Purpose),[
            processPurpose(Purpose, Process),
            requiredDataSet(Process, Dataset),
            dataSetAt(Dataset, Node),
            processAt(Process, Node)
        ],[
            initiates(dataSetAt([output(Process,Dataset)], Node)),
            terminates(dataSetAt(Dataset,Node))
        ]).
```

The `run` action specifies that the preconditions of the action are that, the data set, and the process should be at the same location and `Process` should be an available processing for the specified purpose. It also specifies that the effects of the action are to initiate `dataSetAt([output(Process,Dataset)], Node)` , and to terminate `dataSetAt(Dataset,Node)`. The other primitive action is `load` which is defined in the same way. This actions load the data set from the data base to the server.

**Compound tasks** `multiStepTransfer` is a compound task for transferring data to a destination node in multiple steps. This task is decomposed to a transfer action from the origin node to an adjacent node, and a multistep transfer from the adjacent node to the next one, until the data is reached to the destination node.

```
    htn(multiStepTransfer(Dataset,NodeFrom,NodeTo,Purpose),[
            dataSetAt(Dataset,NodeFrom,Purpose),
            connected(NodeFrom,Node)
            ],[
            transfer(Dataset,NodeFrom,Node,Purpose),
            multiStepTransfer(Dataset,Node,NodeTo,Purpose)
            ]).

        htn(multiStepTransfer(Dataset,Node,Node,Purpose),[
            dataSetAt(Dataset,Node,Purpose)
        ],[
        ]).
```

This compound task requires, the data set to be located at the origin node `dataSetAt(Dataset,NodeFrom)` and the adjacent node to be connected to both the origin node `connected(NodeFrom, Node)`.

The top-level task is to `processHandling` that process some users' data for a particular purpose. In order to perform this processing, the data set is first loaded on a node, then transferred to the processing node, and after applying the process, the result is transferred to the destination node.

```
htn(processHandling(User, NodeTo, Purpose),[
    availableProcess(Purpose, Process),
    processAt(Process, NodeProcess),
    requiredDataSet(Process, User, Dataset),
    contains(Database, Dataset),
    dataBaseAt(Database, NodeOrig)
],[
    load(Dataset, NodeOrig, Purpose),
    multiStepTransfer(Dataset, NodeOrig, NodeProcess, Purpose),
    run(Process, Dataset, NodeProcess, Purpose),
    multiStepTransfer([output(Process, Dataset)],NodeProcess, NodeTo,
↪   Purpose)
]).
```

In order to perform this task, a processing must be available for this purpose on a certain node `availableProcess(Purpose, Process)`, `processAt(Process,` ↪ `NodeProcess)` on the one hand, and on the other hand, the required data set to perform that process must be stored on some node's database.

### 6.3.3 Planning Example

Here, a simple example of planning is presented. The objective task is to process the data set of the user $u2$ for the purpose of the "recommendations", and deliver the result at "node 5". This objective is represented as the task `processHandling` ↪ `(u2, node5, recommendation)`. Then, the planner, decomposes this task to executable actions.

The dataset of the user $u2$ is saved in a database at node 2. The objective task is decomposed in the following way.

1. *load* the required data at node 2
2. *transfer* data through node 1 or 3 to node 4
3. *run* the selected process(P1/P2/P3) on data set
4. *transfer* the output to node 5 through node 3 or 7

Each choice of the processing and the transfer path leads to a different plan. These plans are shown in Figure 6.3.

Here there are 4 pathways and 3 processing, so 12 possible plans are generated. An example of a plan is the following.

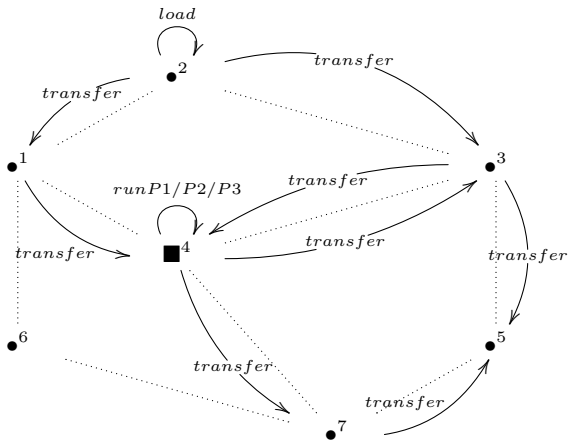



Figure 6.3: Examples of possible plans

```
load( [du21,du22,du23,du24], node2,recommendation),
transfer( [du21,du22,du23,du24],node2, node1,recommendation),
transfer( [du21,du22,du23,du24],node1, node4,recommendation),
run( p1, [du21,du22,du23,du24], node4, recommendation),
transfer( [output(p1,[du21,du22,du23,du24])], node4, node7,
↪ recommendation),
transfer( [output(p1,[du21,du22,du23,du24])], node7, node5,
↪ recommendation)
```

This plan is shown in Figure 6.4 can be translated as:

1. load data set `[du21,du22,du23,du24]`;
2. transfer it to node 1;
3. transfer it to node 4;
4. run the process P1;
5. transfer the data to node 7;
6. transfer the data to node 5.

## 6.4 Compliance Component

The compliance component verifies if a data processing sequence complies with the specified obligations. A representation of GDPR regulations in the form of policies is used to verify compliance. We have translated the SPECIAL policy

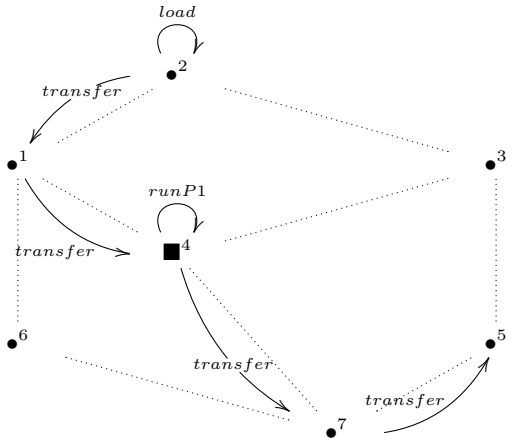



Figure 6.4: Example of a single plan

language (cf. 1.3.2) into Prolog in a very similar way, as in Section 4.1.3. The compliance check process is the same as explained in Chapter 4.

Consider the example of generating recommendations for user $u2$ (cf. Section 6.3.3). In this example, the user $u2$ has given their consent to process and transfer their data set only inside Europe. The data category $c4$ is not authorized to process for the purpose of recommendation. In this case, applying the process $P1$ is not permitted, because $c4$ is among its required data categories `[ ↪ c1,c2,c3,c4]`. In addition, transferring data through *node*7 is not allowed, because it is located outside Europe. In this case, the compliance component, identifies and remove these plans. Here, in the same way as in Chapter 4, the compliance component can highlight the missing obligations and requirements of GDPR in case of non-compliance.

After the compliance checking, all plans that involve $P1$ as the processing and transfer through *node*7 are removed by the compliance component. Here from the total of 12 plans only 4 plans will be left. i.e. the plans that choose either $P2$ or $P3$ and do not transfer data through *node*7.

## 6.5 Ethical Evaluation

The ethical evaluation component is responsible for evaluating and selecting the best plan among the valid ones. The evaluation mechanism is based on the model introduced in Section 5.1. Since in this contribution ethical norms are considered as soft constraints, we call them *soft norms* instead of criteria as in Section 5.1. In order to compare the input plans, they are evaluated according to each soft norm. This process is depicted in Figure 6.6 These evaluations are represented on an ordinal scale, i.e., each soft norm orders the plans according to

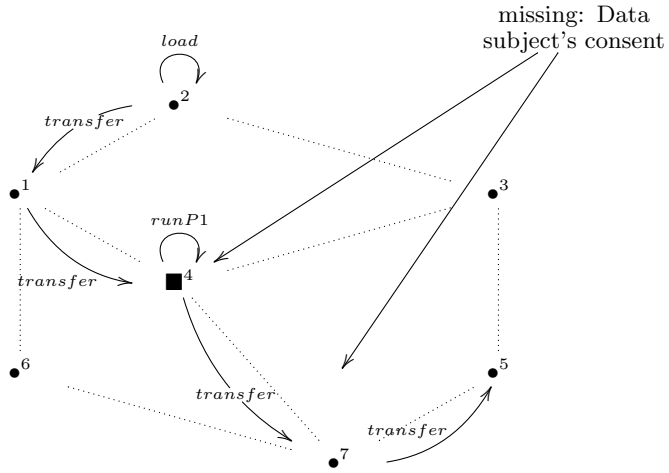



Figure 6.5: Compliance checking

its underlying criteria. After ordering plans according to multiple soft norms, they need to be aggregated to in order to select the best plan. We consider two types of aggregation behavior. An order may be superior to another; in that case, the aggregated order is similar to the superior one, and the inferior order is only taken into account when two alternatives have an equal order i.e. lexicographical ordering. When two (or more) orders can be compromised, they are seen as votes and aggregated by a (suitable)voting rule. Finally, all orders can be aggregated by specifying the superiority and compromise relationship among soft norms. The ethical evaluation component select the best plan by identifying which one is more aligned with the given specification of values.

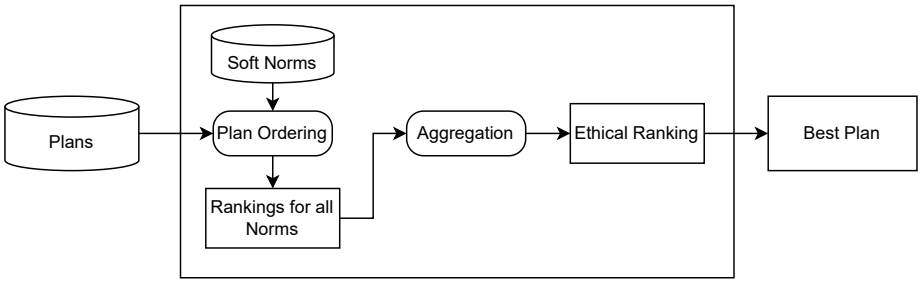



Figure 6.6: Ethical evaluation process

The ethical evaluation component uses domain knowledge to order plans and identify the best one. It takes as input a list of legal plans and orders them according to the specified soft norms. The orders can be seen as the votes

of each norm on the input plans. The soft norms used in our use case are indicated below. These norms concerning processing of personal data are partly taken form AI ethics guidelines (cf. Section 2.1).

- **Number of categories**: Using fewer personal data categories respect the privacy of the user.
- **Number of sensitive data**: Using less sensitive data avoids risks to the privacy and safety of the user.
- **Transfer regions**: Avoiding transfers outside the legislative zone protects personal data and the user's safety.
- **Node safety**: using safer nodes respect the safety of the user personal data and minimize the risk of any breach.
- **Occupancy level**: Using less busy nodes increase the efficiency of data transfers and help increase the overall quality of the service.
- **Algorithmic bias**: Using processing that is not or less biased toward any group is more fair.

Having specified all soft norms, the ethical evaluation component, first, orders the plans according to each norm. Then these orders are aggregated according to the given specification. The aggregation process is similar as Section 5.1.

Some examples of the specifications are represented in the following. Norms that have equal importance represent a class and are noted by {.}. among the classes, there is superiority relationship that is indicated by >.

1. $\{nodeSafety\} > \{nbSensitiveData,\ algoBias\} > \{occupancyLevel,\ nbNonEU,\ nbDataCat\}$
2. $\{nodeSafety\} > \{nbSensitiveData\} > \{algoBias\} > \{nbNonEU\} > \{occupancyLevel\} > \{nbDataCat\}$

In the first specification there are 3 classes; orders in each class are considered as votes and are aggregated using Copeland's rule. Then the alternatives, i.e. plans are ordered logically according to votes of each class (cf. Section. 5.1).

In the first specification, $nodeSafety$ has the highest importance than any other soft norms. The next class includes $nbSensitiveData$ and $algoBias$. That means their corresponding orders over plans are to be aggregated using Copland's rule. Similarly, the vote of this class is strictly superior to the class $\{occupancyLevel, nbNonEU, nbDataCat\}$. In the second specification all norms are set in separate classes that ordered plans according to the superiority order.

Contrary to Section 5.1, here we adopted a special case of the pluralist utility model; only with the root node. In other words, we don't consider hierarchical aggregation process in this section. Although this method is more expressive, once the norms are identified, any reasonable aggregation could be applied. An advantage of aggregating orders in this way is that it allows comparing different soft norms (criteria) from other nodes.

## 6.6 Real-time Execution

In this section, we explain in two example scenarios. How a plan is executed and how the planner reacts to the changes in the environment. Especially the changes that affect compliance and ethical evaluation of plans.

### 6.6.1 Scenario I

This scenario aims to show how the system would react to the physical changes in the operating environment, that is, the use case of connected networks explained in Section 6.2. The objective is to process the personal data of the user $u1$ for the purpose of recommendation. The data is initially stored in a database at node 2 and the output of the processing is also requested at the same node. The initial selected plan is to transfer the data through node 1 to node 4, apply the processing $p2$, and transfer the output back to node 2 via node 1. As shown in Table 6.1, node 1 and node 3 have the same values for every attribute except occupancy level, where node1 is less busy than node 3; therefore, node 1 is selected in the initial plan. During execution, when the data are loaded from the database, the system realizes that node 1 is suddenly deactivated. The planner would do a replanning and select node 3 as intermediate to send data to node 4 and apply the processing, and transfer it back via node 3 again. After applying the selected processing, the system recognizes that node 1 has been reactivated. It takes into account the new change by forming a replanning from the current state and chooses node 1 again as the intermediate node.

### 6.6.2 Scenario II

This scenario shows how the system would react to changes that affect the ethical evaluation. The task given in this scenario is to use the personal data of $u1$ to create recommendations and deliver the result at node 5. $u1$'s data is stored at node 1. In order to perform the task, the planner transfers personal data from node 1 to node 4 to run the selected process and choose an intermediary node between node 3 or 7 to deliver the result to node 5. Since the safety level of node 7 is higher, the ethical checker initially selects the plan that transfers data through this node. However, while executing this plan, the system realizes that, due to some external incidents, the safety level of node 7 has changed to *low*. A re-evaluation is then initiated by the system, and the ethical checker selects the path that passes through node 3, because it is safer. In this scenario, the physical constraints are fixed; however, the properties which affect the ordering of the ethical checker, and consequently the selected plan, are changed. The re-evaluation process shows the functionality of our proposed architecture, and the ethical checker component in similar situations.

## 6.7 Discussions

The objective of this contribution was twofold; the first was to bring together the previous contributions to legal and ethical compliance in a single architecture. The second was to design a system that can handle compliance check in real time. These objectives were fulfilled in the proposed architecture. We used an on-line HTN formalism that cannot deal with changes and compliance issues in real-time. Extensions of this work are also published in collaboration with other researchers. In [94] the communications between components are improved so that the system could operate more efficiently. In another work, this architecture is implemented in distributed mode.

Some perspectives remain to be explored in future work. Analyze the computational complexity of the proposed architecture and optimize the underlying algorithms to improve efficiency. A way to improve efficiency in the implementation of the proposed model is to consider legal obligations as preconditions of actions. In this way, we can avoid generating illegal plans. However, since examining the legal checking process was in our interest, the legal compliance module was implemented separately in the current work. Another work could be considering more computationally efficient ways to aggregate orders in the ethical evaluation component and to design an ontology to collect the principles and prescriptions in AI ethics guidelines. A better way to support real-time compliance is to adapt the compliance components to deal with incomplete information.

One of the drawbacks of the proposed architecture is that, since the legal compliance and ethical evaluation components are performed in two separate steps, some interactions cannot be modeled. This is based on the view that whatever is considered legal is also ethical, which may not always be the case. The ethical evaluation component is only viewed as an optimization step, rather than as a verification of permissibility or compliance. In other words, ethical norms are integrated as soft norms, and it is assumed that the hard ethical rules are already reflected in law. This is not always the case, and there may be ethical rules in conflict with legal norms. This distinction between legal and ethical norms is useful in the implementation phase; however, it does not reflect the difference and similarities between law and ethics. In the next chapter, we proposed an architecture for a new use case that integrates both components on a single component that can interact more efficiently.

# Chapter 7

# Unified Legal and Ethical Compliance

## *An Automated Delivery System for Health Care Items*

In Chapter 6 we integrated both legal and ethical compliance in a planning architecture. In the proposed architecture, legal norms were represented as hard constraints, and ethical norms as soft constraints. This is based on the assumption that hard ethical norms are already reflected in the law. From a more fundamental point of view, legality means that an act is in accordance with the law, and ethics is about concepts of right and wrong behavior. Although in many cases the legal and ethical judgments coincide, they also differ fundamentally. For example, testing medicines on animals, cutting an old tree, or eating meat is considered legal in many countries, but some people believe that they are not ethical. There are also actions that may be ethical but not legal, for example, stealing medicine to save someone's life or passing the red light on a quiet street. Moreover, certain laws do not have anything to do with ethics. For example, the law prescribes driving to the right in France and to the left in England. Although these laws prevent chaos on the roads, they have nothing to do with ethics. Although considering legal norms as hard and ethical norms as soft constraints is useful in selecting the best compliant option in many cases, it does not support hard ethical norms. In addition, viewing ethics as soft constraints means that the model does not capture conflicts between legal and ethical norms.

In this contribution, we explore a new architecture for integrating legal and ethical compliance. Instead of treating legal and ethical norms differently in separate components, we merge them in a single component. In other words, we have a single norm repository where they are divided into soft or hard norms. This model allows for the representation of hard ethical norms on one hand and solving the conflicts between legal and ethical norms on the other. In the proposed architecture, a series of combinatorial choices should be made in a heuristic used for planning. Then a compliance check component reviews all the options and selects the best compliant one. Two kinds of norms can be modeled in this architecture, namely absolute and relative. Absolute norms are equivalent to hard norms, and a violation of an absolute norm deems an action impermissible or non-compliant. Relative norms have a comparative norm that helps to identify the best aligned option. Relative norms are equivalent to soft norms.

In order to test this method of integration, we applied it in the use case of an autonomous delivery system in health care. The use case is a realistic model in which the system should automatically make series choices, including planning to fulfill the receiving demands. The system may face different dilemmas where hard ethical and legal norms are in conflict. We also introduce a relaxation mechanism that can be used to solve certain dilemmas. This problem is formulated in ASP, and event calculus is used as a planning formalism. The full implementation codes are available in Appendix C.

This chapter is organized as follows, in Section 7.1 we describe a case study that is used to explain the new architecture. Section 7.2 presents a brief overview of the overall architecture. In Section 7.3 we describe the heuristics used for task assignments and planning. Section 7.4 presents the compliance component in the proposed architecture. Finally, in Section 7.5, we discuss the findings and

future perspectives.

## 7.1 Use Case Model

The use case model describes a demand-delivery system in healthcare through autonomous vehicles. Demands are made for certain resources at different locations. Resources may include medicine, medical equipment, and organs in rare cases. This use case has applications in smart cities and logistics. There are currently well-known companies that have deployed such delivery systems in limited cases and are making experiments and investments to further develop them. In this use case, a system operates several autonomous agents, including autonomous vehicles or drones. The system receives requests from various parts of a city and has to manage the demands, assign tasks to agents, and plan a route for them. The system is subject to multiple legal and moral norms that must be respected to ensure its ethical and legal use. To force the system to respect these norms, we first propose a way to express each norm. We introduce two types of norms, absolute and relative. Absolute norms are hard constraints that must be respected by the system, and relative norms should be respected but can be compromised and traded off by other norms. Absolute norms refer to an obligation or prohibition, and relative norms refer to a precautionary rule to reduce harm or increase a benefit. The system has a compliance mechanism to check the compliance of system behavior and choose the best one considering moral criteria. The system may also face dilemmas, from a compliance checking point of view, which means that there are no compliant plans; however, in an emergency mode, we would like the system to select a plan by relaxing certain prohibitions.

## 7.2 Model Components

Here, we describe each component of the use case model and the knowledge used in each part. The system takes as input a set of demands and gives as output a combination which includes a resource assigned to each demand, a delivery task assigned to each agent, and a planned route for that agent. The components of the system and each of their input and interactions are represented in Figure 7.1

## 7.3 Planning Component

The planning module is made up of 3 components in a heuristic way. The first component is resource allocation, then demand assignment and at the end route planning. These components assign resources to the received demands and then assign delivery tasks to an agent. The route planning component plan a route for each agent in order to perform its delivery task. Each component is described in detail in this section.

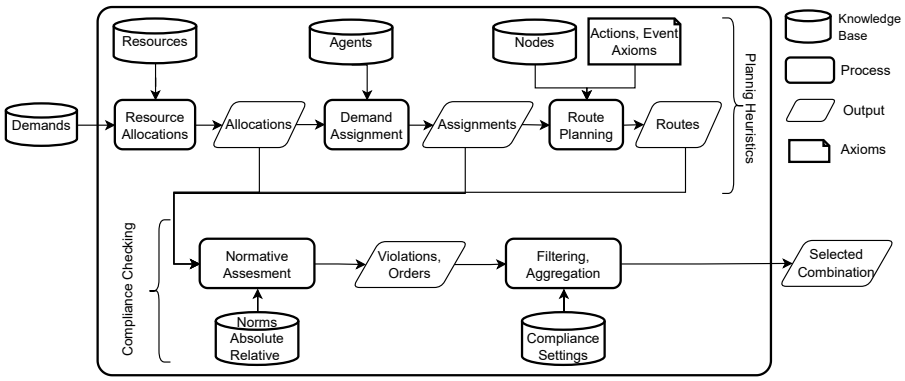



Figure 7.1: Modules in autonomous delivery system.

### 7.3.1 Resource Allocation

This component takes as input the demands and the resource database and assigns a resource to each demand. In this use case, each demand is made for a single resource, and the resource allocation module basically generates all possible ways the demands can be allocated. If there are sufficient resources, the assignment would be one-to-one; otherwise, if there is only a single resource and several demands have been made for that resource, this module generates all possible assignments.

```
insufficientResource(Resource):-
    demandedResource1(Demand, Resource),
    demandedResource1(Demand1, Resource),
    Demand !=Demand1.

sufficientResource(Resource):-
    not insufficientResource(Resource),
    resource(Resource).

allocatedResource(Resource, Demand):-
    demandedResource1(Demand, Resource),
    sufficientResource(Resource).

{allocatedResource(Resource,Demand)}:-
    demandedResource1(Demand, Resource),
    insufficientResource(Resource).

:-  allocatedResource(Resource,Demand1),
    allocatedResource(Resource,Demand2),
    insufficientResource(Resource),
    Demand1!=Demand2.

```

```
23 :-  not allocatedResource(Resource,_),
24     insufficientResource(Resource).
```

Listing 7.1: Resource allocation module

The list 7.1 is the resource allocation program; A resource is insufficient if there are more than two demands for it (line 1); otherwise, that resource would be considered sufficient (line 6). Each sufficient resource will be assigned to its corresponding demand. In case of insufficiency, we use a Choice rule to generate all possible allocations (line 14). There are two cases among all possible allocations that are not desired, and we use integrity rules to remove them. The first is the case where an inadequate resource is allocated to more than two demands (line 18), and the second is the case where an insufficient resource is not allocated to any demand (line 23). This allocation mechanism is useful when the system faces a scarcity of resources, and by generating all possible cases allows examining which one is more compliant.

As an example, consider 3 demands $d_1, d_2, d_3$ that are made, respectively, for resources $r_1, r_2, r_2$, as illustrated in Figure 7.2. In this case, $r_2$ is an insufficient resource. Thus, there will be two possible allocations in which $r_2$ is once allocated to $d_2$, and in the other it is allocated to $d_3$.

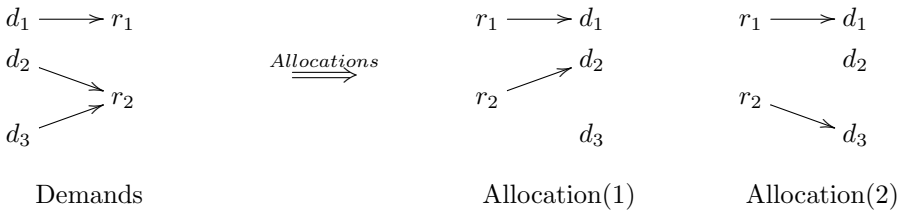



Figure 7.2: Resource allocation example

### 7.3.2 Demands Assignment to Agents

After the resource allocation step, the delivery task for demands with an allocated resource is assigned to the available agents. Demand assignment takes as input the allocations and information about the agent and generates all possible assignments. For simplicity, we assume that agents could perform only a single task, i.e. carry a single resource at a time. Furthermore, the agent's initial location is at the same place where the resources are stocked. Listing 7.2 shows the demand assignment program in ASP.

```
1 {assignedDemand(Demand,Agent)}:-
2   allocatedResource(_, Demand),
3   agent(Agent).
4
5 :- assignedDemand(Demand,Agent1),
6    assignedDemand(Demand,Agent2),
7    Agent1!=Agent2.
```

```
8
9  :- not assignedDemand(Demand,_),
10    allocatedResource(_, Demand).
```

Listing 7.2: demand assignments to agents

To obtain all possible assignments, we use a choice rule that assigns an agent to every demand to which a resource is assigned (line 1). Among the possible answers, it is not desired that a demand is assigned to several agents or that a single demand with an allocated resource is left unassigned. We use integrity rules to remove these cases, respectively, at lines 5 and 9 of the Listing 7.2. As an example, consider the allocation (1) in Figure 7.2, in which $d_1$ and $d_2$ are allocated a resource but $d_3$ is left unallocated. Suppose that we have 2 agents available $a_1, a_2$, then the assignments will be as in Figure 7.3

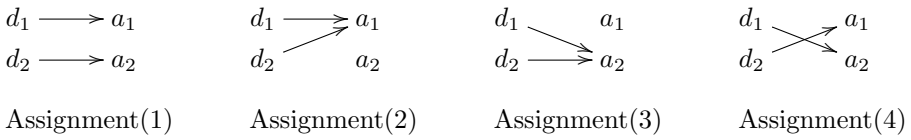



Figure 7.3: Demand assignment example

### 7.3.3 Route Planning

In this section, we describe the route planning process. After assigning the delivery tasks, a route is planned for each agent with a delivery task. This process takes as input knowledge about the nodes and their connections, the specified actions of the agents, and the axiomatic formalization for handling actions. For the latter, we have made use of event calculus, a well-known formalism for handling events and their effects, which is also used for planning. When there is more than one agent with an assigned task, since there are no assumed interactions between the agents, we plan the route for each one independently. It is also possible that more than two tasks are assigned to a single agent, in this case since we suppose that the capacity of each agent is one, i.e. only one resource can be delivered at a time, the agent is expected to deliver one of the resources, move back to the stockroom to take the other resource and deliver it.

Agents can perform several actions, e.g. board, move, and deliver. boarding means to pick up a resource and embark it in the vehicle; moving means going from one node to the other; and delivering means to disembark the demanded resource at a demanded location. We represent the variant state of the world by fluents which are used to check for the preconditions and apply the effects of a certain action after performing it. These preconditions and effects are noted in Table 7.1.

As stated in Table 7.1, to perform the action `board(Agent, Resource, X)`, the agent's repository must be empty `empty (Agent)` and must be in the same

| Action | Preconditions | Effects |
|---|---|---|
| `board(Agent, Resource, X)` | `resourceAt(Resource, X)`<br>`agentAt(Agent, X)`<br>`empty(Agent)` | `onBoard(Resource, Agent)`<br>`neg(resourceAt(Resource, X))`<br>`neg(empty(Agent))` |
| `move(Agent, X, Y)` | `agentAt(Agent, X)` | `agentAt(Agent,Y)`<br>`neg(agentAt(Agent, X))` |
| `deliver(Agent ,Resource, Demand,X)` | `agentAt(Agent, X)`<br>`demandAt(Demand, LocX)`<br>`onBoard(Resource, Agent)` | `delivered(Agent,Resource, Demand)`<br>`empty(Agent)`<br>`neg(onBoard(Resource, Agent))` |

Table 7.1: Actions' preconditions and effects

location as the resource represented by the fluents `resourceAt(Resource, X)` and `agentAt(Agent, X)`. when a board action is performed, as an effect `onBoard(Resource, Agent)` will hold afterward. The predicate `neg/1` is used to represent the negative effects, i.e. the effects that will no longer hold after performing an action. Here `neg(resourceAt(Resource, X))` and `neg(empty(Agent))` indicate that after boarding, a resource will not be at its previous location and the agent's repository will no longer be empty. The other actions and their preconditions could be described in the same way.

For every agent, the planner takes as input its assigned task and finds the shortest possible sequence of actions that satisfy the associated demand(s). The objective of the planning process is represented as a goal state in ASP and the Event Calculus formalism, as in the Listing 7.3. It states that for every assignment of demand to an agent that asks for a certain resource, that resource must be delivered by that agent before the maximum time step. The algorithm for finding the sequence of actions that satisfy the given goal in the shortest possible time step is shown in Algorithm 1.

```
:- not holds(delivered(Agent, Resource, Demand), maxtime),
    assignedDemand(Demand, Agent),
    demandedResource(Demand, Resource).
```

Listing 7.3: The goal state

For example, in assignment (1) in Figure 7.3, demand $d_1$ is assigned to $a_1$ and the resource $r_1$ is allocated to this demand according to allocation (1) in Figure 7.2. Thus, the associated task of the agent $a_1$ is to provide the resource $r_1$ and satisfy the demand $d_1$. We suppose that the system is operating on a simple map with 4 nodes that are connected to each other as in Figure 7.4, this knowledge is represented in the ASP as in Listing 7.4.

```
location(locA;locB;locC;locD).
connected(locA, locB).
```

**Algorithm 1** Planning algorithm

```
for every agent a do
    if a has a delivery task then
        t ← 1
        while Goal not satisfied do
            Generate a plan to achieve the goal state in t steps.
            t ← t + 1
        end while
    end if
end for
```

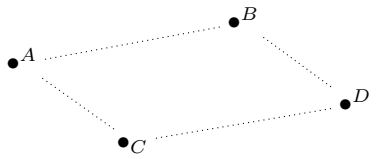



Figure 7.4: Map of the nodes

```
connected(locA, locC).
connected(locB, locD).
connected(locC, locD).

connected(X,Y):- connected(Y,X).
```

Listing 7.4: Nodes and connections

Agent $a_1$ and resource $r_1$ are initially in location $A$ and $r_1$ are asked in location $D$, that is, $d_1$ is initially at location $D$. These fluents are represented, respectively, by `initially ( agentAt(r1, locA))`, `initially (resourceAt(r1 , locA))` and `initially (demandAt (d1, locD))`. The planner would then generate two possible sequences of actions, each with a different route that satisfies the demand $d_1$. These plans are shown in Listing 7.5 using the predicate `perform/2` which shows the action that must be performed in each time step. A graphical representation of these plans is illustrated in the map graph in Figure 7.5(a).

```
%===Plan 1
performs(board(a1,r1,locA),0).
performs(move(a1,locA,locB),1).
performs(move(a1,locB,locD),2).
performs(deliver(a1,r1,d1,locD),3).

%===Plan 2
performs(board(a1,r1,locA),0).
performs(move(a1,locA,locC),1).
```

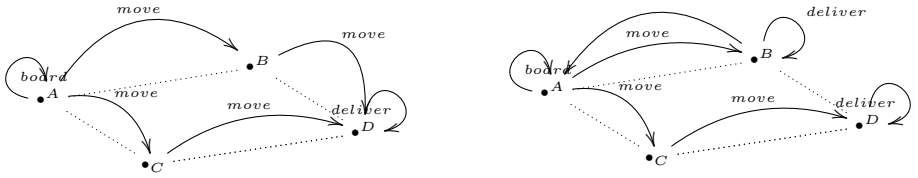
```
performs(move(a1,locC,locD),2).
performs(deliver(a1,r1,d1,locD),3).
```

Listing 7.5: Possible plans for $a_1$

When more than one delivery task is assigned to an agent, as in association (2) in Figure 7.3 the planner would also generate plans in which one resource is delivered, and then the agent returns to the warehouse to load the other resource and deliver it to its requested destination. This is because the agent repository has the capacity of one; it is possible to model agents with higher capacities; however, this is not necessary for our use case. Suppose that we have another demand $d_2$ at the point $B$ that has asked for the resource $r_2$, in this case a possible plan would be the sequence in the listing 7.6, which is illustrated in Figure 7.5(b). Note that this is one of the many possible plans, and other similar plans to satisfy the same goal are also possible by choosing different locations or order for delivery.

```
performs(board(a1,r2,locA),0).
performs(move(a1,locA,locB),1).
performs(deliver(a1,r2,d1,locB),3).
performs(move(a1,locB,locA),4).
performs(board(a1,r1,locA),5).
performs(move(a1,locA,locC),6).
performs(move(a1,locC,locD),7).
performs(deliver(a1,r1,d1,locD),8).
```

Listing 7.6: A possible plan for $a_1$




(a)Plannig with one task

(b)Plannig with two tasks

Figure 7.5: Example of plans

## 7.4 Compliance checking

The combination of allocations, assignments, and generated plans represents different alternatives in which the system can achieve its objective, that is, meet the demands received. We anticipate that the system adheres to any applicable norms that govern its functionality. Norms may have different implications

and/or nature depending on the problem of interest. In this model, we focus primarily on norms related to morality and law. Both moral and legal norms are imperative; in this context, moral norms correspond to regulating the system's behavior with regard to humans that may be affected by its actions. However, legal norms are attributed to the system when specific conditions are met according to certain regulations. There may be various normative behaviors that are expected in such a system. Sometimes, the system must avoid certain actions or behaviors because they violate an obligation or infringe on a prohibition. In certain cases, specific behaviors should be encouraged because they are more desirable, or the system should refrain from behaviors that are less favorable.

### 7.4.1 Normative Assessment

To enforce such normative behavior, the system must be equipped with a compliance mechanism that enables the specification and assessment of various norms. To do so, we consider two types of norm, namely *absolute* and *relative*. An absolute norm is a rule that refers to an obligation or a prohibition that can be violated or complied with by an alternative. Relative norms are rules that allow the comparison of two alternatives considering a desirable criterion. An alternative is considered invalid or unacceptable if it violates an absolute norm; in that sense, absolute norms can be considered as validity or hard constraints. However, a relative norm represents prescriptive or precautionary advice to minimize a harm or maximize a benefit, and it states that an alternative is better than or equal to another one according to the desired criteria. In other words, relative norms are used to compare alternatives rather than verify their validity; thus, they can be viewed as soft constraints. Absolute and relative norms are used to model, respectively, the deontological and utilitarianist nature of normative decision-making.

As mentioned previously, absolute norms are modeled as hard or validity constraints, so the system can filter out invalid alternatives that are considered illegal or immoral when at least an absolute norm is violated. Relative norms provide a means to compare alternatives on some scale that could be aggregated. Although a comparison on different scales may be possible depending on the problem, we utilized an ordinal scale and model the comparisons as preference relations. If an order by a relative norm prefers alternative $a$ to $b$, it is interpreted as "everything else being equal, this relative norm prefers alternative $a$ to $b$". Orders from several relative norms can be aggregated according to a given setting to determine the best alternative(s). Figure 7.6 illustrates the process of normative assessment using the two types of norms. All absolute norms must be met, and the satisfaction of relative norms should be maximized on the basis of a given setting.

As shown in Figure 7.6 after removing the invalid alternatives, the remaining ones would be ordered according to the relative norms and aggregated to obtain a single order that identifies the best alternative. we will discuss the procedure

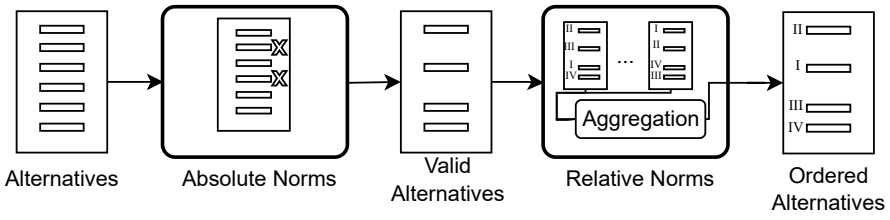



Figure 7.6: Normative assessment process

for the aggregation in the next section.

To show the functionality of the compliance process in our use case model, we make more assumptions in our model and extend the knowledge base. The system operates two types of agents, drones or autonomous vehicles, to deliver to each resource. A specific type of agent may not be allowed to pass through or fly over a certain location due to the municipal regulations of the city in which the system is operating. In addition, each type has a different speed and cost rate, which is used to calculate the duration and cost of a delivery, and that also requires information about the distance between nodes on the map. Demands can have different severity levels and time limits and correspond to subjects or patients with different age categories.

This knowledge will be used in the evaluation of norm compliance.

```
%Agent's type
type(a1, drone).
type(a2, autoBox).

%Prohibited zones
prohibited(drone, locB).
prohibited(autoBox, locC).

%Agent's speed
speed(drone, 2).
speed(autoBox, 1).

%Distance between nodes
distance(locA, locB, 8).
distance(locA, locC, 8).
distance(locB, locD, 6).
distance(locC, locD, 4).

% Demand's subject age category
ageCategory(d1, child).
ageCategory(d2, adult).

```

```
23 %Severity level of the Demands
24 severityLvl(d1, high).
25 severityLvl(d2, medium).
26
27 %Demand's time limit
28 timeLimit(d1, 20).
29 timeLimit(d2, 30).
```

Listing 7.7: Additional knowledge for compliance check

Since the choices made by the system affect humans and are subject to certain regulations, we expect its behavior to respect the moral and legal norms specified in the compliance check process. The norms specified in this use case model and their descriptions are listed in Table 7.2.

| Norm | Type | Definition | Description |
|---|---|---|---|
| `forbiddenZone` | Absolute | Agents must not pass through locations that are forbidden for them | This is a legally binding norm and applies to the choice of the delivery agent and the route planning step. |
| `missdDemand` | Absolute | The system must not miss any demand | This norm implies that the system must choose an alternative so that no demand is missed. This is not a legally binding norm, but a moral commitment of the system or the responsible company operating it |
| `severity` | Relative | Demands should be prioritized according to their severity order, that is, a more severe has higher priority | This is a pre-coutationary moral norm that tk to reduce the risk of any possible latency in delivery. According to this relative norm, an alternative in which more a severe order is delivered faster has prority over others |
| `agePriority` | Relative | Demands should be priortised according to a predefined order between age categories | This norm shows the preference of the system over age categories assuming that everything else is considered equal in comparison alternatives. The predefined order is, respectively, children, elderly and middle-aged |
| `costPriority` | Relative | Minimize the cost | According to this norm an alternative that uses less overall cost to meet the demands is considered to have more benefit and therefore it is preferable to a more costly alternative |

Table 7.2: The specified norms

The norms stated in Table 7.2 are used as an example to show the compliance check process in our model. Note that, for instance, the relative norm `agePriority` is not to say that the priority order, that is, children over the elderly and elderly over the middle-aged, is a morally acceptable order, but to show how such kinds of moral preferences are integrated in our model. Furthermore, since the system is running automatically and is expected to minimize the cost, it is arguable that `costPriority` should not be considered a moral norm. In such a case, we can hold two points of view; we either view it as moral criteria that is beneficiary for some involved party, or view it as non-moral criteria that need to be optimized along with other criteria which may be moral. We hold the first view in this use-case model, since it suggests a unified way to handle relative

norms.

The absolute and relative norms, in Table 7.2, are represented in ASP using the predicates `absolute/1` and `relative/1` as shown in the listing 7.8. In case of `forbiddenZone`, the prohibition expands to every agent with an assigned demand, which is indicated by the rule at line 1.

```
1 %Absolute Norms
2 absolute(forbiddenZone(Agent)):- assignedDemand(_, Agent).
3 absolute(missingDemand).
4
5 %Relative Norms
6 relative(severity).
7 relative(agePriority).
8 relative(costPriority).
```

Listing 7.8: Absolute and relative norms

After some auxiliary predicates and rules are defined, violations of absolute norms and associated orders of relative norms are obtained. These violations and orders are shown in the Listing 7.9. In the case of the absolute norms, instead of checking the violation, we count the number violation because of a mechanism called relation, which will be described later. The relative norms `severity` and `agePriority` order the alternatives lexicographically based on a predefined order of severity level and age categories.

```
%Absolute Norms
nbViolate(forbiddenZone(Demand), Plan, N):- nbBreachedLoc(Plan, Demand, N
    ↪ ).

nbViolate(missingDemand, Plan, N):- nbMissedDemands(Plan, N).

%===== Relative Norms

pref(severity, Plan1, Plan2):-
    lexicoPref(severity, Plan1, Plan2).

pref(agePriority, Plan1, Plan2):-
    lexicoPref(agePriority, Plan1, Plan2).

pref(costPriority, PlanA, PlanB):-
    overallCost(PlanA, C1),
    overallCost(PlanB, C2),
    C1<=C2.
```

Listing 7.9: Violations and orders

### 7.4.2 Aggregation

Violations and orders are used to find the most aligned valid alternative. An alternative must comply with all the specified absolute norms to be considered

compliant. After this step, the orders of the compliant alternatives are aggregated in order to choose the best. The aggregation process for the relative norms is based on the voting system as explained in section 5.1.4.

When orders are aggregated using a given compliance setting, the violating alternatives are first excluded to keep only the compliant ones. This is represented by the predicate `compliant / 1` on line 1 of the Listing 7.10. After filtering out noncompliant alternatives, we compare the alternatives and choose the one that is preferred to every other alternative in the final aggregated ordering. This is indicated by the rule on line 5 and is represented by the predicate `winner/1`.

```
1 nonCompliant(A):- criteria(C), absolute(C), nbViolate(C,A,N), alt(A), N
      ↪ >0.
2 compliant(A):- not nonCompliant(A), alt(A).
3
4 loser(A):- prefRelative(B,A), compliant(A), compliant(B), A!=B.
5 winner(A):- not loser(A), compliant(A).
```

Listing 7.10: The best compliant alternative

The aggregation method allows you to specify various compliance settings according to the conditions under which the system is operating. In our model, we have introduced several absolute and relative norms. Each way of adjusting the equivalency and superiority among the relative norms leads to different settings for compliance. The flexibility of the input setting allows us to adapt the functionality of the system to our desired behavior. When the system deals with high-risk demands, we may prioritize harm preventive norms, e.g. *severity* and *age Priority* over *cost* that is a beneficiary norm. And when the system deals with low-risk requests, *cost* may be prioritized over *severity* and *age Priority*. We use a graphical representation to illustrate the input settings. Some examples are shown in Figure 7.7. In this representation, the absolute norms are separated with a line, the equivalent relative norms that form a class are placed in a single column, and each class of relative norms is separated with a dashed line. The classes on the left are considered superior to those on the right.

As shown in Figure 7.7, in the settings(1) each relative norm forms a single class in which, *severity* has superiority over *age Priority* and that also has a superiority over *cost*. In the Settings(2) *severity* and *age Priority* are in the same class, which means that their corresponding order will be aggregated by a voting rule, and this collective vote has superiority over *cost*. In the setting(3) all the relative norms are considered equivalent and so are in the same class. And in the settings(4) *cost* is prioritized over *severity* and *age Priority*.

### 7.4.3 Norm Relaxation

There are situations in which no alternative is compliant, i.e. all of them violate at least one absolute norm. Although such situations may rarely occur in a well-deployed system, we expect it to have a mechanism to manage these situations.

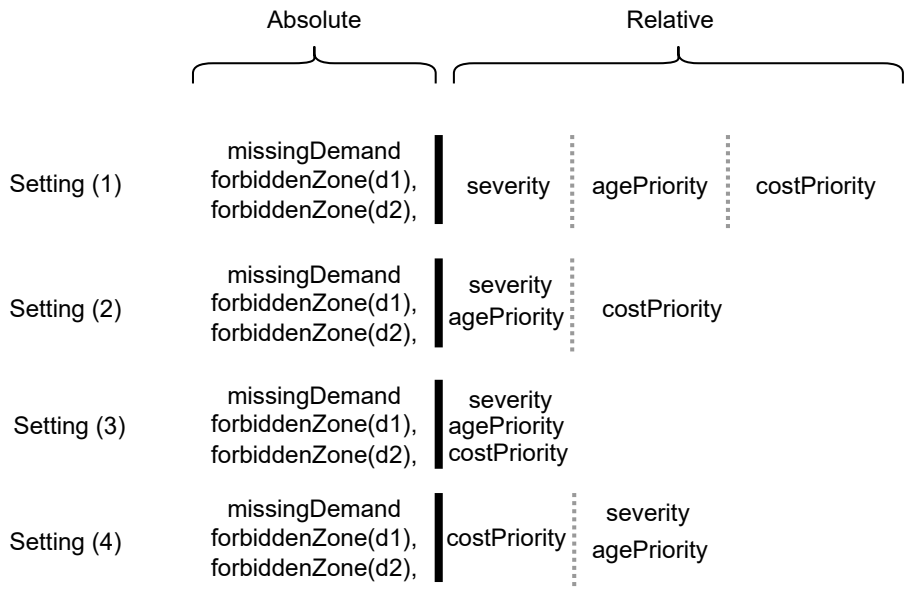



Figure 7.7: Various settings for compliance check

For example, when there is a scarcity of resources, the system cannot avoid missing some demand. In this case, the system is actually faced with a dilemma between demands that can be fulfilled and those that must be skipped. Another example is when certain demands are very urgent, so that the system's capacity is not enough, i.e. the agents cannot deliver them on time. In this situation, let us suppose that there is a certain prohibited route that is taken and that the system can meet these demands. Therefore, the system is faced with the dilemma of violating the prohibition of the forbidden zone or the missing demand obligation. One way to solve these dilemmas is to relax certain absolute norms, i.e. treat an absolute norm as a relative norm. In this case, instead of checking the violation of an absolute norm, we assign an order based on the number of violations. The code snippet in the listing 7.11 shows how to integrate the relaxation mechanism in the compliance check process, we use the `relaxed/1` predicate to indicate the relaxation of an absolute norm.

```
pref(C, PlanA, PlanB):-
  absolute1(C),
  nbViolate(C, PlanA, N1),
  nbViolate(C, PlanB, N2),
  N1<=N2.

absolute(Criteria):- absolute1(Criteria), not relaxed(Criteria).
relative(Criteria):- relative1(Criteria).
```

```
9 relative(Criteria):- relaxed(Criteria).
```

Listing 7.11: Knowledge representation

In our use case, when there is a scarcity of resources, by relaxing *missig demand* obligation we can find a compromised plan. In other words, it is not possible to fulfill all demands, but we want to fulfill as much as possible or miss demands to a lesser extent. This is shown in setting (1) in Figure 7.8. Another example is that the system's capacity or the agents' capabilities do not allow the system to fulfill demands by choosing legal routes; in this, if the demands are considered emergency, we may want to relax the *forbidden Zone* obligation. This is shown in setting (2) of Figure 7.8

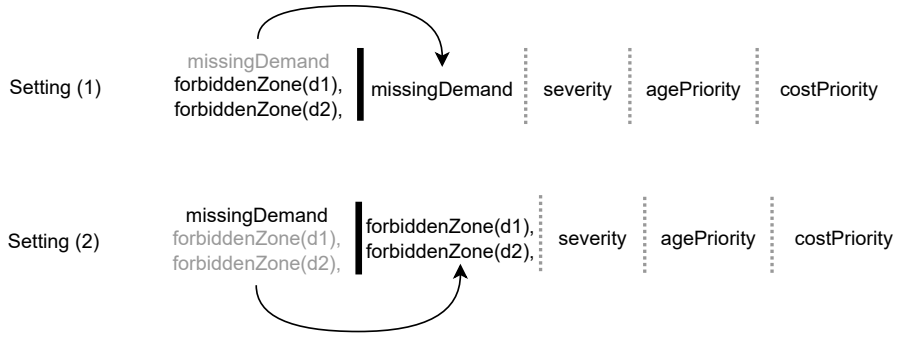



Figure 7.8: Relaxed settings

## 7.5 Discussions

One of the objectives of this contribution was to create a new architecture that better captures the interaction between law and ethics. Moreover, we modeled a new use case that sufficiently demonstrated the features of the proposed architecture. This model shows that merging legal and ethical components in a single component and categorizing them based on relativism better endorse the dynamics between law and ethics.

The proposed architecture is a modeling of the real-world system to be able to examine its behavior in different situations, particularly dilemmas. The components in the planning part can be replaced by other more sophisticated modules, e.g. using another algorithm for resource allocations and a machine learning algorithm for path finding.

This contribution is at its early stage was an attempt to investigate other architectures. Several perspectives remain to be explored; we introduced a relaxation mechanism, but it was only tested manually. Developing a process to automatize the relaxations should be considered in future work. In addition, we need to test the architecture in other use cases to demonstrate its effectiveness.

# Part III

# Discussions

# Chapter 8

# Conclusions

## 8.1 Summary of Findings

from a more technical point of view, the objective of this thesis was to model GDPR compliance and ethical alignment mechanism for an agent with actions that involve the processing of personal data. We studied this problem throughout the presented contributions. We showed that Event Calculus or HTN planning is suitable for model data processing operations and planning. The GDPR requirements were represented using the SPECIAL policy language. The policy language provides a unified representation of GDPR obligations and data subject's consent. We translated the requirements written in the OWL language, into ASP and Prolog in order to integrate it with the planning formalism and use it for the compliance checking of the plans.

We explored architectures to combine the components of legal compliance and ethical evaluation with planning. Two possible ways to integrate these components were proposed. In Chapter 6 we integrated legal norms as hard norms and ethical norms as soft norms. For this purpose, we first developed an ethical evaluation framework based on preference orders. This model uses multiple criteria associated with moral values. These criteria order alternatives and aggregate them using voting rules. An advantage of this model is that it can be adopted in other contexts by changing the ethical specification. In addition, using orders and intuitive voting rules for aggregation increases the expressive power of the model.

This model is integrated with the HTN planner along with the compliance check component. In the corresponding architecture, legal compliance has a priority over ethical compliance because ethical norms are only used to order plans rather than to decide whether they are morally right or wrong. Although this method is useful in most cases, it does not capture ethical hard norms. We then proposed a new architecture where the legal and ethical norms are combined in a single component and integrated with the EC (cf. Chapter 7). This architecture was tested in a use case of health care delivery system that is different from the previous scenarios. This scenario shows that the models used in this research are adaptable to new scenarios.

On the more philosophical side the difficulty in assigning consistent utilities incommunsurability of values pose a major challenge especially in computational ethics. We showed that using preferences and ordinal aggregation methods e.g. voting systems can be useful to develop a rational decision making process. We did not explore what kind of properties should an aggregation process have in order to be considered rational. However, we addressed some obvious types of irrationality, in particular in choosing the suitable voting system for aggregation. One reason why we chose Copeland's rule in our contributions is because of its Condorcet winner property. This property is essential to avoid cyclic preferences, which is a paradoxical.

In addition, representing concepts such as moral values is a major challenge for computational ethics. Values are often used for moral reasoning. It is especially used in the way humans talk about ethics. using moral values in the reasoning process is more understandable by humans. We proposed an approach

that can be used to represent moral values to some extent. In order to capture the complex nature of the values, we propose a hierarchical structure in which values are decomposed into simpler criteria. Another advantage of this representation is that the reasoning process is more expressive. In other words, it is clear what values are composed of in this model, in order to avoid ambiguity. A major question was what are values composed of so that we can decompose them? We showed that one effective ways to respond to this question is by using AI ethics guidelines. There has been a trend of such guidelines that discuss a variety of ethical issues related to AI systems. we adopted these guidelines in our specific domain of study, which is personal data processing. We showed that the criteria mentioned in these guidelines can be represented and used in the reasoning process of AI agent.

## 8.2 Future Works

Although we tried to respond to the main research questions in this thesis, there are many interesting directions that remain to be explored in the future. one of these areas is the analysis of the efficiency and computational complexity of planning with legal and ethical evaluations. During the experiments, we realized that increasing the number of represented regulations for compliance checking has a direct effect on the plan evaluations. The same result occurs when the number of criteria increases in the list. The voting rule used to aggregate orders is significantly important in the complexity of ethical evaluations. Condorcet extensions are computationally more expensive because they need to compare every possible pair of alternatives. The issue of computational complexity is particularly important when our proposed models are applied to larger-scale problems or used in real-world cases; however, it remains as a future work. In addition, integration of planning with more sophisticated legal ontologies such as LegalRuleML 1.3.1 allows covering more regulations and enables legal reasoning with the planning agent. However, this is left for future research.

Moreover, voting rules and their application in implementing moral theories is not sufficiently explored in this thesis. Because of the expressiveness and intuitive process of the voting systems, they are a strong candidate to model comparative approaches for ethical decision making. drawing parallels between the properties of voting systems and their implications in moral decision making could enrich this thesis to a large extent. However, this is also remains a future work.

Another extension of the contributions made in this thesis could be dealing with incomplete information. The frameworks proposed in this thesis deal only with complete information. Therefore, we did not consider cases where part of the information used for legal compliance is missing or some preference orders in the case of ethical evaluation are incomplete. This is particularly important when it comes to real-time legal and ethical evaluations, the agent may not have access to some necessary information in certain time. a robust legal and ethical framework should be equipped with mechanisms to deal with such cases.

The next issue is uncertainty. Although we did some studies related to ethical decision making under uncertainty [87, 147, 127, 55], this remained unexplored in this thesis. Uncertainty in our context may be caused by the environment in which the agent is operating or lack of certainty if a legal or ethical norm applies in a specific case. The latter is called normative uncertainty. The issue of uncertainty has been partially studied during this thesis; however, further developments remain to be explored. This extension is particularly useful when an ML algorithm retrieves certain information used for compliance checking or ethical algorithms. Such algorithms have a large prediction capacity, but usually have a degree of uncertainty about the output. Dealing with this kind of uncertainty increases the adaptability of legal and ethical evaluations to more specific or unobserved cases.

One of the avenues remained to be explored on the more philosophical side is to model different methods discussed in value theory for rational decision-making. These include, *practical wisdom*, *super scales*, *basic preferences* etc. [128]. For example, a super scale discusses the possibility of rationally comparing plural values. The super scale has been discussed by philosophers in different terms. Stocker [175] suggests that a "higher-level synthesizing category" can describe how comparisons are made. Chang [44] advocates that there is a "covering value" that makes the comparison of plural values possible. A covering value is a more comprehensive value that has plural values as parts. These theories seem to align well with our proposed pluralistic utility model. For example, the specification of the preferences among values is a kind of super scale on which we compare different values. However, we did not explore the possible links in detail. A more in-depth investigation of these theories and their computational modeling remain as a future work.

# Bibliography


[1] Dirk Abels et al. "Train scheduling with hybrid ASP". In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2019, pp. 3–17.

[2] Fran Ackermann and Colin Eden. "The role of group decision support systems: negotiating safe energy". In: *Handbook of group decision and negotiation* (2010), pp. 285–299.

[3] Sushant Agarwal et al. "Legislative compliance assessment: framework, model and GDPR instantiation". In: *Annual Privacy Forum*. Springer. 2018, pp. 131–149.

[4] HLEG AI. "High-level expert group on artificial intelligence". In: *Ethics guidelines for trustworthy AI* 6 (2019).

[5] *Algorithmic Accountability Act of 2019*. S.1108, 116th Cong. (2019). 2019. URL: `https://www.congress.gov/bill/116th-congress/senate-bill/1108`.

[6] Michael Anderson and Susan Leigh Anderson. "Machine ethics: Creating an ethical intelligent agent". In: *AI magazine* 28.4 (2007), pp. 15–15.

[7] Michael Anderson, Susan Leigh Anderson, and Chris Armen. "MedEthEx: a prototype medical ethics advisor". In: *Proceedings of the national conference on artificial intelligence*. Vol. 21. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2006, p. 1759.

[8] Michael Anderson, Susan Leigh Anderson, and Chris Armen. "Towards machine ethics". In: *AAAI-04 workshop on agent organizations: theory and practice, San Jose, CA*. 2004.

[9] Michael Anderson, Susan Leigh Anderson, and Chris Armen. "MedEthEx: Toward a Medical Ethics Advisor." In: *AAAI Fall Symposium: Caring Machines*. 2005, pp. 9–16.

[10] Grigoris Antoniou and Frank van Harmelen. "Web ontology language: Owl". In: *Handbook on ontologies* (2009), pp. 91–110.

[11] Krzysztof R Apt and Maarten H Van Emden. "Contributions to the theory of logic programming". In: *Journal of the ACM (JACM)* 29.3 (1982), pp. 841–862.

[12] Martin Aruldoss, T Miranda Lakshmi, and V Prasanna Venkatesan. "A survey on multi criteria decision making methods and its applications". In: *American Journal of Information Systems* 1.1 (2013), pp. 31–43.

[13] Tara Athan et al. "LegalRuleML: Design principles and foundations". In: *Reasoning Web. Web Logic Rules: 11th International Summer School 2015, Berlin, Germany, July 31-August 4, 2015, Tutorial Lectures. 11* (2015), pp. 151–188.

[14] Tara Athan et al. "Oasis legalruleml". In: *proceedings of the fourteenth international conference on artificial intelligence and law*. 2013, pp. 3–12.

[15] Anna Bacciarelli. "The Toronto Declaration: Protecting the right to equality and non-discrimination in machine learning systems". In: (2023).

[16] Kristine Bærøe, Ainar Miyata-Sturm, and Edmund Henden. "How to achieve trustworthy artificial intelligence for health". In: *Bulletin of the World Health Organization* 98.4 (2020), p. 257.

[17] SSJA Baier and Sheila A McIlraith. "HTN planning with preferences". In: *21st Int. Joint Conf. on Artificial Intelligence*. 2009, pp. 1790–1797.

[18] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

[19] Cesare Bartolini et al. "Towards legal compliance by correlating standards and laws with a semi-automated methodology". In: *BNAIC 2016: Artificial Intelligence: 28th Benelux Conference on Artificial Intelligence, Amsterdam, The Netherlands, November 10-11, 2016, Revised Selected Papers 28*. Springer. 2017, pp. 47–62.

[20] Michael Beil et al. "Ethical considerations about artificial intelligence for prognostication in intensive care". In: *Intensive Care Medicine Experimental* 7 (2019), pp. 1–13.

[21] Thomas Beke. *Litigation communication*. Springer, 2014.

[22] Jeremy Bentham. "An introduction to the principles of morals and legislation". In: *History of Economic Thought Books* (1781).

[23] Fiona Berreby, Gauvain Bourgne, and Jean-Gabriel Ganascia. "A declarative modular framework for representing and applying ethical principles". In: *16th Conference on Autonomous Agents and MultiAgent Systems*. 2017.

[24] Avrim L Blum and Merrick L Furst. "Fast planning through planning graph analysis". In: *Artificial intelligence* 90.1-2 (1997), pp. 281–300.

[25] Harold Boley, Adrian Paschke, and Omair Shafiq. "RuleML 1.0: the overarching specification of web rules". In: *International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer. 2010, pp. 162–178.

[26] Michael J Bommarito II, Daniel Martin Katz, and Ron Dolin. "Motivation and Rationale for this Book". In: ().

[27] Piero A Bonatti et al. “Machine understandable policies and GDPR compliance checking”. In: *KI-Künstliche Intelligenz* 34 (2020), pp. 303–315.

[28] Steven J Brams. *Approval voting*. Springer, 2004.

[29] Felix Brandt, Vincent Conitzer, and Ulle Endriss. “Computational social choice”. In: *Multiagent systems* 2 (2012), pp. 213–284.

[30] Felix Brandt et al. “Introduction to computational social choice”. In: *Handbook of Computational Social Choice* (2016), pp. 1–29.

[31] Richard B Brandt. *A Theory of the Good and the Right*. Clarendon Press, 1984.

[32] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001.

[33] Rachael A Briggs. “Normative theories of rational choice: Expected utility”. In: (2014).

[34] Frank M Brown. *The frame problem in artificial intelligence: Proceedings of the 1987 workshop*. Morgan Kaufmann, 2014.

[35] Banu Buruk, Perihan Elif Ekmekci, and Berna Arda. “A critical perspective on guidelines for responsible and trustworthy artificial intelligence”. In: *Medicine, Health Care and Philosophy* 23.3 (2020), pp. 387–399.

[36] Krister Bykvist. “Moral uncertainty”. In: *Philosophy Compass* 12.3 (2017), e12408.

[37] *California Privacy Rights Act*. California Civil Code, 1798.100 et seq. 2020. URL: `https://thecpra.org/`.

[38] Marie Jean Antoine Nicolas de Caritat and Marquis De Condorcet. *Essai sur l’application de l’analyse à la probabilité des décisions rendues à la pluralité des voix*. 1785.

[39] Micah Carroll et al. “Characterizing manipulation from AI systems”. In: *Proceedings of the 3rd ACM Conference on Equity and Access in Algorithms, Mechanisms, and Optimization*. 2023, pp. 1–13.

[40] Center for Democracy & Technology. *Digital Decisions*. Online. Accessed on 21st February 2019. URL: `https://cdt.org/issue/privacy-data/digital-decisions/`.

[41] Iliano Cervesato and Angelo Montanari. “A general modal framework for the event calculus and its skeptical and credulous variants”. In: *The Journal of Logic Programming* 38.2 (1999), pp. 111–164.

[42] Anirban Chakraborty et al. “A survey on adversarial attacks and defences”. In: *CAAI Transactions on Intelligence Technology* 6.1 (2021), pp. 25–45.

[43] Ruth Chang. “Incommensurability, incomparability, and practical reason”. In: (1997).

[44] Ruth Chang. “Value incomparability and incommensurability”. In: *The Oxford handbook of value theory* (2015), pp. 205–224.

[45] Raja Chatila, Kay Firth-Butterfield, and John C Havens. "Ethically aligned design: A vision for prioritizing human well-being with autonomous and intelligent systems version 2". In: *University of southern California Los Angeles* (2018).

[46] Luca Chittaro, Angelo Montanari, et al. "A modal calculus of partially ordered events in a logic programming framework". In: *Proceedings of the Twelfth International Conference on Logic Programming—ICLP'95*. 1995, pp. 299–313.

[47] William F Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[48] Nicolas Cointe, Grégory Bonnet, and Olivier Boissier. "Ethical Judgment of Agents' Behaviors in Multi-Agent Systems." In: *AAMAS*. 2016, pp. 1106–1114.

[49] Alain Colmerauer. "An introduction to Prolog III". In: *Communications of the ACM* 33.7 (1990), pp. 69–90.

[50] Commission Nationale de l'Informatique et des Libertés (CNIL), European Data Protection Supervisor (EDPS) & Garante per la protezione dei dati personali. *Declaration on Ethics and Data Protection in Artificial Intelligence*. Online. Accessed on 9th April 2022. 2018. URL: `https://globalprivacyassembly.org/wp-content/uploads/2018/10/20180922_ICDPPC-40th_AI-Declaration_ADOPTED.pdf`.

[51] Geoff Currie, K Elizabeth Hawk, and Eric M Rohren. *Ethical principles for the application of artificial intelligence (AI) in nuclear medicine*. 2020.

[52] Marina De Vos et al. "ODRL policy modelling and compliance checking". In: *Rules and Reasoning: Third International Joint Conference, RuleML+ RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings 3*. Springer. 2019, pp. 36–51.

[53] Stefan Decker, Prasenjit Mitra, and Sergey Melnik. "Framework for the semantic Web: an RDF tutorial". In: *IEEE Internet Computing* 4.6 (2000), pp. 68–73.

[54] Marc Denecker, Lode Missiaen, and Maurice Bruynooghe. "Temporal reasoning with abductive event calculus". In: *Proceedings of the 10th European Conference on Artificial Intelligence, ECAI92*. John Wiley and Sons; Chichester. 1992, pp. 384–388.

[55] Benjamin Djulbegovic. "Ethics of uncertainty". In: *Patient Education and Counseling* 104.11 (2021), pp. 2628–2634.

[56] Mauro Dragoni et al. "Combining NLP approaches for rule extraction from legal documents". In: *1st Workshop on MIning and REasoning with Legal texts (MIREL 2016)*. 2016.

[57] Ulle Endriss. "Judgment aggregation". In: (2016).

[58] *Ethics Guidelines for Trustworthy AI*. European Commission. Apr. 8, 2019. URL: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai (visited on 05/11/2022).

[59] European Commission. *Digital Services Act: EU rules to ensure safe and accountable online environment*. Press Release. Press Corner. Apr. 2022. URL: https://ec.europa.eu/commission/presscorner/detail/en/IP_22_2321.

[60] European Commission. *Proposal for a Regulation of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain Union legislative acts*. Tech. rep. European Commission, 2021. URL: https://ec.europa.eu/commission/presscorner/detail/en/ip_21_1682.

[61] European Commission. *Proposal for a Regulation of the European Parliament and of the Council on harmonised rules on fair access to and use of data (Data Act)*. European Commission Document. COM(2022) 68 final. Feb. 2022. URL: https://digital-strategy.ec.europa.eu/en/library/data-act-legislative-proposal-harmonised-rules-fair-access-and-use-data.

[62] European Group on Ethics in Science and New Technologies. *Statement on Artificial Intelligence, Robotics and 'Autonomous' Systems*. Online. Accessed on 9th April 2022. 2018. URL: https://op.europa.eu/en/publication-detail/-/publication/dfebe62e-4ce9-11e8-be1d-01aa75ed71a1.

[63] European Parliament, Council of the European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. 2016. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[64] Richard E Fikes and Nils J Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4 (1971), pp. 189–208.

[65] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 2015, pp. 1322–1333.

[66] French National Ethical Consultative Committee for Life Sciences and Health. *Digital Technology and Healthcare: Which Ethical Issues for Which Regulations?* Paris, France, 2018.

[67] Clémence Frioux et al. "Hybrid metabolic network completion". In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 83–108.

[68] Jean Gallier. "SLD-Resolution and Logic Programming". In: *Logic for Computer Science: Foundations of Automatic Theorem Proving* (2003), p. 35.

[69] Jean-Gabriel Ganascia. "Modelling ethical rules of lying with Answer Set Programming". In: *Ethics and information technology* 9 (2007), pp. 39–47.

[70] Ismael Garrido-Muñoz et al. "A survey on bias in deep NLP". In: *Applied Sciences* 11.7 (2021), p. 3184.

[71] Martin Gebser et al. *Answer set solving in practice*. Springer Nature, 2022.

[72] Martin Gebser et al. "clasp: A conflict-driven answer set solver". In: *Logic Programming and Nonmonotonic Reasoning: 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007. Proceedings 9*. Springer. 2007, pp. 260–265.

[73] Martin Gebser et al. "Clingo= ASP+ control: Preliminary report". In: *arXiv preprint arXiv:1405.3694* (2014).

[74] Michael Gelfond. "Answer sets". In: *Foundations of Artificial Intelligence* 3 (2008), pp. 285–316.

[75] Michael Gelfond and Yulia Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.

[76] Michael Gelfond and Vladimir Lifschitz. "The stable model semantics for logic programming." In: *ICLP/SLP*. Vol. 88. Cambridge, MA. 1988, pp. 1070–1080.

[77] Michael Gelfond, Halina Przymusinska, and Teodor Przymusinski. "On the relationship between circumscription and negation as failure". In: *Artificial Intelligence* 38.1 (1989), pp. 75–94.

[78] Michael R Genesereth and Nils J Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 2012.

[79] J Gips. "Creating ethical robots: A grand challenge". In: *AAAI fall 2005 symposium on machine ethics*. 2005.

[80] Google AI. *Our Principles*. Online. 2018. URL: `https://ai.google/principles/`.

[81] Guido Governatori and Sidney Shek. "Rule Based Business Process Compliance." In: *RuleML (2)*. Citeseer. 2012.

[82] Umberto Grandi et al. "Logic-Based Ethical Planning". In: *AIxIA 2022–Advances in Artificial Intelligence: XXIst International Conference of the Italian Association for Artificial Intelligence, AIxIA 2022, Udine, Italy, November 28–December 2, 2022, Proceedings*. Springer International Publishing Cham. 2023, pp. 198–211.

[83] Salvatore Greco, Jose Figueira, and Matthias Ehrgott. *Multiple criteria decision analysis*. Vol. 37. Springer, 2016.

[84] James Griffin. *Well-being: Its meaning, measurement and moral importance*. Clarendon press, 1986.

[85] SFR-IA Group, French Radiology Community, et al. "Artificial intelligence and medical imaging 2018: French Radiology Community white paper". In: *Diagnostic and Interventional Imaging* 99.11 (2018), pp. 727–742.

[86] Tom Gruber. *What is an Ontology*. 1993.

[87] S Hansson. *The ethics of risk: Ethical analysis in an uncertain world*. Springer, 2013.

[88] Richard Mervyn Hare et al. *Moral thinking: Its levels, method, and point*. Oxford: Clarendon Press; New York: Oxford University Press, 1981.

[89] John C Harsanyi. "Morality and the theory of rational behavior". In: *Social research* (1977), pp. 623–656.

[90] Mustafa Hashmi, Guido Governatori, and Moe Thandar Wynn. "Business process data compliance". In: *Rules on the Web: Research and Applications: 6th International Symposium, RuleML 2012, Montpellier, France, August 27-29, 2012. Proceedings 6*. Springer. 2012, pp. 32–46.

[91] Hisashi Hayashi and Ken Satoh. "Towards Legally and Ethically Correct Online HTN Planning for Data Transfer." In: *NMR*. 2022, pp. 4–15.

[92] Hisashi Hayashi et al. "Dynagent: An incremental forward-chaining HTN planning agent in dynamic domains". In: *Declarative Agent Languages and Technologies III: Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers 3*. Springer. 2006, pp. 171–187.

[93] Hisashi Hayashi et al. "Multi-agent Online Planning Architecture for Real-time Compliance". In: *RuleML Challenge* (2023).

[94] Hisashi Hayashi et al. "Toward smooth integration of an online HTN planning agent with legal and ethical checkers". In: *VECOMP Workshop* (2024).

[95] High-Level Expert Group on Artificial Intelligence. *Ethical Guidelines for Trustworthy AI by the European Commission*. Tech. rep. European Commission, 2019. URL: `https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai`.

[96] Kenneth Einar Himma. "Artificial agency, consciousness, and the criteria for moral agency: What properties must an artificial agent have to be a moral agent?" In: *Ethics and Information Technology* 11 (2009), pp. 19–29.

[97] *Ethics and data protection*. European Commission, Sept. 14, 2018. (Visited on 09/14/2018).

[98] Charles Hu and Yen-Hung Frank Hu. "Data poisoning on deep learning models". In: *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2020, pp. 628–632.

[99] Rosalind Hursthouse. "On virtue ethics". In: *Applied Ethics*. Routledge, 2017, pp. 29–35.

[100] IBM. *Everyday Ethics for Artificial Intelligence: A Practical Guide for Designers & Developers*. Online. Accessed on 20th February 2022. 2018. URL: `https://www.ibm.com/watson/assets/duo/pdf/everydayethics.pdf`.

[101] Internet Society. *Artificial Intelligence & Machine Learning: Policy Paper*. Online. Accessed on 21st February 2019. 2017. URL: `https://www.internetsociety.org/resources/doc/2017/artificial-intelligence-and-machine-learning-policy-paper/`.

[102] Anna Jobin, Marcello Ienca, and Effy Vayena. "The global landscape of AI ethics guidelines". In: *Nature machine intelligence* 1.9 (2019), pp. 389–399.

[103] Antonios Kakas and Rob Miller. "A simple declarative language for describing narratives with actions". In: *The Journal of Logic Programming* 31.1-3 (1997), pp. 157–200.

[104] Roland Kaminski, Torsten Schaub, and Philipp Wanko. "A tutorial on hybrid answer set solving with clingo". In: *Reasoning Web. Semantic Interoperability on the Web: 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures 13* (2017), pp. 167–203.

[105] Immanuel Kant. *Groundwork of the metaphysic of morals (HJ Paton, Trans.)* 1964.

[106] Daniel Martin Katz, Ron Dolin, and Michael J Bommarito. *Legal informatics*. Cambridge University Press, 2021.

[107] John G Kemeny and LJ Snell. "Preference ranking: an axiomatic approach". In: *Mathematical models in the social sciences* (1962), pp. 9–23.

[108] D Marc Kilgour and Keith W Hipel. "Conflict analysis methods: The graph model for conflict resolution". In: *Handbook of group decision and negotiation* (2010), pp. 203–222.

[109] Tae-Won Kim, Joohyung Lee, and Ravi Palla. "Circumscriptive event calculus as answer set programming". In: *Twenty-First International Joint Conference on Artificial Intelligence*. 2009.

[110] Olli Koski, Kai Husso, et al. "Work in the age of artificial intelligence: Four perspectives on the economy, employment, skills and ethics". In: (2018).

[111] Robert Kowalski. "Database updates in the event calculus". In: *The Journal of Logic Programming* 12.1-2 (1992), pp. 121–146.

[112] Robert Kowalski. "Predicate logic as programming language". In: *IFIP congress*. Vol. 74. 1974, pp. 569–544.

[113] Robert Kowalski and Marek Sergot. "A logic-based calculus of events". In: *New generation computing* 4 (1986), pp. 67–95.

[114] Adam Lally and Paul Fodor. "Natural language processing with prolog in the ibm watson system". In: *The Association for Logic Programming (ALP) Newsletter* 9 (2011), p. 2011.

[115] Ho-Pun Lam and Mustafa Hashmi. "Enabling reasoning with LegalRuleML". In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 1–26.

[116] Joohyung Lee and Ravi Palla. "Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming". In: *Journal of Artificial Intelligence Research* 43 (2012), pp. 571–620.

[117] Fan Li, Nick Ruijs, and Yuan Lu. "Ethics & AI: A systematic review on ethical concerns and related strategies for designing with AI in healthcare". In: *Ai* 4.1 (2022), pp. 28–53.

[118] Yunqi Li et al. "Fairness in recommendation: A survey". In: *arXiv preprint arXiv:2205.13619* (2022).

[119] Vladimir Lifschitz. *Answer set programming*. Vol. 3. Springer Heidelberg, 2019.

[120] Raynaldio Limarga et al. "Non-monotonic reasoning for machine ethics with situation calculus". In: *AI 2020: Advances in Artificial Intelligence: 33rd Australasian Joint Conference, AI 2020, Canberra, ACT, Australia, November 29–30, 2020, Proceedings 33*. Springer International Publishing. 2020, pp. 203–215.

[121] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable ai: A review of machine learning interpretability methods". In: *Entropy* 23.1 (2020), p. 18.

[122] Felix Lindner, Robert Mattmüller, and Bernhard Nebel. "Evaluation of the moral permissibility of action plans". In: *Artificial Intelligence* 287 (2020), p. 103350.

[123] Christian List. "Social choice theory". In: (2013).

[124] Yifei Liu et al. "Ml-ljp: Multi-law aware legal judgment prediction". In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2023, pp. 1023–1034.

[125] House Of Lords et al. "AI in the UK: ready, willing and able?" In: *Retrieved August* 13 (2018), p. 2021.

[126] Nathaniel Love and Michael Genesereth. "Computational law". In: *Proceedings of the 10th international conference on Artificial intelligence and law*. 2005, pp. 205–209.

[127] William MacAskill. "Normative uncertainty as a voting problem". In: *Mind* 125.500 (2016), pp. 967–1004.

[128] Elinor Mason. "Value pluralism". In: (2006).

[129] John McCarthy and Patrick J Hayes. "Some philosophical problems from the standpoint of artificial intelligence". In: *Readings in artificial intelligence*. Elsevier, 1981, pp. 431–450.

[130] Deborah L McGuinness, Frank Van Harmelen, et al. "OWL web ontology language overview". In: *W3C recommendation* 10.10 (2004), p. 2004.

[131] Paul McNamara. "Deontic logic". In: *Handbook of the History of Logic*. Vol. 7. Elsevier, 2006, pp. 197–288.

[132] Dennis Merritt. *Building expert systems in Prolog*. Springer Science & Business Media, 2012.

[133] Microsoft. *Responsible Bots: 10 Guidelines for Developers of Conversational AI*. Online. Accessed on 19th February 2022. 2018. URL: `https://www.microsoft.com/en-us/research/uploads/prod/2018/11/Bot_Guidelines_Nov_2018.pdf`.

[134] John Stuart Mill. "Utilitarianism". In: *Seven masterpieces of philosophy*. Routledge, 2016, pp. 329–375.

[135] Rob Miller and Murray Shanahan. "Some alternative formulations of the event calculus". In: *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski Part II*. Springer, 2002, pp. 452–490.

[136] Rob Miller and Murray Shanahan. *The event calculus in classical logic-alternative axiomatisations*. Linköping University Electronic Press, 1999.

[137] Catrin Misselhorn et al. *Artificial moral agents*. 2022.

[138] Brent Mittelstadt. "Principles alone cannot guarantee ethical AI". In: *Nature machine intelligence* 1.11 (2019), pp. 501–507.

[139] Laurens Mommers. "Ontologies in the legal domain". In: *Theory and Applications of Ontology: Philosophical Perspectives* (2010), pp. 265–276.

[140] James H Moor. "Is ethics computable?" In: *Metaphilosophy* 26.1/2 (1995), pp. 1–21.

[141] James H Moor. "The nature, importance, and difficulty of machine ethics". In: *IEEE intelligent systems* 21.4 (2006), pp. 18–21.

[142] George Edward Moore, Thomas Baldwin, and Thomas Baldwin. *Principia ethica*. Vol. 2. Cambridge University Press Cambridge, 1903.

[143] Erik T Mueller. "Event calculus". In: *Foundations of Artificial Intelligence* 3 (2008), pp. 671–708.

[144] Erik T Mueller. "Event calculus reasoning through satisfiability". In: *Journal of Logic and Computation* 14.5 (2004), pp. 703–730.

[145] Dana S Nau et al. “SHOP2: An HTN planning system”. In: *Journal of artificial intelligence research* 20 (2003), pp. 379–404.

[146] Monica Nogueira et al. “An A-Prolog decision support system for the Space Shuttle”. In: *Practical Aspects of Declarative Languages: Third International Symposium, PADL 2001 Las Vegas, Nevada, March 11–12, 2001 Proceedings 3*. Springer. 2001, pp. 169–183.

[147] Ritesh Noothigattu et al. “A voting-based system for ethical decision making”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[148] Hannu Nurmi. “Voting systems for social choice”. In: *Handbook of Group Decision and Negotiation*. Springer, 2010, pp. 167–182.

[149] Monica Palmirani et al. “Legal ontology for modelling GDPR concepts and norms”. In: *Frontiers in Artificial Intelligence and Applications* 313 (2018), pp. 91–100.

[150] Monica Palmirani et al. “Legalruleml: Xml-based rules and norms”. In: *Rule-Based Modeling and Computing on the Semantic Web: 5th International Symposium, RuleML 2011–America, Ft. Lauderdale, FL, Florida, USA, November 3-5, 2011. Proceedings*. Springer. 2011, pp. 298–312.

[151] Monica Palmirani et al. “Pronto: Privacy ontology for legal reasoning”. In: *Electronic Government and the Information Systems Perspective: 7th International Conference, EGOVIS 2018, Regensburg, Germany, September 3–5, 2018, Proceedings 7*. Springer. 2018, pp. 139–152.

[152] Harshvardhan J Pandit et al. “Creating a vocabulary for data privacy”. In: *OTM Confederated International Conferences” On the Move to Meaningful Internet Systems”*. Springer. 2019, pp. 714–730.

[153] Zdzisaw Pawlak and Roman Sowinski. “Rough set approach to multi-attribute decision analysis”. In: *European journal of Operational research* 72.3 (1994), pp. 443–459.

[154] Fernando CN Pereira and Stuart M Shieber. *Prolog and natural-language analysis*. Microtome Publishing, 2002.

[155] Dana Pessach and Erez Shmueli. “A review on fairness in machine learning”. In: *ACM Computing Surveys (CSUR)* 55.3 (2022), pp. 1–44.

[156] Körner Philipp et al. “Fifty Years of Prolog and Beyond”. In: *THEORY AND PRACTICE OF LOGIC PROGRAMMING* 22.6 (2022), pp. 776–858.

[157] Shelley Powers. *Practical RDF: solving problems with the resource description framework*. ” O’Reilly Media, Inc.”, 2003.

[158] Livio Robaldo and Xin Sun. “Reified input/output logic: combining input/output logic and reification to represent norms coming from existing legislation”. In: *Journal of Logic and Computation* 27.8 (2017), pp. 2471–2503.

[159] Livio Robaldo et al. "Formalizing GDPR provisions in reified I/O logic: the DAPRECO knowledge base". In: *Journal of Logic, Language and Information* 29 (2020), pp. 401–449.

[160] Livio Robaldo et al. *Introduction for artificial intelligence and law: special issue "natural language processing for legal texts"*. 2019.

[161] William David Ross. *The right and the good*. Oxford University Press, 2002.

[162] Catharina Rudschies, Ingrid Schneider, and Judith Simon. "Value pluralism in the AI ethics debate–different actors, different priorities". In: *The International Review of Information Ethics* 29 (2020).

[163] Fariba Sadri and Robert A Kowalski. "Variants of the event calculus". In: (1995).

[164] Ahti Salo and Raimo P Hämäläinen. "Multicriteria decision analysis in group decision processes". In: *Handbook of group decision and negotiation* (2010), pp. 269–283.

[165] Ute Schmid. *Inductive synthesis of functional programs: universal planning, folding of finite programs, and schema abstraction by analogical reasoning*. Vol. 2654. Springer Science & Business Media, 2003.

[166] Murray Shanahan. *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT press, 1997.

[167] Murray Shanahan. "The event calculus explained". In: *Artificial intelligence today: Recent trends and developments*. Springer, 2001, pp. 409–430.

[168] Keng Siau and Weiyu Wang. "Artificial intelligence (AI) ethics: ethics of AI and ethical AI". In: *Journal of Database Management (JDM)* 31.2 (2020), pp. 74–87.

[169] Henry Sidgwick. *The methods of ethics*. DigiCat, 2022.

[170] Peter Singer. "Practical ethics 2nd ed". In: *New York: Cambridge University* (1993).

[171] Marianne WMC Six Dijkstra et al. "Ethical considerations of using machine learning for decision support in occupational health: an example involving periodic workers' health assessments". In: *Journal of Occupational Rehabilitation* 30 (2020), pp. 343–353.

[172] Robert Sparrow. "Why machines cannot be moral". In: *AI & SOCIETY* 36.3 (2021), pp. 685–693.

[173] Daniel Statman. "Introduction to virtue ethics". In: *Virtue ethics: A critical reader* (1997), pp. 1–41.

[174] Mark E Stickel. "A Prolog technology theorem prover: Implementation by an extended Prolog compiler". In: *Journal of Automated reasoning* 4 (1988), pp. 353–380.

[175] Michael Stocker. *Plural and conflicting values*. Clarendon Press, 1992.

[176] Yousef Taheri, Gauvain Bourgne, and Jean-Gabriel Ganascia. "A Compliance Mechanism for Planning in Privacy Domain Using Policies". In: *New Frontiers in Artificial Intelligence*. Ed. by Katsutoshi Yada et al. Cham: Springer Nature Switzerland, 2023, pp. 77–92. ISBN: 978-3-031-36190-6.

[177] Yousef Taheri, Gauvain Bourgne, and Jean-Gabriel Ganascia. "Modelling Integration of Responsible AI Values for Ethical Decision Making". In: *Workshop on Computational Machine Ethics, International Conference on Principles of Knowledge Representation and Reasoning*. 2023.

[178] Suzanne Tolmeijer et al. "Implementations in machine ethics: A survey". In: *ACM Computing Surveys (CSUR)* 53.6 (2020), pp. 1–38.

[179] Gregory Velazco y Trianosky. "What is virtue Ethics All About?" In: *Virtue Ethics* (1997), p. 42.

[180] A Tutorial. "LegalRuleML: from Metamodel to Use Cases". In: ().

[181] GOV UK. *Initial code of conduct for data-driven health and care technology*. 2019.

[182] UNI Global. *10 Principles for Ethical AI*. Online. Accessed on 9th April 2022. 2017. URL: `https://uniglobalunion.org/wp-content/uploads/uni_ethical_ai.pdf`.

[183] *United States Code. Title 17, Sec. 107*. Accessed: date. URL: `http://uscode.house.gov/`.

[184] Cédric Villani, Yann Bonnet, Bertrand Rondepierre, et al. *For a meaningful artificial intelligence: Towards a French and European strategy*. Conseil national du numérique, 2018.

[185] Wendell Wallach and Colin Allen. *Moral machines: Teaching robots right from wrong*. Oxford University Press, 2008.

[186] Wendell Wallach, Colin Allen, and Iva Smit. "Machine morality: bottom-up and top-down approaches for modelling human moral faculties". In: *Machine Ethics and Robot Ethics*. Routledge, 2020, pp. 249–266.

[187] Bernhard Waltl et al. "Classifying legal norms with active machine learning." In: *JURIX*. 2017, pp. 11–20.

[188] Yifan Wang et al. "A survey on the fairness of recommender systems". In: *ACM Transactions on Information Systems* 41.3 (2023), pp. 1–43.

[189] Meredith Whittaker et al. *AI now report 2018*. AI Now Institute at New York University New York, 2018.

[190] Jess Whittlestone et al. "The role and limits of principles in AI ethics: towards a focus on tensions". In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. 2019, pp. 195–200.

[191] Jan Wielemaker et al. "Swi-prolog". In: *Theory and Practice of Logic Programming* 12.1-2 (2012), pp. 67–96.

[192] Edward Wierenga. “A defensible divine command theory”. In: *Nous* (1983), pp. 387–407.

[193] Runhua Xu, Nathalie Baracaldo, and James Joshi. “Privacy-preserving machine learning: Methods, challenges and directions”. In: *arXiv preprint arXiv:2108.04417* (2021).

[194] Gary Chan Kok Yew. “Trust in and ethical design of carebots: the case for ethics of care”. In: *International Journal of Social Robotics* 13.4 (2021), pp. 629–645.

[195] Constantin Zopounidis and Michael Doumpos. “Multicriteria classification and sorting methods: A literature review”. In: *European Journal of Operational Research* 138.2 (2002), pp. 229–246.

# Appendices

# Appendix A

# Codes: Data Processing with GDPR Compliance

```
%=======================================
%=== Planning domain ==================
%=======================================

%+++++++++++++++++++++ inputs ++++++++++++++++++++++++++

%========= initial state =====================
initially( hasData(s2,d1)).

%======= gaol state ================================
requestAnalysis(s5, d1, marketing).
holds(goal,T):- holds( hasData(A, analysisOutput(D,P)), T ),
    ↪ requestAnalysis(A,D,P).
:- holds(goal,T), not holds( hasData(A, analysisOutput(D,P)), T ) ,
    ↪ requestAnalysis(A,D,P).
:- not holds(goal, maxtime).

#const maxtime=5.

%#show occurs/2.
#show missing/2.

%============ storage and data ================================

storage(s1; s2; s3; s4; s5; s6; s7).
connected(s1, s2).
connected(s1, s4).
connected(s1, s6).
connected(s2, s3).
connected(s3, s4).
```

```
connected(s3, s5).
connected(s4, s7).
connected(s5, s7).
connected(s6, s7).
connected(A,B):- connected(B,A).

%%% storage properties: controller
has(s1, controller, aCompany).
has(s2, controller, aCompany).
has(s3, controller, aCompany).
has(s4, controller, aCompany).
has(s5, controller, aCompany).
has(s6, controller, aProcessor).
has(s7, controller, aCompany).

%%% storage properties: location
has(s1, location, us).
has(s2, location, eu).
has(s3, location, eu).
has(s4, location, eu).
has(s5, location, eu).
has(s6, location, us).
has(s7, location, eu).

has(s4, analysisPurpose ,marketing).
has(s4, analysisPurpose ,personalisedAdvertising).

has(s6, analysisPurpose ,optimisationForController).

resource(d1; d2; d3; d4; d5; d6; d7).

has(d1, dataCategory, purchasesAndSpendingHabit).
has(d2, dataCategory, serviceConsumptionBehavior).
has(d3, dataCategory, authenticationHistory).
has(d4, dataCategory, demographic).
has(d5, dataCategory, contact).
has(d6, dataCategory, interest).
has(d7, dataCategory, opinionData).


%================== action specification

%initiate actions with the given purpose
purpose(P):- requestAnalysis(_,_,P).

data(D):- resource(D).
data( analysisOutput(D,P)):-  resource(D), purpose(P).
processedData( analysisOutput(D,P)):- data( analysisOutput(D,P) ).

has( analysisOutput(D,P), dataCategory, X):- data(analysisOutput(D,P)),
```

```
    ↪ has( D , dataCategory, X).

act(transfer(D,A,B,P)):- data(D), storage(A),storage(B), purpose(P),
    ↪ connected(A,B), A!=B.
prec( hasData(A,D), transfer(D,A,B,P)):- act(transfer(D,A,B,P)).
effect(transfer(D,A,B,P), hasData(B,D)):- act(transfer(D,A,B,P)).
effect(transfer(D,A,B,P), neg(hasData(A,D))):- act(transfer(D,A,B,P)).

act(analyze(D,A,P)):- data(D), storage(A), purpose(P) ,has(A,
    ↪ analysisPurpose ,P), not processedData(D).
prec( hasData(A,D), analyze(D,A,P)):- act(analyze(D,A,P)).
effect(analyze(D,A,P), hasData( A, analysisOutput(D,P)  )):- act(analyze(
    ↪ D,A,P)).
effect(analyze(D,A,P), neg( hasData(A,D) )):- act(analyze(D,A,P)).

%=================== event calculus ===================

time(0..maxtime).

% effect axioms
negative(neg(F)) :- effect(E,neg(F)).
initiates(E,F,T) :- effect(E,F), occurs(E,T), not negative(F).
terminates(E,F,T) :- effect(E,neg(F)), occurs(E,T),time(T).
clipped(F,T) :- terminates(E,F,T).

holds(F,0) :- initially(F).
holds(F,T) :- initiates(E,F,T-1), time(T).
holds(F,T) :- holds(F,T-1), not clipped(F,T-1), time(T).

% precondition axioms
:- occurs(E,T), prec(F,E), not holds(F,T), act(E), time(T).

% action generator
0 {occurs(E, T)} 1 :- act(E), time(T),T<maxtime.
:- occurs(E1,T), occurs(E2,T) ,E1!=E2.
```

Listing A.1: The implementation of the planning domain

```
%%%% legal info inputs
%system_duties(art128_22_SubjectRights; art32_37_Obligations).

system_duties(getValidConsent; getAccessReqs; getRectifyReqs;
    ↪ getDeleteReqs; art32_37_Obligations).

%comment or uncomment this line to check the obligations of chapter5 data
    ↪  transfer
transfer_safeguard(s2, s1, art46_2_e_ApprovedCodeOfConduct).
transfer_safeguard(s4, s1, art46_2_e_ApprovedCodeOfConduct).
transfer_safeguard(s1, s6, art46_2_b_BindingCorporateRulesasperArt47).
```

```
transfer_safeguard(s7, s6, art46_2_b_BindingCorporateRulesasperArt47).
system_legal_basis(art6_1_a_Consent).

%======== assign legal info
% transfer action
has(transfer(D,A,B,P), dataCategory, X) :- act(transfer(D,A,B,P)), has(D,
    ↪  dataCategory, X).
has(transfer(D,A,B,P), storage, B):- act(transfer(D,A,B,P)).
has(transfer(D,A,B,P), purpose, P):- act(transfer(D,A,B,P)).
has(transfer(D,A,B,P), processing, transfer):- act(transfer(D,A,B,P)).
has(transfer(D,A,B,P), recipient, X):- act(transfer(D,A,B,P)), has(B,
    ↪ controller, X).
has(transfer(D,A,B,P), storageLocation, eu):- act(transfer(D,A,B,P)), has
    ↪ (B, location, eu).
has(transfer(D,A,B,P), storageLocation, thirdCountries):- act(transfer(D,
    ↪ A,B,P)), not has(B, location, eu).
has(transfer(D,A,B,P), storageCountry, X):- act(transfer(D,A,B,P)), has(B
    ↪ , location, X).
has(transfer(D,A,B,P), legalBasis, X):- act(transfer(D,A,B,P)),
    ↪ system_legal_basis(X).
has(transfer(D,A,B,P), duty, X):- act(transfer(D,A,B,P)), system_duties(X
    ↪ ).
has(transfer(D,A,B,P), measures, X):- act(transfer(D,A,B,P)),
    ↪ transfer_safeguard(A,B, X).


% analyse action
has(analyze(D,A,P), dataCategory, X) :- act(analyze(D,A,P)), has(D,
    ↪ dataCategory, X).
has(analyze(D,A,P), storage, A):- act(analyze(D,A,P)).
has(analyze(D,A,P), purpose, P):- act(analyze(D,A,P)).
has(analyze(D,A,P), processing, analyze):- act(analyze(D,A,P)).
has(analyze(D,A,P), recipient, X):- act(analyze(D,A,P)), has(A,
    ↪ controller, X).
has(analyze(D,A,P), storageLocation, X):- act(analyze(D,A,P)), has(A,
    ↪ location, X).
has(analyze(D,A,P), legalBasis, X):- act(analyze(D,A,P)),
    ↪ system_legal_basis(X).
has(analyze(D,A,P), duty, X):- act(analyze(D,A,P)), system_duties(X).


%=====================================================
%===== data subject's given consent ==================
%=====================================================

%%%%%%%%%%%% data subject's given consent
consent(c1;c2).

%%consent for transfering data
has(c1, dataCategory, purchasesAndSpendingHabit).
```

```
has(c1, processing, transfer).
has(c1, purpose, marketing).
has(c1, recipient, aCompany).
has(c1, storageLocation, eu).

% consent for analysing data

has(c2, dataCategory, purchasesAndSpendingHabit).
has(c2, processing, analyze).
has(c2, purpose, marketing).
has(c2, recipient, aCompany).
has(c2, storageLocation, eu).

% the below code represents the user‘s consent for transfering data to third countries
% we can check for the obligations of chapter 3 data subjets rights by commenting or uncommenting the code below
%*  *%
consent(c3).
has(c3, dataCategory, purchasesAndSpendingHabit).
has(c3, processing, transfer).
has(c3, purpose, marketing).
has(c3, recipient, aCompany).
has(c3, storageLocation, thirdCountries).



%%%%%%%%%%
% compliance checking
non_coherent(P,C):- has(P,A, Z1) , has(C, A, Z2) ,Z1!=Z2, act(P), consent(C).
validConsentSatisfied(P):- not non_coherent(P,C), act(P), consent(C).
%%%

%=========================================================
%========== policy layers  ==============================
%=========================================================
%the regulatory norms encoded in OWL are part of the SPECIAL policy language taken from
% https://specialprivacy.ercim.eu/platform/pilots-policies-and-the-formalization-of-the-gdpr

% inUnionOf
fulfills(P,F1):- fulfills(P,F2), inUnionOf(F2,F1), act(P).
fulfills(P,F1):- not fulfills(P,F2), inUnionOf(comp(F2),F1), act(P).

% inIntersectOf
incompleteRequirment(P,F1):- inIntersectOf(F2,F1) ,not fulfills(P,F2), act(P).

```

```
fulfills(P,F1):- not incompleteRequirment(P, F1), act(P), inIntersectOf(_
    ↪ ,F1).
fulfills(P , comp(F1)):- not fulfills(P , F1 ), regulation(F1), act(P).
%class hirerarchy
upperClass(A,B):-inUnionOf(A,B).
upperClass(A,B):-inIntersectOf(A,B).



missing1(P,A,A):- not fulfills(P,A),regulation(A), act(P), occurs(P,_).
missing1(P,A,C):- not fulfills(P,C), missing1(P,A,B), upperClass(B,C),
    ↪ regulation(A),regulation(C).
missing(P,A):- missing1(P,A,gdpr_Requirements), not auxiliaryRegulation(A
    ↪ ).
% plan compliance evaluation
%satisfies(non_compliance):-  occurs( O,T) , not fulfills(O,
    ↪ gdpr_Requirements).

%:- satisfies(non_compliance).

regulation(X):- auxiliaryRegulation(X).
%=====================================================
%======= Regulatory nroms ==========================
%=====================================================
%the regulatory norms encoded in OWL are part of the SPECIAL policy
    ↪ language taken from
% https://specialprivacy.ercim.eu/platform/pilots-policies-and-the-
    ↪ formalization-of-the-gdpr

%=================== SPECIAL ================================
%============= GDPR_Requirements ============================
%============================================================
%*
ObjectUnionOf(
  ObjectSomeValuesFrom(spl:hasData
    ObjectComplementOf(PersonalData) % NonPersonalData
  )
  ObjectIntersectionOf(
    Chap2_LawfulProcessing
    Chap3_RightsOfDataSubjects
    Chap4_ControllerAndProcessorObligations
    Chap5_DataTransfer)
  Chap9_Derogations
  )
*%
%================== ASP translation ==========================


regulation(isSensitiveData;
    isCriminalData;
```

```
    art6_LawfulProcessing;
    art6_1_LegalBasis;
    chap2_LawfulProcessing;
    chap9_Derogations;
    art9_2_legalBasis;
    art9_SensitiveData;
    gdpr_Requirements).


auxiliaryRegulation(art6_LawfulProcessing_aux1; chap2_5_aux1).

inUnionOf(chap2_5_aux1, gdpr_Requirements ).
inUnionOf(chap9_Derogations, gdpr_Requirements ).

inIntersectOf(chap2_LawfulProcessing ,chap2_5_aux1).
inIntersectOf(chap3_RightsOfDataSubjects ,chap2_5_aux1).
inIntersectOf(chap4_ControllerAndProcessorObligations ,chap2_5_aux1).
inIntersectOf(chap5_DataTransferToThirdCountry ,chap2_5_aux1).

%============= SPECIAL ================================
%============ Chap2_LawfulProcessing ==================
%=======================================================
%*
ObjectUnionOf(
  Art6_LawfulProcessing
  Art9_SensitiveData
  Art10_CriminalData
  )
*%
%========== ASP translation ==============================

inUnionOf(art6_LawfulProcessing, chap2_LawfulProcessing ).
inUnionOf(art9_SensitiveData, chap2_LawfulProcessing).
inUnionOf(art10_CriminalData, chap2_LawfulProcessing ).

%================ SPECIAL ================================
%== Chap2_LawfulProcessing -> Art6_LawfulProcessing =====
%=======================================================
%*
ObjectIntersectionOf(
  ObjectSomeValuesFrom(spl:hasData PersonalData)
  ObjectSomeValuesFrom(spl:hasData ObjectComplementOf(
    ↪ SensitiveData_as_per_Art9))
  ObjectSomeValuesFrom(spl:hasData ObjectComplementOf(
    ↪ CriminalConvictionData_as_per_Art10))
  ObjectUnionOf(
    Art6_1_LegalBasis
    Art6_4_CompatiblePurpose)
  )
*%
```

```
%=============== ASP translation ============================


sensitiveData( opinionData; sexualData ; racialData ;ethnicData).
criminalData(criminal).

inIntersectOf(comp(isSensitiveData), art6_LawfulProcessing ).
inIntersectOf(comp(isCriminalData), art6_LawfulProcessing ).
inIntersectOf(art6_LawfulProcessing_aux1, art6_LawfulProcessing ).
inUnionOf(art6_1_LegalBasis, art6_LawfulProcessing_aux1 ).
inUnionOf(art6_4_CompatiblePurpose, art6_LawfulProcessing_aux1 ).

fulfills(P , isSensitiveData ):- has(P, dataCategory, X), act(P),
    ↪ sensitiveData(X).
fulfills(P , isCriminalData ):- has(P, dataCategory, X), act(P),
    ↪ criminalData(X).

%================== SPECIAL ==================================
%=== Chap2_LawfulProcessing -> Art6_LawfulProcessing -> Art6_1_LegalBasis
%============================================================
%*
ObjectSomeValuesFrom(hasLegalBasis
  ObjectUnionOf(
    Art_6_1_a_Consent
    Art_6_1_b_Contract
    Art_6_1_c_LegalObligation
    Art_6_1_d_VitalInterest
    Art_6_1_e_PublicInterest
    Art_6_1_f_LegitimateInterest
    )
  )
*%
%============ ASP translation ============================
legalBasis_art6_1( art6_1_a_Consent;
                        art6_1_b_Contract;
                        art6_1_c_LegalObligation;
                        art6_1_d_VitalInterest;
                        art6_1_e_PublicInterestOfficialAuthority;
                        art6_1_f_LegitimateInterest).

fulfills(P ,art6_1_LegalBasis ):- has(P, legalBasis, X), act(P),
    ↪ legalBasis_art6_1(X).


%============= SPECIAL ================================
%==== Chap2_LawfulProcessing -> Art9_SensitiveData ========
%============================================================
%*
ObjectUnionOf(
  ObjectSomeValuesFrom(spl:hasData ObjectComplementOf(
```

```
    ↪ SensitiveData_as_per_Art9))
  ObjectSomeValuesFrom(hasLegalBasis
  ObjectUnionOf(
    Art9_2_a_Consent
    Art9_2_b_EmploymentAndSocialSecurity
    Art9_2_c_VitalInterest
    Art9_2_d_LegitimateActivitiesOfAssociations
    Art9_2_e_PublicData
    Art9_2_f_Juducial
    Art9_2_g_PublicInteres
    Art9_2_h_PreventiveOrOccupationalMedicine
    Art9_2_i_PublicHealth
    Art9_2_j_ArchivingResearchStatistics)
  )
)
*%
%========= ASP translation =====================

inUnionOf(comp(isSensitiveData), art9_SensitiveData ).
inUnionOf(art9_2_legalBasis, art9_SensitiveData ).

legalBasis_art9_2( art9_2_a_Consent;
    art9_2_b_EmploymentAndSocialSecurity;
    art9_2_c_VitalInterest;
    art9_2_d_LegitimateActivitiesOfAssociations;
    art9_2_e_PublicData;
    art9_2_f_Juducial;
    art9_2_g_PublicInterest;
    art9_2_h_PreventiveOrOccupationalMedicine;
    art9_2_i_PublicHealth;
    art9_2_j_ArchivingResearchStatistics).

%fulfills(P ,isSensitiveData ):- has(P, dataCategory, X), act(P),
    ↪ sensitiveData(X).
fulfills(P ,art9_2_legalBasis ):- has(P, legalBasis, X), act(P),
    ↪ legalBasis_art9_2(X).


%============== SPECIAL ================================
%====== Chap2_LawfulProcessing -> Art10_CriminalData ====
%=======================================================
%*
ObjectUnionOf(
  ObjectIntersectionOf(
    ObjectSomeValuesFrom(sbpl:hasDuty Art10_Requirements10)
    Refinements_as_per_Chap9)
  ObjectSomeValuesFrom(spl:hasData ObjectComplementOf(
    ↪ CriminalConvictionData_as_per_Art10)
  )
)
```

```
*%
%============= ASP translation ===================

regulation(art10_CriminalData;refinements_as_per_Chap9).

auxiliaryRegulation(art10_CriminalData_aux1;art10_CriminalData_aux11).

inUnionOf(art10_CriminalData_aux1, art10_CriminalData ).
inUnionOf(comp(isCriminalData), art10_CriminalData ).

inIntersectOf(art10_CriminalData_aux11 ,art10_CriminalData_aux1).
inIntersectOf(refinements_as_per_Chap9 ,art10_CriminalData_aux1).

%fulfills(P ,isCriminalData ):- has(P, dataCategory, X), act(P),
    ↪ criminalData(X).
fulfills(P ,art10_CriminalData_aux11 ):- has(P, duty, art10_Requirements)
    ↪ , act(P).

%================ SPECIAL ============================
%========== Chap3_RightsOfDataSubjects ================
%=======================================================
%*
ObjectUnionOf(
  Exceptions_as_per_Art23
  ObjectSomeValuesFrom(sbpl:hasDuty Art12-22_SubjectRights)
)
*%
%============ ASP translation ==============

regulation(chap3_RightsOfDataSubjects;exceptions_as_per_Art23;
    ↪ art12_22_SubjectRights).
auxiliaryRegulation(chap3_RightsOfDataSubjects_aux1).


inUnionOf(exceptions_as_per_Art23, chap3_RightsOfDataSubjects ).
inUnionOf(chap3_RightsOfDataSubjects_aux1 , chap3_RightsOfDataSubjects ).



%fulfills(P ,chap3_RightsOfDataSubjects_aux1 ):- has(P, duty,
    ↪ art12_22_SubjectRights), act(P). %, validConsentSatisfied(P)


art_12_22_rights(getValidConsent;
                    getAccessReqs;
                    getRectifyReqs;
                    getDeleteReqs).
inUnionOf(art12_22_SubjectRights, chap3_RightsOfDataSubjects_aux1 ).

fulfills(P, art12_22_SubjectRights):- has(P, duty, X) , act(P),
```

```
    ↪ art_12_22_rights(X),validConsentSatisfied(P) .

%fulfills(P, art12_22_SubjectRights):- fulfills(P ,art12_22_SubjectRights
    ↪ ), validConsentSatisfied(P) ,act(P).


%================== SPECIAL =====================
%=== Chap4_ControllerAndProcessorObligations =====
%================================================
%*
ObjectSomeValuesFrom(sbpl:hasDuty Art32-37_Obligations)
*%
%============== ASP translation ==================

regulation(chap4_ControllerAndProcessorObligations).
fulfills(P ,chap4_ControllerAndProcessorObligations ):- has(P, duty,
    ↪ art32_37_Obligations), act(P).


%============= SPECIAL ==============================
%==== Chap5_DataTransferToThirdCountry ===============
%====================================================
%*
ObjectUnionOf(
  ObjectIntersectionOf(
    Art48_TransfersNotAuthorisedByUnionLaw
    ObjectUnionOf(
      AdequateLevelOfProtection_as_per_Art45
      AppropriateSafeguards_as_per_Art46
      Art49_Derogations
    )
  )
  ObjectComplementOf(
    ObjectIntersectionOf(
      ObjectSomeValuesFrom(spl:hasProcessing svpr:Transfer)
      ObjectSomeValuesFrom(spl:hasStorage ObjectSomeValuesFrom(spl:
    ↪ hasLocation svl:ThirdCountries))
    )
  )
)
*%
%====== ASP translation ==================

regulation(chap5_DataTransferToThirdCountry;
    art48_TransfersNotAuthorisedByUnionLaw;
    adequateLevelOfProtection_as_per_Art45;
    appropriateSafeguards_as_per_Art46;
    art49_Derogations).

auxiliaryRegulation(chap5_DataTransfer_aux1; chap5_DataTransfer_aux2;
```

```
    ↪ chap5_DataTransfer_aux11; is_transfer ; to_thirdCountries).

inUnionOf(chap5_DataTransfer_aux1 , chap5_DataTransferToThirdCountry ).
inUnionOf(comp(chap5_DataTransfer_aux2) ,
    ↪ chap5_DataTransferToThirdCountry ).


inIntersectOf(art48_TransfersNotAuthorisedByUnionLaw,
    ↪ chap5_DataTransfer_aux1).
inIntersectOf(chap5_DataTransfer_aux11 ,chap5_DataTransfer_aux1).

inUnionOf(adequateLevelOfProtection_as_per_Art45 ,
    ↪ chap5_DataTransfer_aux11).
inUnionOf(appropriateSafeguards_as_per_Art46 ,chap5_DataTransfer_aux11).
inUnionOf(art49_Derogations ,chap5_DataTransfer_aux11).


inIntersectOf(is_transfer ,chap5_DataTransfer_aux2).
inIntersectOf(to_thirdCountries ,chap5_DataTransfer_aux2).

fulfills(P ,is_transfer ):- has(P, processing, transfer), act(P).
fulfills(P ,to_thirdCountries):- has(P, storageLocation, thirdCountries),
    ↪  act(P).


%============= SPECIAL ==================================
%= Chap5_DataTransferToThirdCountry ->
    ↪ Art48_TransfersNotAuthorisedByUnionLaw =
%=======================================================
%*
ObjectUnionOf(
  InternationalAgreement_as_in_Art48
  ObjectComplementOf(CourtRequestFromThirdCountry_as_in_Art48)
)
*%
%=== ASP translation ============================

regulation(internationalAgreement_as_in_Art48).
inUnionOf(internationalAgreement_as_in_Art48 ,
    ↪ art48_TransfersNotAuthorisedByUnionLaw).
%inUnionOf(comp(courtRequestFromThirdCountry_as_in_Art48) ,
    ↪ art48_TransfersNotAuthorisedByUnionLaw).
fulfills(P ,internationalAgreement_as_in_Art48 ):- has(P, processing,
    ↪ transfer), has(P, storageCountry, us ), act(P).

% here we suppose that us is among the countries with an international
    ↪ agreement with EU
%============== SPECIAL ==============================
%=== Chap5_DataTransferToThirdCountry ->
    ↪ AppropriateSafeguards_as_per_Art46 ==
```

```
%=====================================================
%*
Art46_2_a_PublicAuthorities
Art46_2_b_BindingCorporateRulesasperArt47
Art46_2_c_DataProtectionClausesAdoptedByEC
Art46_2_d_DataProtectionClausesAdoptedBySupervisoryAuthority
Art46_2_e_ApprovedCodeOfConduct
Art46_2_f_ApprovedCertificationMechanism
Art46_3_a_ContractualClauses
Art46_3_b_ProvisionsInAdministrativeArrangements
*%
%============== ASP translation ====================

appropriateSafeguards_Art46(
  art46_2_a_PublicAuthorities;
  art46_2_b_BindingCorporateRulesasperArt47;
  art46_2_c_DataProtectionClausesAdoptedByEC;
  art46_2_d_DataProtectionClausesAdoptedBySupervisoryAuthority;
  art46_2_e_ApprovedCodeOfConduct;
  art46_2_f_ApprovedCertificationMechanism;
  art46_3_a_ContractualClauses;
  art46_3_b_ProvisionsInAdministrativeArrangements).

fulfills(P ,appropriateSafeguards_as_per_Art46 ):- has(P,   measures   , X
    ↪ ), act(P), appropriateSafeguards_Art46(X).



%================ SPECIAL ==================================
%============ Chap5_DataTransferToThirdCountry -> Art49_Derogations =
%=========================================================
%*
ObjectIntersectionOf(
  ObjectUnionOf(
    Art49_1_a_Consent
    Art49_1_b_ContractByRequestOfDS
    Art49_1_c_ContractInInterestOfDS
    Art49_1_d_PublicInterest
    Art49_1_e_LegalClaims
    Art49_1_f_VitalInterest
    Art49_1_g_PublicData)
  ObjectComplementOf(AdequateLevelOfProtection_as_per_Art45)
  ObjectComplementOf(AppropriateSafeguards_as_per_Art46)
)
*%

%============= ASP translation ==================


auxiliaryRegulation(art49_Derogations_aux1).
```

```
457
458 legalBasis_Art49_1(
459   art49_1_a_Consent;
460   art49_1_b_ContractByRequestOfDS;
461   art49_1_c_ContractInInterestOfDS;
462   art49_1_d_PublicInterest;
463   art49_1_e_LegalClaims;
464   art49_1_f_VitalInterest;
465   art49_1_g_PublicData).
466
467 inIntersectOf(art49_Derogations_aux1 ,art49_Derogations).
468 fulfills(P ,art49_Derogations_aux1 ):- has(P,   legalBasis   , X), act(P),
      ↪  legalBasis_Art49_1(X).
469 inIntersectOf(comp(adequateLevelOfProtection_as_per_Art45) ,
      ↪ art49_Derogations).
470 inIntersectOf(comp(appropriateSafeguards_as_per_Art46) ,art49_Derogations
      ↪ ).
```

Listing A.2: The implementation of the compliance engine

# Appendix B

# Codes: Pluralistic Utility Model

```
%=========================================================
%======================== Alternatives ==================
%=========================================================
alt(sys1;sys2;sys3).

has(sys1,requiredData,clickHabit ).
has(sys1,requiredData,interests ).
has(sys1,requiredData,activity ).

has(sys2,requiredData,interests ).
has(sys2,requiredData,politicalBelief ).

has(sys3, requiredData ,interests).
has(sys3, requiredData ,activity).

has(sys1, item_parity,3 ).
has(sys2, item_parity,6 ).
has(sys3, item_parity,4 ).

has(sys1, gender_parity,1 ).
has(sys2, gender_parity,2 ).
has(sys3, gender_parity,3 ).

has(sys1, racial_parity,2 ).
has(sys2, racial_parity,4 ).
has(sys3, racial_parity,1 ).

has(sys1, perfMetric,30 ).
has(sys2, perfMetric,25 ).
has(sys3, perfMetric,20 ).

```

```
has(sys1, processType, largeScale ).
has(sys2, processType, largeScale ).
has(sys3, processType, smallScale ).

%======================================================================
%======================== Hierarchical Structure ======================
%======================================================================

child(root,privacy).
child(root,fairness).
child(root,performance).

child(privacy, sensitivity).
child(privacy, minimization).
child(privacy, scaleComplexity).
child(fairness, item_fairness).
child(fairness, user_fairness).
child(user_fairness, racial_fairness).
child(user_fairness, gender_fairness).

node(Node):- child(Node,_).
node(Node):- child(_,Node).

%======================================================================
%======================== leaf criteria votes =========================
%======================================================================

% Minimisation
pref(minimization,Alt1,Alt2):-
    has(Alt1, nbData, N1),
    has(Alt2, nbData, N2),
    N2>=N1.

has(Alt,nbData, N):-
    N = #count{ Data : has(Alt, requiredData, Data)},
    alt(Alt).

%Sensitivity
pref(sensitivity, Alt1, Alt2):-
    has(Alt1, nbSensitiveData, N1),
    has(Alt2,nbSensitiveData, N2),
    N2>=N1.

has(Alt, nbSensitiveData, N):-
    N = #count{ Data : has(Alt, requiredData, Data),
    has(Data, category, sensitiveData)},
    alt(Alt).
% knowledge about sensitive data categories
has(politicalBelief, category, sensitiveData).
has(opinionData, category, sensitiveData).
```

```
82
83
84 %Scale and Complexity
85 pref(scaleComplexity,Alt1,Alt2):-
86     has(Alt1, processType, Type1),
87      has(Alt2, processType, Type2),
88      rankAux(Type1, R1),
89      rankAux(Type2, R2),
90      R2>=R1, alt(Alt1), alt(Alt2).
91 rankAux(largeScale, 2).
92 rankAux(smallScale, 1).
93
94
95 % Performance
96 pref(performance, Alt1, Alt2):-
97     has(Alt1, perfMetric, P1),
98     has(Alt2, perfMetric, P2),
99     P1>=P2, alt(Alt1), alt(Alt2).
100
101 % Item Fairness
102 pref(item_fairness,Alt1,Alt2):-
103     has(Alt1, item_parity,P1),
104     has(Alt2, item_parity,P2),
105     P1<=P2, alt(Alt1), alt(Alt2).
106
107 %Item Fairness
108 pref(racial_fairness,Alt1,Alt2):-
109     has(Alt1, racial_parity,P1),
110     has(Alt2, racial_parity,P2),
111     P1<=P2, alt(Alt1), alt(Alt2).
112
113 %Gender Fairness
114 pref(gender_fairness,Alt1,Alt2):-
115     has(Alt1, gender_parity,P1),
116     has(Alt2, gender_parity,P2),
117     P1<=P2, alt(Alt1), alt(Alt2).
118
119 %=====================================================================
120 %========================== Aggregator ===========================
121 %=====================================================================
122
123 % Implementation of votng rule with dominant voters
124 pref(Node, Alt1, Alt2):- prefLex(Node, Alt1, Alt2).
125
126 prefLex(Node, Alt1, Alt2):-
127     childrenClass(Node,Class),
128     pVote(Node, Class, Alt1, Alt2),
129     is_dominant(Node, Class, Alt1, Alt2).
130
131 is_dominant(Node, Class,Alt1,Alt2):-
```

```
    not is_dominated(Node, Class,Alt1,Alt2),
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2).

is_dominated(Node, Class, Alt1,Alt2):-
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2),
    superiorThan(Node, Layer1, Class),
    not pVote(Node, Layer1, Alt1,Alt2).

is_dominated(Node, Class, Alt1,Alt2):-
    childrenClass(Node,Class),
    pVote(Node, Class, Alt1, Alt2),
    not superiorThan(Node, _ , Class),
    pVote(Node, Class, Alt2, Alt1).

superiorThan(Node, Layer1, Layer2):-
    childrenClass(Node, Layer1),
    childrenClass(Node, Layer2),
    not inferiorThan(Node, Layer1, Layer2).

inferiorThan(Node, Layer1, Layer2):-
    childrenClass(Node, Layer1),
    childrenClass(Node, Layer2),
    belongs(Node, Child1 , Layer1),
    belongs(Node, Child2 , Layer2),
    not childPref(Node, Child1, Child2).

childrenClass(Node, Class):-
    belongs(Node, _ , Class).

belongs(Node, Child , Class):-
    Class = #count{ Child1 :
        childPref(Node , Child, Child1)},
    child(Node, Child).


% voting rule

% in case of an only single voter no vote aggregation is needed
pVote(Node, Class, Alt1,Alt2):-
    singleton(Node, Class),
    belongs(Node, Child, Class),
    pref(Child, Alt1, Alt2).

singleton(Node, Class):-
    belongs(Node, Child1, Class),
    not plural(Node, Class).

plural(Node, Class):-
```

```
    belongs(Node, Child1, Class),
    belongs(Node, Child2, Class),
    Child1 != Child2.


% vote aggregation according to Copelands rule
pVote(Node, Class, Alt1, Alt2):-
    plural(Node, Class),
    copeland_score( Node, Class, Alt1, S1),
    copeland_score( Node, Class, Alt2, S2),
    S1>=S2.

copeland_score( Node, Class, Alt, S):-
    S= #sum{ S1:
        nb_pairwise_wins( Node, Class, Alt, Alt1, N1),
        nb_pairwise_ties( Node, Class, Alt, Alt1, N2),
        S1 = N1*2+N2,
        alt(Alt1)},
    childrenClass(Node, Class), alt(Alt) .

nb_pairwise_ties( Node, Class, Alt1, Alt2 , N):-
    N= #count{ N1 :
        nb_strict_voters(Node, Class, Alt1, Alt2 , N1),
        nb_strict_voters(Node, Class, Alt2, Alt1 , N2),
        belongs(Node, Child, Class), N1=N2},
    childrenClass(Node,Class), alt(Alt1), alt(Alt2).

nb_pairwise_wins( Node, Class, Alt1, Alt2, N):-
    N= #count{ N1 :
        nb_strict_voters(Node, Class, Alt1, Alt2 , N1),
        nb_strict_voters(Node, Class, Alt2, Alt1 , N2),
        belongs(Node, Child, Class), N1>N2},
    childrenClass(Node, Class), alt(Alt1), alt(Alt2).

nb_strict_voters(Node, Class, Alt1, Alt2, N):-
    N = #count{ Child :
        pref(Child, Alt1, Alt2),
        not pref(Child , Alt2, Alt1),
        belongs(Node, Child, Class) },
    childrenClass(Node, Class), alt(Alt1), alt(Alt2).


%% ranking adjustment
primiryRank(Node, Alt, R):-
    R1 = #count{ Alt1 : pref(Node, Alt, Alt1), alt(Alt1)} ,
    R2 = #count{ Alt2 : pref(Node, Alt2,_)},
    R = R2-R1+1, node(Node), alt(Alt).

correction(Node, R2 , R+1):-
    R = #count{ R1 : primiryRank(Node,_,R1), R2>R1 },
```

```
    primiryRank(Node,_,R2).

rank(Node, Alt , R):-
    primiryRank(Node, Alt, R1),
    correction(Node, R1 , R).

%root ranking
rankRoot(Alt,R):- rank(root,Alt,R).

%======================================================================
%======================== Input priorities ========================
%======================================================================
% Fixed Inputs
childPref(privacy, sensitivity, minimization).
%childPref(privacy, minimization, sensitivity).
childPref(privacy, minimization, scaleComplexity).
%childPref(privacy, scaleComplexity,  minimization).

childPref(fairness, user_fairness, item_fairness).
childPref(user_fairness, racial_fairness, gender_fairness).
childPref(user_fairness, gender_fairness,  racial_fairness).


% transitivity of the order
childPref(C, X,Z):- childPref(C, X,Y), childPref(C, Y,Z).


% variant inputs for the root to reproduce the result in the article

% case 1) fairness = privacy = pereformance

%childPref(root, fairness, privacy).
%childPref(root, privacy, fairness).
%childPref(root, privacy, performance).
%childPref(root, performance,privacy).


% case 2 fairness > privacy > pereformance

%childPref(root, fairness, privacy).
%childPref(root, privacy, performance).


% case 3 fairness = privacy > pereformance

%childPref(root, fairness, privacy).
%childPref(root, privacy, fairness).
%childPref(root, privacy, performance).

```

```
% case 4 fairness = pereformance > privacy

childPref(root, fairness, performance).
childPref(root, performance, fairness).
childPref(root, performance,privacy).


% case 5 fairness > pereformance > privacy

%childPref(root, fairness, performance).
%childPref(root, performance,privacy).


%======================================================================
%========================== output setting ============================
%======================================================================
% in order to run the code execute:
% clingo filename.lp 0

% show the rankoing of all nodes
#show rank/3.

% shows only the ranking of the root node
%#show rankRoot/2.
```

Listing B.1: The implementation of pluralistic utility model

# Appendix C

# Codes: Health Care Delivery System

```
# -*- coding: utf-8 -*-
"""drugDeliveryV5.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1E_UtcQpSNKNxZXMeOvBwJVNESvArcO4i

## Start Here
"""

pip install clingo

import clingo

"""#Planning Component

## Event Calculus
"""

# Commented out IPython magic to ensure Python compatibility.
#Global Knowledge
eventCalculusStr="""

# %================== event calculus ==================
time(0..maxtime).

# % effect axioms
negative(neg(F)) :- effect(E,neg(F)).
```

```
initiates(E,F,T) :- effect(E,F), performs(E,T), not negative(F).
terminates(E,F,T) :- effect(E,neg(F)), performs(E,T),time(T).
clipped(F,T) :- terminates(E,F,T).


holds(F,0) :- initially(F).
holds(F,T) :- initiates(E,F,T-1), time(T).
holds(F,T) :- holds(F,T-1), not clipped(F,T-1), time(T).

# % precondition axioms
:- performs(E,T), prec(F,E), not holds(F,T), act(E), time(T).

# % action generator
0 {performs(E, T)} 1 :- act(E), time(T),T<maxtime.
:- performs(E1,T), performs(E2,T) ,E1!=E2.

# %=================== event calculus ===================

"""

"""##Action Specifications"""

# Commented out IPython magic to ensure Python compatibility.
#global knowledge
actionsStr="""

# %=======================================================


act(board(Agent, Resource, LocX)):-
    agent(Agent),
    resource(Resource),
    location(LocX).

prec(resourceAt(Resource, LocX), board(Agent, Resource, LocX)):-
    act(board(Agent, Resource, LocX)).
prec(agentAt(Agent, LocX), board(Agent, Resource, LocX)):-
    act(board(Agent, Resource, LocX)).
prec(empty(Agent), board(Agent, Resource, LocX)):-
    act(board(Agent, Resource, LocX)).

effect(board(Agent, Resource, LocX), onBoard(Resource, Agent)):-
    act(board(Agent, Resource, LocX)).

effect(board(Agent, Resource, LocX), neg(resourceAt(Resource, LocX))):-
    act(board(Agent, Resource, LocX)).

effect(board(Agent, Resource, LocX), neg(empty(Agent))):-
    act(board(Agent, Resource, LocX)).

```

```
# % move resource action
act(move(Agent, LocX, LocY)):-
    agent(Agent),
    location(LocX),
    location(LocY),
    connected(LocX, LocY).

prec(agentAt(Agent, LocX) ,move(Agent, LocX, LocY)):-
    act(move(Agent, LocX, LocY)).

effect( move(Agent, LocX, LocY), agentAt(Agent, LocY) ):-
    act(move(Agent, LocX, LocY)).

effect( move(Agent, LocX, LocY), neg(agentAt(Agent, LocX))):-
    act(move(Agent, LocX, LocY)).


# %======= dliver action
act(deliver(Agent, Resource, Demand, LocX)):-
    agent(Agent),
    resource(Resource),
    demand(Demand),
    location(LocX).


prec(agentAt(Agent, LocX), deliver(Agent, Resource, Demand, LocX)):-
    act(deliver(Agent, Resource, Demand, LocX)).

prec(demandAt(Demand, LocX), deliver(Agent, Resource, Demand, LocX)):-
    act(deliver(Agent, Resource, Demand, LocX)).
prec(onBoard(Resource, Agent), deliver(Agent, Resource, Demand, LocX)):-
    act(deliver(Agent, Resource, Demand, LocX)).

prec(active(Demand), deliver(Agent, Resource, Demand, LocX)):-
    act(deliver(Agent, Resource, Demand, LocX)).


effect(deliver(Agent, Resource, Demand, LocX), delivered(Agent, Resource,
    ↪ Demand) ):-
    act(deliver(Agent, Resource, Demand, LocX)).

effect(deliver(Agent, Resource, Demand, LocX), empty(Agent) ):-
    act(deliver(Agent, Resource, Demand, LocX)).

effect(deliver(Agent, Resource, Demand, LocX), neg(active(Demand)) ):-
    act(deliver(Agent, Resource, Demand, LocX)).
effect(deliver(Agent, Resource, Demand, LocX), neg(onBoard(Resource,
    ↪ Agent)) ):-
    act(deliver(Agent, Resource, Demand, LocX)).

```

```
128
129
130 agent1(board(Agent, Resource, LocX),Agent):- act(board(Agent, Resource,
        ↪ LocX)).
131 agent1(move(Agent, LocX, LocY),Agent):- act(move(Agent, LocX, LocY)).
132 agent1(deliver(Agent, Resource, Demand, LocX),Agent):- act(deliver(Agent,
        ↪  Resource, Demand, LocX)).
133
134
135 """
136
137 """##Maps (Nodes and Connections)"""
138
139 # Commented out IPython magic to ensure Python compatibility.
140 outputStr="""
141 # %============  Goal State and Output ================================
142
143 :- not holds(delivered(Agent, Resource, Demand), maxtime), assignedDemand
        ↪ (Demand, Agent), demandedResource1(Demand, Resource) .
144
145
146 #show performs/2.
147 # %#show initially/1.
148
149 """
150
151 # Commented out IPython magic to ensure Python compatibility.
152 mapStr="""
153
154 # %===========
155 # %nodes and connections
156
157 location(locA;locB;locC;locD).
158 connected(locA, locB).
159 connected(locA, locC).
160 connected(locB, locD).
161 connected(locC, locD).
162
163
164 connected(X,Y):- connected(Y,X).
165
166 distance(locA, locB, 8).
167 distance(locA, locC, 8).
168 distance(locB, locD, 6).
169 distance(locC, locD, 4).
170
171 distance(A, B, D):- distance(B, A, D).
172
173 prohibited(drone, locB).
174 prohibited(autoBox, locC).
```

```
175
176 """
177
178 """##Resources"""
179
180 resourcesInfoStr="""
181
182 resource(r1;r2).
183
184 initially(resourceAt(r1, locA)).
185 initially(resourceAt(r2, locA)).
186
187 """
188
189 """##Agents"""
190
191 agentInfoStr="""
192
193 agent(a1;a2).
194
195 type(a1, drone).
196 type(a2, autoBox).
197
198
199 speed(drone, 2).
200 speed(autoBox, 1).
201
202 depreciation(drone, 2).
203 depreciation(autoBox, 1).
204
205 initially(agentAt(a1, locA)).
206 initially(agentAt(a2, locA)).
207
208 initially(empty(a1)).
209 initially(empty(a2)).
210
211 """
212
213 """##Demands
214
215 ##Program: Resource Allocations
216 """
217
218 # Commented out IPython magic to ensure Python compatibility.
219 resourceAllocationStr="""
220 # %============= allocations ===================
221 # %single resource problem
222
223 insufficientResource(Resource):-
224     demandedResource1(Demand, Resource),
```

```
    demandedResource1(Demand1, Resource),
    Demand !=Demand1.

sufficientResource(Resource):-
    not insufficientResource(Resource),
    resource(Resource).

allocatedResource(Resource, Demand):-
    demandedResource1(Demand, Resource),
    sufficientResource(Resource).

{allocatedResource(Resource,Demand)}:-
    demandedResource1(Demand, Resource),
    insufficientResource(Resource).

:-  allocatedResource(Resource,Demand1),
    allocatedResource(Resource,Demand2),
    insufficientResource(Resource),
    Demand1!=Demand2.

:-  not allocatedResource(Resource,_),
    insufficientResource(Resource).

# %#show allocatedResource/2.
"""

"""##Program: Demand Assignments"""

# Commented out IPython magic to ensure Python compatibility.
demandAssignmentStr="""

{assignedDemand(Demand,Agent)}:-
  allocatedResource(_, Demand),
  agent(Agent).

:- assignedDemand(Demand,Agent1),
   assignedDemand(Demand,Agent2),
   Agent1!=Agent2.

:- not assignedDemand(Demand,_),
   allocatedResource(_, Demand).


# %resourceDemandAgentAssign(Resource, Demand, Agent):- allocatedResource
    ↪ (Resource, Demand), assignedDemand(Demand, Agent).

# %deliveryTask(Agent, Resource, Demand):- allocatedResource(Resource,
    ↪ Demand), assignedDemand(Demand,Agent).
#show assignedDemand/2.
"""
```

```
273
274 """##Program: Planning
275
276 #Compliance Component
277
278 ##Auxiliary Features
279 """
280
281 # Commented out IPython magic to ensure Python compatibility.
282 gettingFeaturesStr="""
283
284 # % cost informations
285
286 duration( move(Agent, LocX, LocY) , N):- act(move(Agent, LocX, LocY)),
        ↪ type(Agent, X), speed(X, S), distance(LocX, LocY, D), N=D/S.
287 duration( deliver(Agent, Resource, Demand, LocX) , 1):-  type(Agent,
        ↪ drone), act(deliver(Agent, Resource, Demand, LocX)).
288 duration( deliver(Agent, Resource, Demand, LocX) , 1):- type(Agent,
        ↪ autoBox), act(deliver(Agent, Resource, Demand, LocX)).
289 duration( board(Agent, Resource, LocX) , 2):- type(Agent, drone), act(
        ↪ board(Agent, Resource, LocX)).
290 duration( board(Agent, Resource, LocX) , 1):- type(Agent, autoBox), act(
        ↪ board(Agent, Resource, LocX)).
291
292
293
294 cost( move(Agent, LocX, LocY), C):- type(Agent, X), depreciation(X, N) ,
        ↪ distance(LocX, LocY, D), C= N*D, act(move(Agent, LocX, LocY)).
295 cost( deliver(Agent, Resource, Demand, LocX) , 2):-  type(Agent, drone),
        ↪ act(deliver(Agent, Resource, Demand, LocX)).
296 cost( deliver(Agent, Resource, Demand, LocX) , 1):- type(Agent, autoBox),
        ↪  act(deliver(Agent, Resource, Demand, LocX)).
297 cost( board(Agent, Resource, LocX) , 2):- type(Agent, drone), act(board(
        ↪ Agent, Resource, LocX)).
298 cost( board(Agent, Resource, LocX) , 1):- type(Agent, autoBox), act(board
        ↪ (Agent, Resource, LocX)).
299
300
301
302 cumulDuration(Plan, Agent, 0,D):- plan(Plan, performs(E,0)), agent1(E,
        ↪ Agent) ,duration(E,D).
303 cumulDuration(Plan, Agent, T,D):-  plan(Plan, performs(E,T)), agent1(E,
        ↪ Agent) ,cumulDuration(Plan, Agent,T-1,D1), duration(E,D2), D = D1
        ↪ +D2, T>0.
304
305 duration1(Demand, Plan, S):- plan(Plan, performs( deliver(Agent, _,
        ↪ Demand, _), ArrivalTimeStep)), demand(Demand), cumulDuration(Plan
        ↪ , Agent,ArrivalTimeStep,S).
306
307
```

```
308
309 duration2(Demand, Plan, S):-
310   demand(Demand),
311   plan(Plan,_) ,
312   duration1(Demand, Plan, S).
313
314 duration2(Demand, Plan, 99):-
315   demand(Demand),
316   not duration1(Demand, Plan, _),
317   not plan(Plan, performs(deliver(_,_, Demand,_),_)),
318   plan(Plan,_).
319
320
321
322 cost1(Demand, Plan, S):-  S = #sum{
323     C : cost(E, C),
324     plan(Plan, performs(E,Tstep)),
325     plan(Plan, performs(board(_,Resource,_), StartTimeStep)),
326     plan(Plan, performs(deliver(_,Resource, Demand,_),ArrivalTimeStep)),
327     Tstep>= StartTimeStep, Tstep<= ArrivalTimeStep}, plan(Plan,performs(
      ↪ deliver(_,Resource, Demand,_),_)).
328
329
330 cost2(Demand, Plan, S):-
331   demand(Demand),
332   plan(Plan,_) ,
333   cost1(Demand, Plan, S).
334
335 cost2(Demand, Plan, 0):-
336   demand(Demand),
337   plan(Plan,_),
338   not cost1(Demand, Plan, _),
339   not plan(Plan, performs(deliver(_,_, Demand,_),_)).
340
341
342 overallCost(Plan, S):- S = #sum{C: cost2(_, Plan, C)}, plan(Plan,_).
343
344
345
346 breached(Plan, Demand, LocB):-
347   plan(Plan, performs(board(Agent, Resource, _),T1)),
348   demandedResource1(Demand, Resource),
349   plan(Plan, performs( deliver(Agent, Resource, Demand, _) ,T2)),
350   plan(Plan,performs(move(Agent,_,LocB),Ti)),
351   Ti>T1, Ti<T2,
352   type(Agent, AgentType), location(LocB),
353   prohibited(AgentType, LocB), agent(Agent).
354
355
356 nbBreachedLoc(Plan, Demand, N):- N = #count{ LocB:
```

```
  breached(Plan, Demand, LocB)
}, plan(Plan, performs( deliver(_,_, Demand, _) ,_)).



# %*
breached(Plan, LocB):-
  plan(Plan,performs(move(Agent,_,LocB),_)),
  type(Agent, AgentType), location(LocB),
  prohibited(AgentType, LocB), agent(Agent).

# % if a cost is asssigned to each prohibited location then "count" turns into "sum"
nbBreachedLoc(Plan, N):- N = #count{ LocB:
  breached(Plan, LocB)
}, plan(Plan,_).
*%



missed(Plan, Demand):-
  demand(Demand),
  not plan(Plan, performs(deliver(_,_, Demand,_),_)),
  plan(Plan,_).

missed(Plan, Demand):-
  demand(Demand),
  plan(Plan, performs(deliver(_,_, Demand,_),_)),
  duration2(Demand, Plan, D1),
  timeLimit(Demand, DL),
  D1>DL,
  plan(Plan,_).

nbMissedDemands(Plan, N):- N = #count{ Demand:
  missed(Plan, Demand)
  }, plan(Plan,_).



# %#show nbBreachedLoc/2.
#show nbBreachedLoc/3.
#show nbMissedDemands/2.

#show duration2/3.
#show overallCost/2.


"""

"""##Program: Auxiliary Features
```

```
406
407 ##Norms (Absolute + Relative)
408 """
409
410 # Commented out IPython magic to ensure Python compatibility.
411 normsStr="""
412 # % relative and absolute
413
414 absolute1(forbiddenZone(Demand)):- demand(Demand).
415 absolute1(missingDemand).
416
417
418 relative1(severity).
419 relative1(agePriority).
420 relative1(costPriority).
421
422
423
424 # % absolute1(forbiddenZone(Agent, Demand)):- assignedDemand(Demand,
        ↪ Agent).
425
426 """
427
428 # Commented out IPython magic to ensure Python compatibility.
429 getOrdersStr="""
430
431 # %====Absolute Norms
432 nbViolate(forbiddenZone(Demand), Plan, N):-
433   nbBreachedLoc(Plan, Demand, N).
434
435 nbViolate(missingDemand, Plan, N):-
436   nbMissedDemands(Plan, N).
437
438
439 pref(C, PlanA, PlanB):-
440   absolute1(C),
441   nbViolate(C, PlanA, N1),
442   nbViolate(C, PlanB, N2),
443   N1<=N2.
444
445
446 # %====Relative Norms
447 pref(costPriority, PlanA, PlanB):-
448   overallCost(PlanA, C1),
449   overallCost(PlanB, C2),
450   C1<=C2.
451
452
453 # % Predefined orders for severity
454 pr(high, medium).
```

```
pr(medium, low).
pr(high, low).
pre(severity, Demand1 ,Demand2):-
  severityLvl(Demand1, L1),
  severityLvl(Demand2, L2),
  pr(L1,L2).

# % Predefined orders for age category
pr(child, elderly).
pr(elderly, adult).
pr(child, adult).
pre(agePriority, Demand1 ,Demand2):-
  ageCategory(Demand1, A1),
  ageCategory(Demand2, A2),
  pr(A1,A2).


lexicoNorms(agePriority;severity).

priortize(Demand, Plan1, Plan2):-
    duration2(Demand, Plan1, D1),
    duration2(Demand, Plan2, D2 ),
    D1<=D2.


pref(Crit, Plan1, Plan2):- %lexicoPref(Crit, Plan1, Plan2).
  priortize(Demand, Plan1, Plan2),
  is_dominant(Crit, Demand, Plan1, Plan2).


is_dominant(Crit, Demand, Plan1, Plan2):-
    not is_dominated(Crit, Demand,  Plan1, Plan2),
    lexicoNorms(Crit),
    priortize(Demand, Plan1, Plan2).

is_dominated(Crit, Demand, Plan1, Plan2):-
    priortize(Demand, Plan1, Plan2),
    pre(Crit, Demand1, Demand),
    Demand1!= Demand,
    not priortize( Demand1, Plan1, Plan2).

is_dominated(Crit, Demand, Plan1, Plan2):-
    priortize(Demand, Plan1, Plan2),
    not pre(Crit, _, Demand),
    lexicoNorms(Crit),
    priortize(Demand, Plan2, Plan1).


#show pref/3.
#show nbViolate/3.
```

```
# %#show absolute1/1.

"""

"""##Aggregations"""

# Commented out IPython magic to ensure Python compatibility.
# evalStr="""
# alt(Plan):- plan(Plan,_).
#
#
# criteria(X):- absolute1(X).
# criteria(X):- relative1(X).
#
#
# absolute(Criteria):- absolute1(Criteria), not relaxed(Criteria).
# relative(Criteria):- relative1(Criteria).
# relative(Criteria):- relaxed(Criteria).
#
#
# pluralClass(C) :- belongs(A, C), belongs(B, C), class(C), A!=B.
#
# singleClass(C):- class(C), not pluralClass(C).
#
# superiorThan(X,Z):- superiorThan(X,Y), superiorThan(Y,Z).
#
#
# winner(A):- not loser(A), compliant(A).
# loser(A):- prefRelative(B,A), compliant(A), compliant(B), A!=B.
#
# compliant(A):- not nonCompliant(A), alt(A).
# nonCompliant(A):- criteria(C), absolute(C), nbViolate(C,A,N), alt(A), N
    ↪ >0.
#
# #show winner/1.
#
# %%========
#
# % Implementation of votng rule with dominant voters
# prefRelative(Alt1, Alt2):-
#     class(Class),
#     pVote(Class, Alt1, Alt2),
#     is_dominant( Class, Alt1, Alt2).
#
# is_dominant(Class, Alt1, Alt2):-
#     not is_dominated(Class, Alt1,Alt2),
#     class(Class),
#     pVote(Class, Alt1, Alt2).
#
# is_dominated( Class, Alt1,Alt2):-
```

```
#     class(Class),
#     pVote(Class, Alt1, Alt2),
#     superiorThan( Class1, Class),
#     Class1!=Class,
#     not pVote(Class1, Alt1, Alt2).
#
# is_dominated( Class, Alt1,Alt2):-
#     class(Class),
#     pVote(Class, Alt1, Alt2),
#     not superiorThan( _, Class),
#     pVote(Class, Alt2, Alt1).
#
#
# pVote( Class, Alt1, Alt2):-
#   singleClass(Class),
#   belongs(Criteria, Class),
#   pref(Criteria, Alt1, Alt2).
#
# % vote aggregation according to Copelands rule
# pVote( Class, Alt1, Alt2):-
#     pluralClass(Class),
#     copeland_score(  Class, Alt1, S1),
#     copeland_score(  Class, Alt2, S2),
#     S1>=S2.
#
# copeland_score(Class, Alt, S):-
#     S= #sum{ S1:
#         nb_pairwise_wins(  Class, Alt, Alt1, N1),
#         nb_pairwise_ties(  Class, Alt, Alt1, N2),
#         S1 = N1*2+N2,
#         alt(Alt1)},
#     pluralClass(Class), alt(Alt) .
#
# nb_pairwise_ties(  Class, Alt1, Alt2 , N):-
#     N= #count{ N1 :
#         nb_strict_voters( Class, Alt1, Alt2 , N1),
#         nb_strict_voters( Class, Alt2, Alt1 , N2),
#         belongs( Criteria, Class), N1=N2},
#     pluralClass(Class), alt(Alt1), alt(Alt2).
#
# nb_pairwise_wins(  Class, Alt1, Alt2, N):-
#     N= #count{ N1 :
#         nb_strict_voters( Class, Alt1, Alt2 , N1),
#         nb_strict_voters( Class, Alt2, Alt1 , N2),
#         belongs( Criteria, Class), N1>N2},
#     pluralClass(Class), alt(Alt1), alt(Alt2).
#
# nb_strict_voters( Class, Alt1, Alt2, N):-
#     N = #count{ Criteria :
#         pref(Criteria, Alt1, Alt2),
```

```
#         not pref(Criteria , Alt2, Alt1),
#         relative(Criteria),
#         belongs( Criteria, Class) },
#     pluralClass(Class), alt(Alt1), alt(Alt2).
#
#
#
#
# """

class deliverySystem:
  def __init__(self, mapInfo, deliveryAgentsInfo, resourcesInfo):
    self.mapInfo=mapInfo
    self.deliveryAgentsInfo=deliveryAgentsInfo
    self.resourcesInfo=resourcesInfo

  #auxiliary functions
  def aggregateAssignments(self, assignDict):
    aggDict = {}
    for k, v in assignDict:
      aggDict.setdefault(k, []).append(v)
    return aggDict

  def cartMultiplyLst(self, Lst1, Lst2):
    Lst=[]
    if Lst1 != []:
      for x in Lst1:
        for y in Lst2:
          Lst.append( x + y)
    else:
      Lst=Lst2
    return Lst


  def allocationAssignment(self, demandsInfo):
    demandAssignmentPrg = clingo.Control(["0"])
    demandAssignmentPrg.add('base', [], demandsInfo)
    demandAssignmentPrg.add('base', [], self.resourcesInfo)
    demandAssignmentPrg.add('base', [], self.deliveryAgentsInfo)
    demandAssignmentPrg.add('base', [], resourceAllocationStr)
    demandAssignmentPrg.add('base', [], demandAssignmentStr)
    demandAssignmentPrg.ground([("base", [])])
    assignments=demandAssignmentPrg.solve(yield_=True)

    self.assignmentsList=[]
    for assignmnt in assignments:
        usedAgents=[(str(atom.arguments[1]), str(atom)) for atom in
    ↪ assignmnt.symbols(shown=True) if atom.name == 'assignedDemand']
        uniqueAgents = self.aggregateAssignments(usedAgents)
        listAgent=[]
```

```
653         for k in uniqueAgents:
654           demandsPrgStr="\n {}".format(". \n ".join(uniqueAgents[k])) + "
    ↪ ."
655           listAgent.append(demandsPrgStr)
656         self.assignmentsList.append(listAgent)
657 
658 
659   def generatePlans(self, allocAssignList, demandsInfo):
660 
661     outPlans=[]
662     for assignment in allocAssignList:
663       assignmentPlans=[]
664       for agentTask in assignment:
665           maxTime=1
666           isUnSatisfiable= True
667           while (isUnSatisfiable and maxTime<20):
668             nbMaxTime="-c maxtime=" + str(maxTime)
669             prg2 = clingo.Control(["0", nbMaxTime])
670 
671             prg2.add('base', [], eventCalculusStr)
672             prg2.add('base', [], actionsStr)
673             prg2.add('base', [], outputStr)
674 
675             prg2.add('base', [], self.deliveryAgentsInfo)
676             prg2.add('base', [], demandsInfo)
677             prg2.add('base', [], self.resourcesInfo)
678             prg2.add('base', [], self.mapInfo)
679             prg2.add('base', [], agentTask)
680             prg2.ground([("base", [])])
681             out=prg2.solve(yield_=True)
682             #out=prg2.solve(on_model=lambda m: print("Answer: {}".format(
    ↪ m)))
683             agnetPlans=[]
684             for answer in out:
685               sorted_model = [str(atom) for atom in answer.symbols(shown=
    ↪ True)]
686               singlePlanLst=sorted_model
687               agnetPlans.append(singlePlanLst)
688             isSatisfiable=out.get().satisfiable
689             isUnSatisfiable = not isSatisfiable
690             maxTime=maxTime+1
691           #####
692           assignmentPlans=self.cartMultiplyLst(assignmentPlans,
    ↪ agnetPlans)
693       outPlans= outPlans + assignmentPlans
694 
695     # parsing all plans
696     self.plansStr=""
697     for i in range(len(outPlans)):
698       for j in outPlans[i]:
```

```
699         self.plansStr = self.plansStr + "plan(p{index}, {action}).".
    ↪ format(index=i+1, action=j)+"\n"
700       self.plansStr=self.plansStr+"\n"
701
702   def generateMixture(self, demandsInfo):
703     self.demandsInfoStr=demandsInfo
704     self.allocationAssignment(demandsInfo)
705     self.generatePlans(self.assignmentsList, demandsInfo)
706     print(self.plansStr)
707
708   #Evaluation
709   def getFeatures(self, inputPlans):
710     costTimePrg = clingo.Control(["0"])
711     costTimePrg.add('base', [], actionsStr)
712     costTimePrg.add('base', [], self.deliveryAgentsInfo)
713     costTimePrg.add('base', [], self.demandsInfoStr)
714     costTimePrg.add('base', [], self.resourcesInfo)
715     costTimePrg.add('base', [], self.mapInfo)
716     costTimePrg.add('base', [], inputPlans)
717     costTimePrg.add('base', [], gettingFeaturesStr)
718     costTimePrg.ground([("base", [])])
719     outCostTime=costTimePrg.solve(yield_=True)
720
721     sorted_model = [str(atom) for atom in outCostTime.model().symbols(
    ↪ shown=True)]
722     #print(" {}".format(".\n ".join(sorted_model))+ ".")
723     self.featuresStr=" {}".format(".\n ".join(sorted_model))+ "."
724
725   def normAssesment(self, featuresStr):
726     ordersPrg = clingo.Control(["0"])
727
728     ordersPrg.add('base', [], self.demandsInfoStr)
729     ordersPrg.add('base', [], featuresStr)
730     ordersPrg.add('base', [], normsStr)
731     ordersPrg.add('base', [], getOrdersStr)
732
733     ordersPrg.ground([("base", [])])
734
735     #outOrders=ordersPrg.solve(on_model=lambda m: print("Answer: {} ".
    ↪ format(m)))
736
737     outOrders=ordersPrg.solve(yield_=True)
738
739     sorted_model = [str(atom) for atom in outOrders.model().symbols(shown
    ↪ =True)]
740     #print(" {}".format(".\n ".join(sorted_model))+ ".")
741     self.assessments=" {}".format(".\n ".join(sorted_model))+ "."
742     #print(prefsStr)
743
744
```

```
  def aggregate(self, assessments, inputSettingStr):
    evalPrg = clingo.Control(["0"])

    evalPrg.add('base', [], self.demandsInfoStr)
    evalPrg.add('base', [], self.plansStr)
    evalPrg.add('base', [], evalStr)
    evalPrg.add('base', [], assessments)
    evalPrg.add('base', [], normsStr)
    evalPrg.add('base', [], inputSettingStr)


    evalPrg.ground([("base", [])])

    #outOrders=evalPrg.solve(on_model=lambda m: print("Answer: {} ".
    ↪ format(m)))

    outEvals=evalPrg.solve(yield_=True)

    sorted_model = [str(atom) for atom in outEvals.model().symbols(shown=
    ↪ True)]
    self.selectedPlan=" {}".format(".\n ".join(sorted_model))+ "."

  def evaluate(self, inputSettingStr):
    self.getFeatures(self.plansStr)
    self.normAssesment(self.featuresStr)
    self.aggregate(self.assessments, inputSettingStr)
    print(self.selectedPlan)

"""#Scenarios"""

#Making an Instance of the system
deliveySystemInstance = deliverySystem(mapStr, agentInfoStr,
    ↪ resourcesInfoStr)

"""##Compliance Scenario:

###Demand Information

<center>

| Demnd| Resource | Subject| Severity| Time limit| Location |
| -----| ---------|------- | --------|-----------|----------|
| d1   | r1       |Child   | high    | 20        | Node D   |
| d2   | r2       |Elderly | medium  | 30        | Node D   |

</center>
"""

demandsInfoStr1="""
demand(d1;d2).
```

```
792
793 demandedResource1(d1, r1).
794 demandedResource1(d2, r2).
795
796 ageCategory(d1, child).
797 ageCategory(d2, elderly).
798
799 severityLvl(d1, high).
800 severityLvl(d2, medium).
801
802
803 timeLimit(d1, 20).
804 timeLimit(d2, 30).
805
806 initially(demandAt(d1, locD)).
807 initially(demandAt(d2, locD)).
808
809 initially(active(Demand)):- assignedDemand(Demand,_).
810
811 """
812
813 """###Compliance Setting"""
814
815 # Commented out IPython magic to ensure Python compatibility.
816 #Compliance Setting:
817
818 #                Relative
    ↪ Absolute
819 # [costPriority]<[agePriority]<[severity]  ||   forbiddenZone(d1),
    ↪ forbiddenZone(d2), missedDemand
820
821
822 inputSettingStr="""
823 # %input setting
824 class( severityClass; agePriorityClass; costPriorityClass ).
825
826 belongs(severity, severityClass).
827 belongs(agePriority, agePriorityClass).
828 belongs(costPriority, costPriorityClass).
829
830 superiorThan(severityClass, agePriorityClass).
831 superiorThan(agePriorityClass, costPriorityClass).
832
833 """
834
835 deliveySystemInstance.generateMixture(demandsInfoStr1)
836
837 deliveySystemInstance.evaluate(inputSettingStr)
838
839 """##Dilemma Scenario 1:
```

```
### (Insufficient Resource)
insufficient resources -> we can not avoid missing a demand in any case
    ↪ -> so we relax missing demand

###Demand Information

<center>

| Demnd| Resource | Subject| Severity| Time limit| Location |
| -----| ---------|------- | --------|-----------|----------|
| d1   | r1       |Child   | high    | 20        | Node D   |
| d2   | r1       |adult   | medium  | 30        | Node D   |

</center>
"""

demandsInfoStr2="""
demand(d1;d2).

demandedResource1(d1, r1).
demandedResource1(d2, r1).

ageCategory(d1, child).
ageCategory(d2, adult).

severityLvl(d1, high).
severityLvl(d2, medium).


timeLimit(d1, 20).
timeLimit(d2, 30).


initially(demandAt(d1, locD)).
initially(demandAt(d2, locD)).

initially(active(Demand)):- assignedDemand(Demand,_).

"""

"""###Compliance Setting
(Relaxed)
"""

deliveySystemInstance.generateMixture(demandsInfoStr2)

# Evaluation with compliance setting
#Compliance Setting:

#                Relative
```

```
    ↪ Absolute
# [costPriority]<[agePriority]<[severity]  ||   forbiddenZone(d1),
    ↪ forbiddenZone(d2), missedDemand

deliveySystemInstance.evaluate(inputSettingStr)

# Commented out IPython magic to ensure Python compatibility.
# Relaxed Setting1:
# missedDemand Obligation is relaxed
#                Relative
    ↪  Absolute
# [costPriority]<[agePriority]<[severity] < [missedDemand] ||
    ↪ forbiddenZone(d1),forbiddenZone(d2)


inputSettingStr2="""
# % the input setting
relaxed(missingDemand).

class(missingDemandClass; severityClass; agePriorityClass;
    ↪ costPriorityClass ).

belongs(missingDemand, missingDemandClass).
belongs(severity, severityClass).
belongs(agePriority, agePriorityClass).
belongs(costPriority, costPriorityClass).


superiorThan(missingDemandClass, severityClass).
superiorThan(severityClass, agePriorityClass).
superiorThan(agePriorityClass, costPriorityClass).

"""

# Evaluation with relaxed setting
# Relaxed Setting1:
# missedDemand Obligation is relaxed
#                Relative
    ↪  Absolute
# [costPriority]<[agePriority]<[severity] < [missedDemand] ||
    ↪ forbiddenZone(d1),,forbiddenZone(d2)
deliveySystemInstance.evaluate(inputSettingStr2)

"""##Dilemma Scenario 2:
### (Time Limit)


There is no compliant which satisfy all demands or does not miss any
    ↪ demand

```

```
Relax forbidden Zone

###Demand Information

<center>

| Demnd| Resource | Subject| Severity| Time limit| Location |
| -----| ---------|------- | --------|-----------|----------|
| d1   | r1       |Child   | Medium    | 10        | Node D   |
| d2   | r2       |adult   | High  | 15        | Node D   |

</center>
"""

demandsInfoStr3="""
demand(d1;d2).

demandedResource1(d1, r1).
demandedResource1(d2, r2).

ageCategory(d1, child).
ageCategory(d2, adult).

severityLvl(d1, medium).
severityLvl(d2, high).


timeLimit(d1, 10).
timeLimit(d2, 15).


initially(demandAt(d1, locD)).
initially(demandAt(d2, locD)).

initially(active(Demand)):- assignedDemand(Demand,_).

"""

"""###Compliance Setting"""

deliveySystemInstance.generateMixture(demandsInfoStr3)

# Evaluation with compliance setting
#Compliance Setting:

#               Relative
    ↪ Absolute
# [costPriority]<[agePriority]<[severity]  ||   forbiddenZone(d1),
    ↪ forbiddenZone(d2), missedDemand

```

```
deliveySystemInstance.evaluate(inputSettingStr)

# Commented out IPython magic to ensure Python compatibility.
# Relaxed Setting2:
# missedDemand Obligation is relaxed
#                Relative
    ↪  Absolute
# [costPriority]<[agePriority]<[severity] < [forbiddenZone(d2)] ||
    ↪ forbiddenZone(d1), missedDemand


inputSettingStr3="""
# % the input setting
relaxed(forbiddenZone(d2)).

class(forbidZoneClass; severityClass; agePriorityClass; costPriorityClass
    ↪  ).

belongs(forbiddenZone(d1), forbidZoneClass).
belongs(severity, severityClass).
belongs(agePriority, agePriorityClass).
belongs(costPriority, costPriorityClass).


superiorThan(forbidZoneClass, severityClass).
superiorThan(severityClass, agePriorityClass).
superiorThan(agePriorityClass, costPriorityClass).

"""

# Evaluation with relaxed setting
# forbiddenZone(d2) Obligation is relaxed
#                Relative
    ↪  Absolute
# [costPriority]<[agePriority]<[severity] < [forbiddenZone(d2)] ||
    ↪ forbiddenZone(d1), missedDemand

deliveySystemInstance.evaluate(inputSettingStr3)

"""#Dilemma Scenario 3:
(Forbidden Zone as Legal)

Let's say forbidden zone is a legal norm or a hard absolute norm

I can never be realxed under any circumstances

In such a case there wouldn't be a compliant Plan(all plans conatin a
    ↪ missied demand)

We relax the missing demand to satisfy demands as much as possible
```

```
1023
1024 ###Demand Information
1025
1026 <center>
1027
1028 | Demnd| Resource | Subject| Severity| Time limit| Location |
1029 | -----| ---------|------- | --------|-----------|----------|
1030 | d1   | r1       |Child   | Medium    | 10        | Node D   |
1031 | d2   | r2       |adult   | High  | 15        | Node D   |
1032
1033 </center>
1034 """
1035
1036 demandsInfoStr4="""
1037 demand(d1;d2).
1038
1039 demandedResource1(d1, r1).
1040 demandedResource1(d2, r2).
1041
1042 ageCategory(d1, child).
1043 ageCategory(d2, adult).
1044
1045 severityLvl(d1, medium).
1046 severityLvl(d2, high).
1047
1048
1049 timeLimit(d1, 10).
1050 timeLimit(d2, 15).
1051
1052
1053 initially(demandAt(d1, locD)).
1054 initially(demandAt(d2, locD)).
1055
1056 initially(active(Demand)):- assignedDemand(Demand,_).
1057
1058 """
1059
1060 """###Compliance Setting"""
1061
1062 deliveySystemInstance.generateMixture(demandsInfoStr4)
1063
1064 # Evaluation with compliance setting
1065 #Compliance Setting:
1066
1067 #                Relative
      ↪ Absolute
1068 # [costPriority]<[agePriority]<[severity]  ||   forbiddenZone(d1),
      ↪ forbiddenZone(d2), missedDemand
1069
1070 deliveySystemInstance.evaluate(inputSettingStr)
```

```

# Commented out IPython magic to ensure Python compatibility.
# Relaxed Setting2:
# missedDemand Obligation is relaxed
#                Relative
    ↪  Absolute
# [costPriority]<[agePriority]<[severity] < [missedDemand] ||
    ↪ forbiddenZone(d1), forbiddenZone(d2)


inputSettingStr4="""
# % the input setting
relaxed(missingDemand).

class(missingDemandClass; severityClass; agePriorityClass;
    ↪ costPriorityClass ).

belongs(missingDemand, missingDemandClass).
belongs(severity, severityClass).
belongs(agePriority, agePriorityClass).
belongs(costPriority, costPriorityClass).


superiorThan(missingDemandClass, severityClass).
superiorThan(severityClass, agePriorityClass).
superiorThan(agePriorityClass, costPriorityClass).

"""

# Evaluation with relaxed setting
# Relaxed Setting1:
# missedDemand Obligation is relaxed
#                Relative
    ↪  Absolute
# [costPriority]<[agePriority]<[severity] < [missedDemand] ||
    ↪ forbiddenZone(d1),forbiddenZone(d2)
deliveySystemInstance.evaluate(inputSettingStr4)
```

Listing C.1: Health care delivery system implementation code