COMPARATIVE ANALYSIS OF LOGIC-BASED POLICY FRAMEWORKS

Michael Ellis

A THESIS

Presented to the Faculty of Miami University in partial
fulfillment of the requirements
for the degree of

Master of Science

Department of Computer Science & Software Engineering

The Graduate School
Miami University
Oxford, Ohio

2024

Dr. Daniela Inclezan, Advisor
Dr. Khodakhast Bibak, Reader
Dr. John Femiani, Reader

ABSTRACT

The increasing utilization of intelligent agents has led to a growing demand for policy frameworks to govern their behavior. AOPL (Authority Obligation Policy Language) stands as a well-established policy framework within the logic-based AI (Artificial Intelligence) community. This thesis evaluated the suitability and effectiveness of AOPL by comparing it against other prominent policy frameworks. The evaluation assessed their capacity to effectively express policies, including role-based policies, and their ability to analyze policies. The methodology comprised a series of literature policy scenarios designed to test the scalability and efficiency of the policy frameworks. Additionally, the AOPL framework was extended with missing features identified through com-parison with EPA (Expressive Policy Analysis) and other logic-based policy frameworks. The outcome of this research contributed to a better understanding of the strengths and limitations of these policy frameworks, providing valuable insights for the development of future policy-based intelligent agent systems.

Table of Contents

List of Tables

List of Figures

Dedication

I would like to dedicate this thesis to family and friends who kept me motivated till the end.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Daniela Inclezan, for her guidance, expertise, and unwavering support on this thesis. I would also like to extend my appreciation to my committee members, Dr. Khodakhast Bibak and Dr. John Femiani for agreeing to assist me and provide their insight into how I can make this thesis better.

Chapter 1

Introduction

Intelligent agents are software programs designed to make autonomous decisions and take actions based on their ever-changing environment and objectives. They have revolutionized numerous fields and industries with their ability to learn, adapt, and interact with their surroundings. An agent is said to exhibit intelligent behaviour if it consists of the following [1] :

- a mathematical model which allows it to map its environment
- its own goal(s) and knowledge of its limited capabilities (e.g., limited observations)
- its own algorithm for planning and determining how best to achieve said goal(s) given its limitations and capabilities.

While intelligent agents have demonstrated tremendous potential in enhancing efficiency, accuracy, and convenience, their increasing use has also raised concerns about privacy, security, and accountability. Therefore, the development and implementation of policies and regulations that govern the behavior of intelligent agents have become essential. This is to ensure that they operate in a manner that is ethical, transparent, and aligned with social and legal norms. One example mentioned by Meyer and Inclezan [2] explains this by saying, "we would want a self-driving car to not only take us to our destination but also do so while respecting the law and cultural conventions". Unfortunately, however, traditional approaches often fall short in capturing the complexity and nuance of policy rules and their interactions with the environment. As a result, policy enforcement can become error-prone and difficult to maintain. However, a logic-based approach [1] provides a promising alternative that enables the representation of and reasoning about policies in a declarative and modular way. By using logical rules and constraints, logic programming allows for the automatic generation of policy-compliant behaviors while minimizing the risk of conflicts or inconsistencies. Therefore, incorporating logic programming techniques into policy implementation can help address some of the most pressing challenges in this field. There have been many papers written by researchers in a attempt to solve this problem. This thesis focused on two particular papers, one written by Gelfond and Lobo [3] and one by Craven *et al* [4].

Gelfond and Lobo, distinguished researchers in logic programming, authored a significant paper titled "Authority and Obligation Policy Language" ($\mathcal{AOPL}$) [3]. In their work, they introduced the $\mathcal{AOPL}$ framework, which is based on Answer Set Programming ($\mathcal{ASP}$) and specifically tailored for encoding authorization and obligation policies for intelligent agents. This framework stands out for its syntax, which closely resembles natural language, thereby enhancing readability. Additionally, it allows for seamless translation into $\mathcal{ASP}$, the language commonly used for encoding domains for intelligent agents. "Expressive Policy Analysis" ($\mathcal{EPA}$) [4], published a year after

$\mathcal{AOPL}$, presents an alternative framework rooted in Event Calculus as opposed to $\mathcal{ASP}$. Unlike $\mathcal{AOPL}$, $\mathcal{EPA}$ offers a range of policy analysis tools, including the capability to manage conflicts of permissions and support roles. This stands in contrast to $\mathcal{AOPL}$, which lacks such functionalities. This disparity sets the stage for a comparative examination between not just those two frameworks but ten (10) other policy frameworks, shedding light on their respective strengths and limitations in the realm of policy enforcement.

1.1 Motivation

The development and implementation of policies and regulations that govern the behavior of intelligent agents have become essential to ensure that agents operate in a manner that is ethical, transparent, and aligned with social and legal norms. However, enforcing these policies can be challenging due to the complexity and nuance of policy rules and their interactions with the environment. Therefore, a robust policy framework that can provide policy analysis, and representing policies for different roles is necessary.

In this thesis, we compared the policy framework, Authority and Obligation Policy Language ($\mathcal{AOPL}$) against the Expressive Policy Analysis ($\mathcal{EPA}$) framework and other policy frameworks, with respect to their ability to provide policy analysis, and represent policies for different roles. This covered a substantial gap in the existing literature in terms of studying the relationship between different formalisms for policies.

The outcome of this research was a comparative study that evaluated the effectiveness and expressiveness of $\mathcal{AOPL}$. An additional outcome was an extension of $\mathcal{AOPL}$ with the capabilities to support roles.

1.2 Contributions

The contributions of this thesis were as follows:

1. We provided a thorough comparison of the capabilities of $\mathcal{AOPL}$ against $\mathcal{EPA}$ and other policy frameworks based on their capabilities to support:
    - Policy analysis
    - Representing policies for different roles
2. We proposed a method to extend $\mathcal{AOPL}$ with the capabilities to support roles.

Chapter 2

Background & Related Work

The objective of this chapter is to explore the necessary background information needed to understand the two logic-based languages: that underline the policy frameworks we will study: Answer Set Programming ($\mathcal{ASP}$) and *Event Calculus*. The section explores the policy language $\mathcal{AOPL}$, which is based on $\mathcal{ASP}$, and describes how policies are created under its rules. The definition of intelligent agents using $\mathcal{ASP}$ and *Event Calculus* for policy analysis and planning are also explored to give context on what these policies are being created for exactly.

This chapter is broken down into four (4) sections. Section 2.1 will explore the features and potential of intelligent agents in various domains. Section 2.2 will provide an overview of a programming paradigm based on $\mathcal{ASP}$. Section 2.3 will delve into the logic based approaches to representing and reasoning over agent policies. A Subsection 2.3.1 will include $\mathcal{AOPL}$ syntax and definitions and Subsection 2.3.2 will include Expressive Policy Analysis ($\mathcal{EPA}$) and its policy language *Event Calculus*. Finally, Section 2.4 will discuss work closely related to this thesis.

2.1 Intelligent Agents

Intelligent agents are systems that can perceive their environment and take actions to achieve their goals. They are often used in complex and dynamic environments where traditional rule-based systems may not be effective. The definition of intelligent agents has been discussed in many research papers, such as Wooldridge [5] and Russell and Norvig [6].

Wooldridge [5] defines an intelligent agent as an autonomous entity that observes through sensors and acts upon an environment using actuators. The agent is capable of generating actions that achieve its objectives, and it is able to adapt to changes in the environment over time. Wooldridge [5] also describes different types of intelligent agents, including reactive agents, deliberative agents, and hybrid agents. Reactive agents make decisions based on their current perception of the environment, while deliberative agents make decisions by reasoning about their goals and the possible consequences of their actions. Hybrid agents combine both reactive and deliberative approaches. Examples of intelligent agents described in the book include the Pac-Man game, where the Pac-Man agent must navigate a maze and avoid ghosts, and the Mars Rover, which uses intelligent agents to analyze data and make decisions about where to explore on the Martian surface.

Additionally, in the paper "Artificial Intelligence: A Modern Approach" by Russell and Norvig [6], they further classify agents based on their degree of autonomy and intelligence, ranging from simple reflex agents to agents with goal-based planning and learning capabilities. These classifications are broken down as follows:

- **Simple Reflex Agents**: These agents select actions based solely on the current percept. They do not have a memory of past percepts or actions.
- **Model-Based Reflex Agents**: These agents have a model of the environment, which they use to keep track of past percepts and actions. They use this model to select actions that are likely to achieve their goals.
- **Goal-Based Agents**: These agents have a set of goals that they try to achieve. They use their knowledge of the environment and their own capabilities to generate a plan of action to achieve their goals.
- **Utility-Based Agents**: These agents have a set of goals that are assigned a degree of desirability or ”utility.” They use their knowledge of the environment and their own capabilities to select actions that will maximize their expected utility.

Other research papers expand on the concept of intelligent agents by discussing various approaches to their design and implementation. For example, Jain and Singh [7] propose a framework for designing intelligent agents that includes four stages:

- sensing
- reasoning
- decision-making
- action execution

For instance, this is implemented in the Autonomous Agent Architecture (AAA) agent architecture [8] via the loop in Figure 2.1. More specifically, the loop consists of the following steps executed by the agent [8]:

1. The agent observes the world, explains observations, and updates its knowledge base.
2. The agent selects an appropriate goal.
3. The agent finds a sequence of actions (i.e., a plan) to achieve the goal.
4. The agent executes part of the plan, updates its knowledge base, and goes back to Step 1.

Overall, the concept of intelligent agents is an important area of research in artificial intelligence, with a wide range of applications in areas such as robotics, gaming, and automation. The definition and design of intelligent agents is important to understand when comparing policy frameworks which this thesis will focus on.

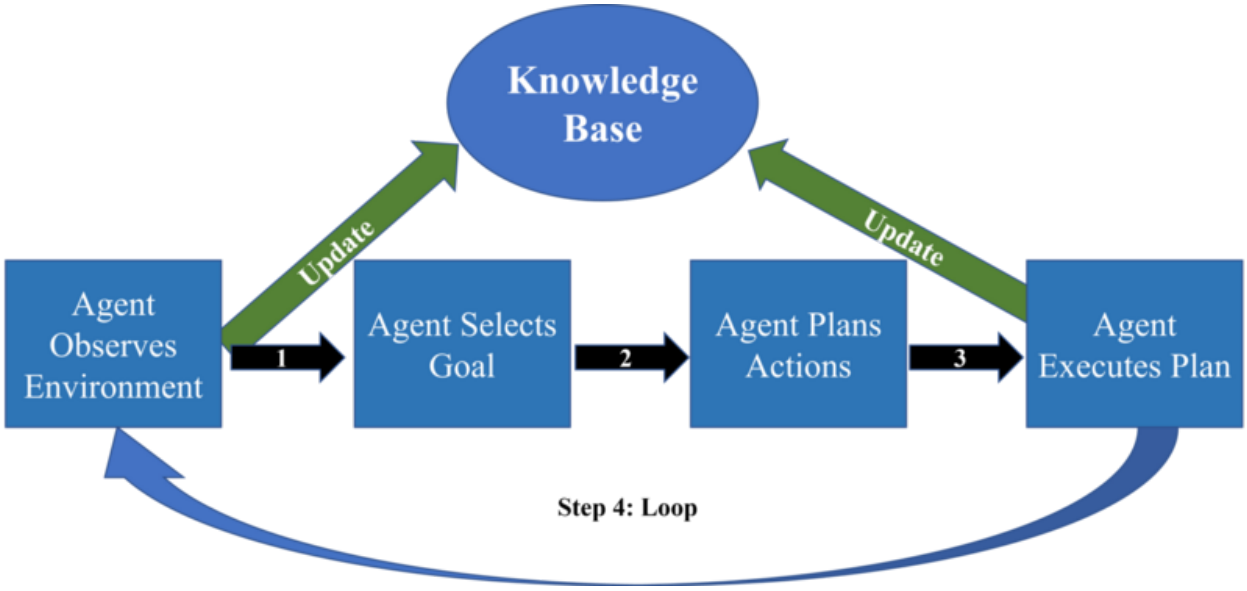

Figure 2.1: AAA agent architecture loop

2.2 Answer Set Programming

Answer Set Programming ($\mathcal{ASP}$) [9, 10] is a declarative programming paradigm introduced by Gelfond and Lifschitz. It has been extensively studied in the field of Artificial Intelligence (AI) for knowledge representation, reasoning, and problem solving [11]. $\mathcal{ASP}$ has emerged as a powerful tool for building intelligent agents that can reason about complex domains and exhibit human-like behavior [1]. $\mathcal{ASP}$ provides a high-level language for specifying the knowledge base of an agent, which can be queried and updated by the agent through logical inference and deduction. Below this thesis will briefly outline some syntax of $\mathcal{ASP}$ from basic concepts to the definition of an $\mathcal{ASP}$ program. For a more thorough understanding of $\mathcal{ASP}$, please refer to work by Calimeri et al. [12] and Gebser et al.[13].

$\mathcal{ASP}$ Syntax/Definitions

1. A *signature* $\Sigma =_{def} \langle O, F, P, V \rangle$ is a four-tuple comprised of the disjoint sets O, F, P, and V, where O contains objects, F contains functions, P contains predicates, and V contains variables [14].

2. A *term* over a signature Σ can be a variable, an object constant, or a function applied to one or more terms [14]. For instance, in the signature

   $\Sigma = \langle \{0, 1, 2\}, \{+, -, \times\}, \{, \geq, even\}, \{N\} \rangle$, the expression $(2 \times N + 1)$ is a term.

3. A *literal* is either an atom or the negation of an atom. For example, $p(x)$ and $\neg q(y)$ are literals.

4. A term or literal is called *ground* if it contains no variables or arithmetic functions [15]. For instance, $even(N)$ is not ground, but $even(0)$ and $\neg even(1)$ are ground, where uppercase letters denote variables.

5. A *rule* [15] is an implication with a head and a body, where the body is a conjunction of literals. The head is written on the left-hand side of the arrow, and the body is on the right-hand side. The head represents the conclusion of the rule, while the body represents its premises. The notation not represents default negation [15] which reads as "there is no reason to believe" or "there is no way infer". For example, the rule

$$\begin{array}{rcl} \neg father(X,Y) & \leftarrow & person(X),\ person(Y), \\ & & \text{not } father(X,Y) \end{array}$$

   says that if there is no reason to believe that X is the father of Y, then he is not.

6. A *fact* is a rule with an empty body. For example:

$$\text{even(0).}$$

7. A *constraint* is a special kind of rule that has an empty head, meaning that it only imposes a restriction on the body [15]. For example, the constraint

$$\leftarrow \text{married}(X,Y), \text{married}(X,Z), Y \neq Z$$

   says that X cannot be married to two different people, Y and Z.

8. A *program* is a pair consisting of a signature Σ and a set of rules Π over Σ [15]. For instance, given the previously seen signature $\Sigma = \langle \{0,1,2\}, \{+,-,\times\}, \{,\geq, even\}, \{N\} \rangle$,

$$\begin{array}{rcl} \Pi & = \{ & even(0), \\ & & even(N) \leftarrow even(N-2), N \geq 2 \ \} \end{array}$$

   is a program that defines the set of even numbers using arithmetic recursion

9. An answer set or model of a program Π is a collection of the consequences of Π under $\mathcal{ASP}$ semantics [15]. In other words, an answer set is a consistent set of ground literals that satisfies the rules of Π and is minimal, meaning there is no proper subset of the set that satisfies the rules of Π.

   There are two cases for determining the answer set of Π:

   Case 1: If Π consists only of rules without default negation, then an answer set of Π is a consistent set S of ground literals that satisfies the rules of Π and is minimal [14].

   Case 2: If Π is an arbitrary program, then let S be a set of ground literals. Let Π^S be a derivative program of Π (called the reduct of Π with respect to S) such that [14]:

- For all literals $l \in S$, remove all rules that contain **not** l.
- Remove all other clauses containing **not**.

S is an answer set of Π if S is an answer set of Π^S (as defined in Case 1).

Π can have multiple answer sets [15]. For example, the program [15]:

$$p \leftarrow \text{not } q.$$
$$q \leftarrow \text{not } p.$$

has two answer sets: $\{p\}$ and $\{q\}$.

In the case when the condition of a constraint is satisfied, Π will have zero answer sets (and will be called inconsistent or unsatisfied). For example, the program:

$$p.$$
$$\leftarrow p.$$

is inconsistent since the literals of the constraint are satisfied (i.e. p is true).

In addition to the two cases, $\mathcal{ASP}$ also supports choice rules in order to simplify the expression of alternatives for which multiple answer sets can be created.

10. A *choice rule*, also known as a constraint literal, is an expression that limits the number of atoms in a set [12, 15]. It is represented in the form of:

$$m \leq |\{l_1, l_2, \ldots, l_k\}| \leq n$$

or

$$m\{l_1, l_2, \ldots, l_k\}n$$

where m and n are arithmetic expressions and $l_1, l_2, \ldots, l_k$ are literals. The constraint literal is satisfied if the number of atoms in the set is within the range of m and n. To simplify the syntax, variables can be used to represent the constraint literal in the form of:

$$m \leq |\{l(A, B) : dom\_a(A), dom\_b(B)\}| \leq n$$

where dom_a and dom_b are predicate symbols that define the domains of variables A and B, respectively.

One of the main advantages of $\mathcal{ASP}$ is its ability to handle incomplete and uncertain knowledge, as well as to reason about conflicting and inconsistent information [16]. $\mathcal{ASP}$ has been successfully applied in a variety of domains, such as: planning [17, 2, 18, 19], natural language processing [20], natural language understanding [21, 22, 23, 24, 25], design pattern detection [26], or modeling scientific theories [27, 28].

$\mathcal{ASP}$ is based on the concept of answer sets, which are a set of interpretations of a logical program that satisfy certain conditions, such as consistency and minimality [16]. Answer set solvers, such as **Clingo** [29] and **DLV** [30], can efficiently compute the answer sets of a logical program, which can be used to reason about a domain and solve complex problems.

The Clingo[1] solver allows arithmetic expressions to be evaluated by external functions written in Python, Lua, C, Rust, or Haskell. For instance, the function

$$add(a : clingo.Symbol, b : clingo.Symbol) -> int$$

adds two symbols and returns an integer value. Although external functions can be used to delegate computations or queries, they should not be used to achieve side effects.

```
1 # script (python)
2 import clingo
3
4 def add (a : clingo . Symbol , b : clingo . Symbol ) -> int:
5     return a.number+ b.number
6
7 # end
8
9 a (@add (1,2)).
```

Figure 2.2: Clingo: Inline Functions

Figure 2.2 defines a Python function called "add" that takes two Clingo symbols as input and returns their sum as an integer. This function is then called in a Clingo program using the "@" syntax, as shown in line 9 where the result of calling add with arguments 1 and 2 is passed to the predicate "a". This is an example of how external functions can be used in Clingo programs to delegate computations or queries to traditional programming languages.

The program's output is seen in Figure 2.3

```
1 # a(3).
```

Figure 2.3: Clingo: Output for Inline Functions

In recent years, there has been a growing interest in $\mathcal{ASP}$ and its applications, which has led to

[1] https://potassco.org/clingo/

the development of new algorithms, tools, and techniques for ASP-based reasoning and problem solving, and more efficient solvers [1].

Representing Changing Environments in ASP

In $\mathcal{ASP}$, changing environments are represented through the use of *fluents* and *actions*. Fluents are functions whose value may change as a result of an action being performed [3]. Whereas, actions are atoms that can trigger the execution of update programs [3]. For example, consider a briefcase that automatically pops open when both clasps are up [1]. The signature of this briefcase domain consists of sort $clasp = \{1, 2\}$, inertial fluent $up(C)$ that holds **iff** (if and only if) clasp C is up, defined fluent $open$ that holds iff both claps are up, and action $toggle(C)$ that toggles clasp C [1]. The system description of our domain consists of the axioms below, which are written in a high-level logic-based language called $\mathcal{AL}_d$ [31] that translates into $\mathcal{ASP}$:

$$\begin{gathered} toggle(C) \ \textbf{causes} \ up(C) \ \textbf{if} \ \neg up(C) \\ toggle(C) \ \textbf{causes} \ \neg up(C) \ \textbf{if} \ up(C) \\ open \ \textbf{if} \ up(1), \ up(2) \end{gathered}$$

where C ranges over the sort $clasp$. Action languages are logic languages created specifically to enable correct and concise specifications of dynamic domains. Other action languages exist, for example modular action language ALM [32, 33].

The translation of this system description into a logic program for the Clingo solver is shown in Figure 2.4. This figure shows the definition of the sort $clasp$ and the fluents and actions in the domain. Note that the symbol "←" is replaced by ":-" in Clingo and all rules end with a period.

```
1 %% Domain Signature
2 clasp(1).
3 clasp(2).
4 fluent(inertial, up(C)) :- clasp(C).
5 fluent(defined, open).
6 action(toggle(C)) :- clasp(C).
```

Figure 2.4: $\mathcal{ASP}$: Domain Signature

Additionally, the axioms of the domain will be translated into a logic program for Clingo as shown in Figure 2.5.

The last law of the system description is a state constraint and as such we get the Clingo rule in Figure 2.6.

A visual representation of the system description explained above can be seen in the transition diagram in Figure 2.7 below.

```
7  #const n = 1.
8  step(0..n).
9
10  %% toggle(C) causes up(C) if -up(C)
11 holds(up(C),I+1) :- occurs(toggle(C),I), -holds(up(C),I), I < n.
12
13 %% toggle(C) causes -up(C) if up(C)
14 -holds(up(C),I+1) :- occurs(toggle(C),I), holds(up(C),I), I < n.
```

Figure 2.5: $\mathcal{ASP}$: Axioms

```
15 %% open if up(1), up(2).
16 holds(open,I) :- holds(up(1),I), holds(up(2),I).
```

Figure 2.6: $\mathcal{ASP}$: State Constraint

2.3 Logic-based Approaches to Representing and Reasoning over Agent Policies

When looking at intelligent agents, representing and reasoning about agent policies has been a central issue in multi-agent systems (**MAS**) research for many years. Policies are typically seen as plans of actions that an agent can use to achieve its objectives while rejecting certain rules or norms, and therefore they are essential for an agent's decision-making process. Logic-based approaches to representing and reasoning about agent policies have been gaining popularity due to their ability to provide a formal and rigorous framework for this task.

2.3.1 Authorization and Obligation Policy Language

Authorization and obligation policies are essential components of multi-agent systems (**MAS**) that aim to manage the interactions between agents with different roles and permissions. In this context, authorization policies specify who is allowed to perform a certain action, while obligation policies define what agents are required to do or not do under certain circumstances [3].

The study of authorization and obligation policies has gained attention in recent years due to its increasing relevance in the design of intelligent systems. Logic-based approaches have proven to be an effective tool for representing and reasoning about these policies, and have been used to develop formal models that capture the dynamics of authorization and obligation in **MAS**.

One influential work in this area which this thesis will be using as a basis of comparison is "Authorization and Obligation Policies in Dynamic Systems" by Michael Gelfond and Jorge Lobo [3]. This paper presents a framework for reasoning about authorization and obligation policies based on $\mathcal{AOPL}$. The authors define a language for expressing policies: Authorization and Obligation

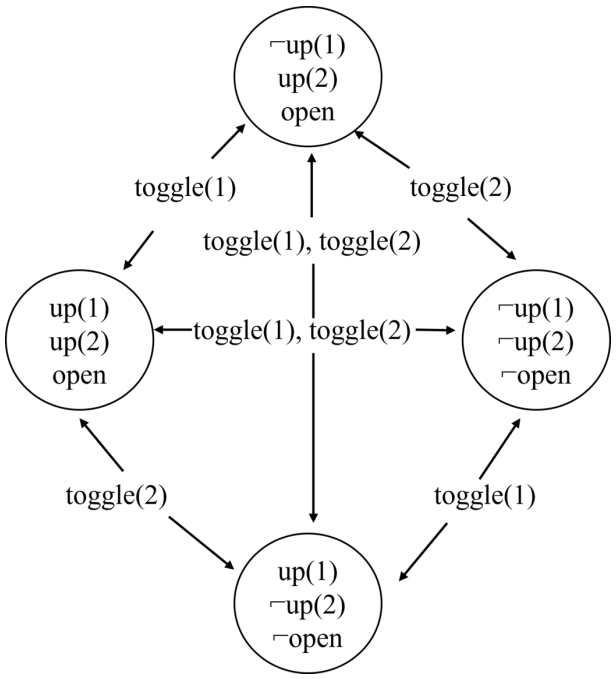

Figure 2.7: The Two Clasp Briefcase Domain

Policy Language. They also present a formal semantics for the language and develop a reasoning procedure that allows agents to reason about their obligations and permissions, both via a translation into $\mathcal{ASP}$.

Since its publication, the framework presented in this paper has inspired further research in the area of authorization and obligation policies. For instance, Sabri et al. [34] propose a formalism for specifying authorization policies of a dynamic system. In their paper, they utilize a temporal defeasible logic (**TDL**) to handle access control policies, including conflicts and timing constraints. They emphasize the importance of reaching plausible conclusions based on incomplete and uncertain information while considering the sensitivity and criticality of access control policies.

Abdali *et al.* [35], on the other hand, focus on developing a metamodel for hybrid access control policies. Their approach combines textual languages and visual models to create clear and formal specifications for access control. Their metamodel provides formal semantics, modularity, and support for concurrent application of multiple reusable access control models, enhancing the readability, clarity, and verification support of access control specifications.

$\mathcal{AOPL}$ Syntax/Definitions

In $\mathcal{AOPL}$, policies are expressed in terms of *facts* and *rules* [3]. Facts represent static information about the system, while rules define dynamic behavior of the system. $\mathcal{AOPL}$ includes the following syntax:

1. **Fluents**: atoms whose truth value may change over time [3]. For example, a fluent could represent the state of a door being open or closed, and could be updated as events occur.

2. **Actions**: actions are represented as atoms, and can trigger the execution of update programs [3]. For example, an action could represent a person opening a door, which would trigger an update program that changes the state of the door fluent to "open". Actions could be elementary (denoted by e below) or compound (i.e., sets of actions performed simultaneously).

3. **Rules**: rules define how fluents change over time [3], and are expressed using a combination of *normal logic programs* and *update programs* . Normal logic programs define the initial state of the system and the conditions that must hold for rules to apply. Update programs specify how the system changes over time in response to events.

4. **Agents**: $\mathcal{AOPL}$ models a system as a set of agents, each with its own set of actions.

5. **Authorization Policy**: Authorization policies specify what actions are allowed to be performed by agents [3]. The authorization policy is a set of rules of the form:

$$\begin{array}{rcl} permitted\,(e) & \textbf{if} & cond \\ \neg permitted\,(e) & \textbf{if} & cond \\ & & \\ d : \textbf{normally} & permitted(e) & \textbf{if } cond \\ d : \textbf{normally} & \neg permitted(e) & \textbf{if } cond \end{array} \tag{2.1}$$

   where e represents an agent action, $cond$ denotes a (possibly empty) collection of atoms from the signature formed by fluents and actions, excluding $prefer$ atoms (defined in point 7 below), and d serves as a rule label for a defeasible rule, which means it may have exceptions.

6. **Obligation Policy**: Obligation policies specify what actions must be performed by agents. The obligation policy is a set of rules of the form:

$$\begin{array}{rcl} obl\,(h) & \textbf{if} & cond \\ \neg obl\,(h) & \textbf{if} & cond \\ & & \\ d : \textbf{normally} & obl(h) & \textbf{if } cond \\ d : \textbf{normally} & \neg obl(h) & \textbf{if } cond \end{array} \tag{2.2}$$

   where h is a happening (i.e., an action or ites negation).

7. **Preference Statements**: Preference statements can be used to specify preferences over defeasible policy statements. A preference statement is a rule of the form:

$$prefer(d_1, d_2) \tag{2.3}$$

   where d_1 and d_2 are defeasible rule labels. This says that d_1 overrides d_2.

Example: Imagine these are the policies that need to be implemented into an agent [3]:

1. A military officer is not allowed to command a mission he/she authorized.

2. A colonel is allowed to command a mission he/she authorized.

3. A military observer can never authorize a mission.

First, we should break down each policy into actions and fluents [3].

$$\text{Actions: } authorize(C, M) \text{ and } assume\_command(C, M);$$
$$\text{Fluents: } authorized(C, M),$$
$$commands(C, M) \text{ and } observer(C),$$

where *M* and *C* are mission and commanders respectively.

Since the first two policy statements are contradictory of each other, they are referred to as defeasible and as such they are expressed as:

$$d_1(C, M) : \textbf{normally } \neg permitted(assume\_command(C, M))$$
$$\textbf{if } authorized(C, M)$$
$$d_2(C, M) : \textbf{normally } permitted(assume\_command(C, M))$$
$$\textbf{if } colonel(C)$$

The second statement seems to overide the first one, which we state as:

$$prefer(d_2(C, M), d_1(C, M))$$

Finally, the last policy, which seems strict, would be represented as:

$$\neg permitted(authorize(C, M)) \textbf{ if } observer(C)$$

On the other hand, the obligation policy is defined by Gelfond and Lobo [3] as actions which the agent must perform or abstain from performing. This is expressed very similarly to the aforementioned authority policy the only difference being that you are dealing with obligations of elementary actions. An example of an obligation policy is that an observer must report to its superiors.

$$obl\ (report(O, S))\ \textbf{if}\ observer\ (O)\ , supervisor\ (O, S)$$

In summary, $\mathcal{AOPL}$ is a powerful formalism for specifying authorization and obligation policies in dynamic systems. It is based on Answer Set Programming, and uses a version of $\mathcal{AL}_d$ [31] that includes strong negation and preference statements. The syntax of $\mathcal{AOPL}$ includes agents, actions, authorization policies, obligation policies, and preference statements, and can be used to model a wide range of dynamic systems and their policies.

2.3.2 Expressive Policy Analysis

Expressive Policy Analysis ($\mathcal{EPA}$) is an area of research that focuses on developing methods for analyzing and understanding complex policies. One important goal of this research is to improve the transparency and accountability of policy decisions, especially in the context of critical systems such as healthcare, finance, and transportation.

The paper that has made significant contributions to the field of expressive policy analysis is "Expressive Policy Analysis with Enhanced System Dynamicity" by Craven et al. [4]. This paper presents a novel approach to policy analysis based on a combination of Abductive Constraint Logic Programming ($\mathcal{ACLP}$) and *Event Calculus*. Expressive Policy Analysis with Enhanced System Dynamicity (**EPASD**) is a framework for representing, analyzing, and modifying policies for systems with dynamic properties [4]. The paper looks at policy analysis as the process of evaluating and assessing policies withing a specific domain or system. It involves the understanding of rules, regulations and guidelines that govern the behavior or decision-making processes of agents or entities within that system. Some issues that are present in policies that [4] set out to solve include:

1. **Modality Conflicts**: refers to a situation where an action is permitted but the agent is obligated not to perform it, or when an action is obligated but there is no authorization for that agent to perform the action.

2. **Coverage gaps**: refers to a situation where there exists no policy to dictate a correct response to which a request was made.

3. **Policy Comparison**: is the method of analysing one policy to determine whether the policy is included in, has been implied by or is equivalent to another policy.

The **EPASD** framework extends previous work on policy languages by incorporating temporal logic constructs and dynamic components, allowing for a more comprehensive representation of system policies [4].

EPASD policies are represented in a logic programming language called Extended Logic Programs with Time (**ELPT**), which encompasses $\mathcal{ACLP}$ and *Event Calulus*. This allows for the specification of policies with temporal components.

Example: *"A person cannot assist in a medical situation once he/she has taken part in surveying a contaminated area."* [4]

This is represented as follows:

$$
\begin{aligned}
&denied\left(Subs, M_1, assist, T\right) \leftarrow \\
&\quad do\left(Sub, M_2, assist, T'\right), T' < T, \\
&\quad holdsAt\left(activity\_type(M_1, medical), T\right), \\
&\quad holdsAt\left(activity\_type(M_2, survey(A)), T'\right), \\
&\quad holdsAt\left(area\_classify(A, contaminated), T'\right).
\end{aligned}
$$

where sub refers to the subject involved, M_1 represents the first medical entity, M_2 represents the second entity, T' denotes a specific time (earlier in this case) an action occurred and T denotes the current time at which the statement is being evaluated.

EPASD has been applied in various domains, including cloud computing, privacy policy analysis, and access control policy analysis, among others [4, 36]. For example, in a cloud computing scenario, **EPASD** was used to specify a policy for managing the allocation of computing resources based on user preferences and available resources. In a privacy policy analysis scenario, **EPASD** was used to specify a policy for managing the collection, use, and disclosure of personal information.

Event Calculus

Event Calculus as discussed by Craven *et. al* [4] is a logical framework used for reasoning about events and their effects in dynamic systems. It is one of the core methods used by Craven *et. al* in their novel approach to reason about changing properties of domains regulated by policies. It is particularly useful for representing and reasoning about temporal aspects of systems, including the temporal ordering of events and the effects of events on system properties. The Event Calculus was first introduced by Kowalski and Sergot in 1986 [37] and has since been applied in various domains, including robotics, natural language processing, and automated planning, among others.

In the context of policy analysis, the Event Calculus can be used to represent and reason about policies with temporal components. Policies can be expressed as rules that specify the conditions under which certain events should occur and the effects that those events should have on the system. For example, a policy for managing the allocation of computing resources in a cloud computing environment might include rules that specify the conditions under which resources should be allocated or deallocated based on user demand and system availability.

The syntax of the Event Calculus that differs from normal $\mathcal{ASP}$ syntax is based on first-order logic and includes the following constructs:

- **Initial Conditions**: These are assertions about the initial state of the system, such as the initial values of fluents.
- **Frame Axioms**: These are rules that define the effects of actions on fluents. For example, a frame axiom might specify that allocating a resource to a user causes the availability of that resource to decrease.
- **Completion Axioms**: These are rules that specify the conditions under which an action is considered complete. For example, an action to allocate a resource might be considered complete when the resource has been successfully allocated to a user.

The Event Calculus provides a powerful framework for representing and reasoning about policies with temporal components. However, it can be challenging to apply in practice, particularly for complex systems. To address this challenge, various extensions and refinements to the Event Calculus have been proposed, including the Fluent Calculus [38] and the ConGolog language [39], among others.

The relationship between Event Calculus and $\mathcal{ASP}$ has been previously explored and translations from Event Calculus into $\mathcal{ASP}$ have been proposed [40, 41].

2.4 Related Work

This thesis aims to provide a comparative analysis of two logic-based policy frameworks and propose methods for implementing items not covered in $\mathcal{AOPL}$. A survey of literature indicates that there have been numerous papers relating to $\mathcal{AOPL}$ and *Event Calculus* however, none of them delve into the comparison or methods of implementing and $\mathcal{EPA}$ functionality into $\mathcal{AOPL}$. The three most relevant to this paper are described in the following sections.

2.4.1 $\mathcal{APIA}$

Meyer and Inclezan [2] present an extension to the Architecture for Intentional Agents ($\mathcal{AIA}$) known as the Architecture for Policy-Aware Intentional Agents ($\mathcal{APIA}$). $\mathcal{APIA}$ addresses limitations found in $\mathcal{AIA}$, particularly its inability to determine whether actions performed by an external agent controller adhere to the domain's policies. To overcome this limitation, the authors introduce $\mathcal{AOPL}$, which allows $\mathcal{AIA}$ to reason over policies. The approach of $\mathcal{AOPL}$ [3] is to evaluate the history of actions from the perspective of the world, whereas $\mathcal{AIA}$ distinguishes between agent actions and exogenous actions. Furthermore, while both $\mathcal{AOPL}$ and $\mathcal{AIA}$ reason over past actions, $\mathcal{AIA}$ emphasizes planning for future actions to achieve a desired state.

However, due to the distinct approaches of $\mathcal{AOPL}$ and $\mathcal{AIA}$, seamless integration between the two is hindered, leading to interoperability challenges. To address these issues, Meyers and Inclezan propose modifications. They require that $\mathcal{AOPL}$ policies only describe agent actions, enforcing this rule through a constraint. Additionally, to resolve the second main problem, the authors adjust their policy compliance rules to ensure that only future actions impact the compliance of the world. These enhancements aim to improve the integration and compatibility between $\mathcal{AOPL}$ and $\mathcal{AIA}$.

2.4.2 DALICC

Pellegrini et al. [42] describe the Data Licenses Clearance Center (DALICC), a software framework designed to facilitate efficient and transparent resolution of licensing conflicts that arise during the reuse of digital assets. The framework automates rights clearance, aiding in the detection of licensing conflicts and reducing the cost of rights clearance for derivative works. The paper addresses issues such as high transaction costs in manual clearance of licensing terms, expertise required to detect compatibility conflicts between licenses, and negotiation and resolution of licensing conflicts between parties.

The DALICC framework consists of four main functional components: license library, license search, license composer, and license substitutability check. It leverages technologies such as Virtuoso triplestore, a Drupal-based web application, the PoolParty Semantic Suite, and the Clingo Answer Set Programming ($\mathcal{ASP}$) reasoner. Through a modeling process involving license analysis, vocabulary definition, mapping mechanisms, and a common model for annotating licenses, DALICC enables the creation of machine-readable license representations using ODRL policies and the DALICC vocabulary extension.

Reasoning over licenses is performed using $\mathcal{ASP}$, specifically the Clingo system, which enables conflict detection, identification of suitable licenses, and substitutability checks. The $\mathcal{ASP}$-based reasoning component ensures logical coherence, assesses compatibility, and supports conflict resolution between licenses. Overall, the comprehensive DALICC framework contributes to cost-efficient and transparent license management for digital assets.

2.4.3 Normative design using inductive learning

Corapi *et.al* [43] introduces an approach to normative design for multi-agent systems, specifically focusing on generating agent behavior that adheres to predefined norms. The authors address the limitations of manual normative design and propose an automated solution using inductive learning techniques.

The paper describes an iterative process that involves acquiring and refining normative knowledge from a set of training examples. They utilize an $\mathcal{ASP}$-based Inductive Logic Programming (**ILP**) system, Aleph, to generate a set of rules that capture the desired normative behavior. The proposed approach aims to improve system performance, reduce the need for manual intervention, and increase transparency by automating the design of normative behavior.

The authors evaluate their approach using a multi-agent simulation and demonstrate its effectiveness in generating behavior that conforms to the specified norms. They highlight the advantages of their method in terms of efficiency, scalability, and adaptability to different domains. The paper contributes to the field of multi-agent systems by providing a systematic and semi-automated approach to normative design, which can be applied to various real-world applications such as e-commerce, supply chain management, and intelligent transportation systems.

Chapter 3

Overview

In this thesis we aimed to answer the following research questions stemming from the work of Gelfond and Lobo on $\mathcal{AOPL}$ [3] and other work on logic-based policies, including Craven et al. on $\mathcal{EPA}$ [4]:

- **RQ1**: How do the $\mathcal{AOPL}$, $\mathcal{EPA}$, and other policy frameworks compare, especially in their ability to support (a) **policy analysis**; and (b) encoding policies for different **roles**?
- **RQ2**: What extensions of $\mathcal{AOPL}$ are required in order to increase $\mathcal{AOPL}$'s expressive power?
- **RQ3**: How could $\mathcal{AOPL}$ be extended to encode policies for different roles?

To answer these research questions we focused on the three tasks outlined in the sections below.

3.1 Task 1: Core Literature Set Creation

Task 1 encompassed a thorough and systematic approach to paper selection, combining targeted author searches, temporal constraints, and strategic expansions to related works. The resulting literature set creation served as a robust foundation for evaluating the capabilities of frameworks in the tasks that follow.

This task partially answered **RQ1** by procuring the research papers that would be used to produce a comparison task.

3.2 Task 2: Comparison and Analysis

Task 2, an in-depth comparison and analysis, served as a cornerstone in understanding the nuanced differences between $\mathcal{AOPL}$, $\mathcal{EPA}$, and other frameworks. The systematic exploration of criteria, the creation of a detailed comparison table, and the extraction of insightful conclusions contribute to a comprehensive evaluation. This process not only facilitates a nuanced understanding of the frameworks but also lays the groundwork for potential improvements in $\mathcal{AOPL}$'s capabilities.

This task answered **RQ1** and **RQ2** by utilizing the research papers gathered in Task 1 and curating a comparison table showcasing the differences and similarities between different policy frameworks in the papers.

3.3 Task 3: Roles in $\mathcal{AOPL}$

Task 3 focused on making $\mathcal{AOPL}$ more robust with the addition of roles to its framework. It aimed to give $\mathcal{AOPL}$ the ability to handle specific titles or positions, in its framework. Unlike before, where we evaluated existing frameworks, this task is about improving $\mathcal{AOPL}$ by adding features that deal with different roles in a system. We'll explore the differences between roles and group norms (shared expectations in a community) using a hypothetical university example. By doing this and looking at insights from selected papers, we want to understand how roles and group norms work together. This task aims to contribute to making $\mathcal{AOPL}$ more flexible and effective in guiding how systems behave in various situations.

This task answered **RQ3** by showcasing the proposed extension of $\mathcal{AOPL}$ with the capabilities of handling roles.

Chapter 4

Task 1: Core Literature Set Creation

Task 1 involved a meticulous process of literature selection and the subsequent creation of a literature set of policies and scenarios. The aim was to ensure a comprehensive evaluation of frameworks within realistic and diverse scenarios.

4.1 Literature Set Creation

The journey of paper selection commenced with the navigation of digital libraries, with a particular focus on the Digital Bibliography & Library Project (**DBLP**) platform. Our first step involved identifying the authors responsible for seminal papers crucial to our research – namely, Gelfond and Lobo on $\mathcal{AOPL}$ [3] and Craven et al. on $\mathcal{EPA}$ [4].

Subsequently, we initiated author-specific searches, scouring the digital landscape for papers authored by the identified individuals. To narrow our focus, we applied a stringent time frame, spanning from 2015 to 2023, to ensure the relevance of the identified literature to the contemporary research landscape.

Acknowledging the dynamic nature of academic exploration, we expanded our search strategy. In instances where these endeavors yielded no results, we strategically shifted our focus to papers citing the targeted works. This extension of our search scope allowed for a broader exploration of related literature, enriching our understanding and contextualizing our core literature set.

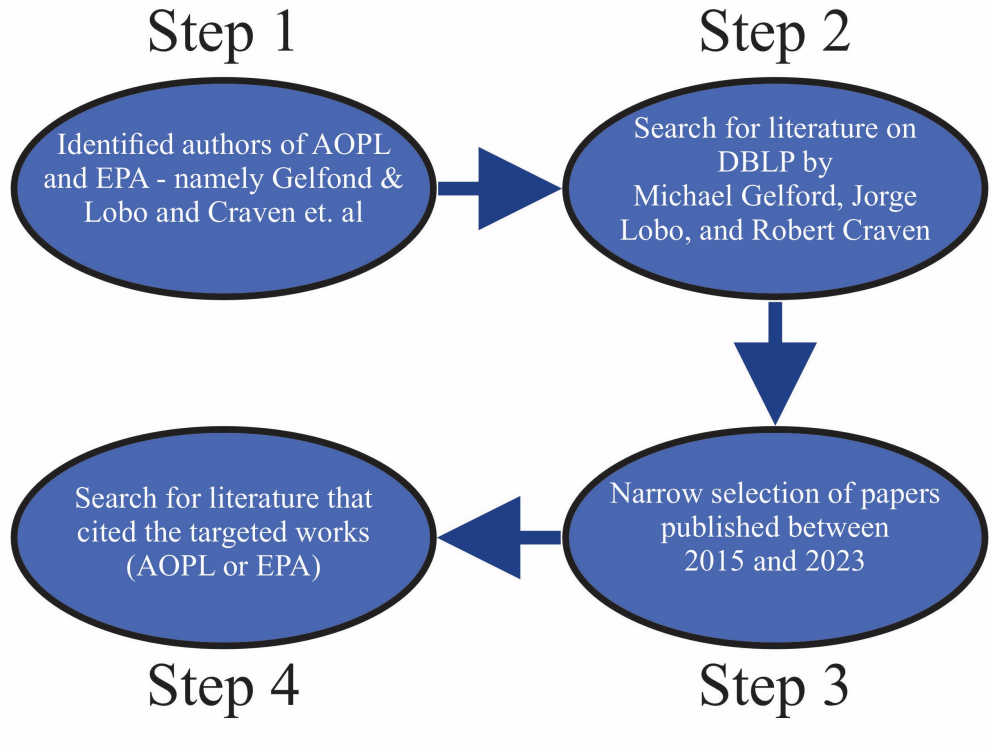

Figure 4.1: Figure Showcasing the Flowchart of Task 1

The outcome of these efforts was the compilation of a curated set of 10 papers meeting the stringent criteria of relevance, temporal alignment, and thematic congruence with our research goals. See Table 4.1 for more details. This carefully selected pool formed the bedrock for the creation of our core literature set, ensuring its richness and applicability to the subsequent tasks outlined in our methodology.

Table 4.1: Core Literature Set

	Title	Authors	Publication Date
1	Argumentation-Based Reasoning about Plans, Maintenance Goals, and Norms	Shams et. al.	Feb. 2020
2	Practical reasoning with norms for autonomous software agents	Shams et. al.	Sep. 2017
3	Automated multi-level governance compliance checking	King et. al.	Apr. 2017
4	ODRL Policy Modelling and Compliance Checking	De Vos et. al.	Aug. 2019
5	Simulating Normative Behaviour in Multi-agent Environments Using Monitoring Artefacts	Chang and Meneguzzi	Jul. 2016
6	CÒIR: Verifying Normative Specifications of Complex Systems	Gasparini et. al.	Jul. 2016
7	Reasoning with Group Norms in Software Agent Organizations	Aldewereld et. al.	Jul. 2016
8	Modeling and Detecting Norm Conflicts in Regulated Organizations	Jiang and Aldewereld	Jul. 2016
9	An ASP Framework for the Refinement of Authorization and Obligation Policies	Inclezan	Jul. 2023
10	LegalRuleML: XML-Based Rules and Norms	Palmirani et. al.	2011

4.2 Collection of Policy Examples

Starting from the *Core Literature Set*, we developed an additional *Collection of Policy Examples*. Incorporating examples taken from the ten (10) pivotal papers improved the robustness of our literature set to accurately test our extended $\mathcal{AOPL}$ framework. Below, we include some of the most illustrative examples from the *Collection of Policy Examples*:

1. Shams et al. (2020) [44]: Our benchmark incorporates normative reasoning frameworks applied in disaster recovery missions, emphasizing the dynamic nature of decision-making amidst conflicting goals and norms. Shams et al. consider the following goals:

   - *"(1) Running a hospital to help wounded people: fulfilled when, between time-steps 5 and 8, medics are present to offer help and they have access to water and medicines."*
   - *"(2) Organizing a survivors' camp: fulfilled when the camp area is secured and a shelter is built by timestep 15"*

   Norms that the agents must consider are as follows:

- *"(1) Prohibition against building shelters within 3 time units of detecting shocks."*
- *"(2) Obligation to halt water distribution for 2 time units upon detecting contamination."*
- *"(3) Prohibition against halting water distribution within 3 time units of building a shelter."*

2. Shams et al. (2017) [45]: Autonomous software agents, acting as decision support systems during emergencies, navigate complex normative landscapes, highlighting the necessity of norm-aware planning and decision-making mechanisms. The following goals are considered in this paper:

   - *"(1) Running a hospital to help wounded people: this goal is fulfilled when medics are present to offer help and they have access to water and medicines."*
   - *"(2) Organising a survivors' camp: this is fulfilled when the camp's area is secured and a shelter is built."*

   Norms that the agents must consider are as follows:

   - *"(1) Prohibition against building shelters within 3 time units of detecting shocks, with a cost of violation set at 5 units."*
   - *"(2) Obligation to halt water distribution for 2 time units upon detecting contamination, with a violation cost of 10 units."*

3. Aldewereld et al. (2015) [46]: Exploring group norms, this paper examines obligations like school attendance for children, and the associated roles and responsibilities of parents/-guardians.

   - *"Obligation for children under the age of 16 to attend school."*
   - *"Groups of more than 3 children are forbidden to be in a shop"*
   - *"The chairperson of a meeting is obliged to have the secretary circulating the minutes"*
   - *"Soldiers are forbidden to enter area (x, y)"*

4. Chang and Meneguzzi (2016) [47]: Norm-based simulations reveal the consequences of actions, such as penalties for theft, within complex environments governed by monitoring and enforcement roles. For instance take this illustrative scenario [47]:

   "The government of a fictional emerging nation started an immigration program to accelerate development through the hiring of foreigners. The country welcomes visitors, besides landed immigrants, to the country, since money from tourism greatly boosts the local economy. At the border, immigration officers must inspect immigrant passports. The foreigner acceptance policy is quite straightforward, and immigration agents must immediately accept immigrants with valid passports and no criminal records, and reject John Does and refugees outright. The government believes that the more immigrants it accepts, the better. Each

officer's responsibility is to accept as many immigrants as possible, while still following the guidelines that were passed to them. Each accepted able worker nets the officer 5 credits, which eventually turn into a bonus to the officer's salary. There are no rewards for rejecting immigrants. It becomes clear that the bonus each officer accumulates depends entirely on chance, and some officers may accumulate more than others, if at all. As such, some officers might feel inclined to accept immigrants they should not, only to add to their personal gain. To ensure officers act on the best interests of the nation only, the government introduced an enforcement system to the offices at the borders. Among the officers working in the immigration office, one is responsible for observing and recording the behaviour of those working in booths. This officer is known as the "monitor". His job is to write reports about what the officers do and send these reports to another officer, known as the "enforcer". The enforcer then reads the reports that are passed to him and look for any inconsistencies, such as the approval of an illegal immigrant. As this represents a violation of a rule, or norm, the enforcer then carries out an action to sanction the offending officer. The penalties for approving an illegal immigrant are the immediate loss of 10 credits and suspension of work activities for up to 10 s. Considering that immigrants arrive at a rate of 1 per 2 s, in a 10-s timespan 5 immigrants would have arrived at a given booth, meaning that a violating officer potentially loses 25 credits. Added to the other portion of the sanction, the potential loss rises up to 35 credits. The enforcement system, however, is not cost free. Each monitor and enforcer has an associated cost and it is within the interests of the nation to spend as little as possible with such a system. Therefore, the government wants to know how intensive the system must be to cover enough cases of disobedience so that officers will know violating norms is a disadvantage rather than an advantage."

There are two norms that can be extracted from this scenario, which are defined below [47]:

- *"All immigrants holding valid passports must be accepted. Failure to comply may result in the loss of 5 credits."*
- *"All immigrants holding passports that are not valid must not be accepted. Failure to comply may result in the loss of 10 credits and suspension from work activities for up to 10 time steps."*

5. Gasparini et al. (2016) [48]: Normative systems encompass collective imperatives and deadlines, as seen in scenarios involving coalitions of agents monitoring restricted areas.

   "Consider a coalition of agents of the sea-guard, consisting of a set of Unmanned aerial vehicles (UAVs), helicopters, and boats. Their goal is to monitor and intercept unauthorized boats trying to access a restricted area. The norms that guide the behaviour of the coalition are:

   - *(1) At any moment at least one member of the coalition must monitor the area. Moreover, we prefer having UAVs monitoring the area over helicopters. We assume that only helicopters and UAVs are capable of monitoring.*

- *(2) Whenever an unauthorized boat enters the area, a member of the coalition must intercept it before a certain deadline expires.*
- *(3) If no one intercepts the boat, then at least one member of the coalition must send a report to head-quarters before a certain deadline expires."*

6. Jiang and Aldewereld (2016) [49]: Conflicting norms within regulated organizations, exemplified by regulations from various institutions concerning actions like book returns and customs inspections, underscore the challenges of normative conflict resolution.

    - *"If a student borrows a book from the library, the student should return the book within 1 month"*
    - *"Students should return the book within 1 month after they borrow the book, otherwise they have to pay a fine within 1week"*
    - *"The World Customs Organization has defined a framework called the Authorized Economic Operator (AEO) program [50] in order to address the tensions created by the simultaneous growth in international trade and requirements for increased security. The European Communities' implementation of AEO permits various customs administrations to grant AEO certificates to qualified companies under which they enjoy special privileges. Taking the scenario of importing food from a country outside the EU to the Netherlands, a number of governmental authorities and companies are involved, which are governed by different sets of regulations concerning different aspects of the food importation process. For example, the EU has a set of general regulations, one of which specifies that the food authority is obliged to carry out a food quality inspection. With the introduction of the AEO programme, the Dutch government introduced new regulations for the specific domain of AEO-certified goods in order to improve trading efficiency. For example, one regulation specifies that a food authority is forbidden to carry out a food quality inspection, if the customs has already done so. Additionally, companies such as container terminals play an important role and bring their own regulations, e.g., a regulation at one container terminal is that carriers are obliged to transport their goods thence within two days of unloading. With different values and interests, the regulations from these institutions are likely to be inconsistent."*

Summary

This section presented a core literature set, and from that, a collection of policy examples intended to serve as qualitative tests. The aim was to measure the expressive power of the frameworks by examining how they respond to the complexities presented in these scenarios. Specifically, we were looking for how $\mathcal{AOPL}$ with the extension of roles, and other policy frameworks handle intricate rules and conditions, such as those depicted in the examples.

These examples illustrated a diverse array of policy scenarios, each highlighting distinct complexities and levels of realism. Some scenarios depicted real-world situations with intricate rules

and conditions, mirroring the complexities of policy enforcement in practical contexts. For instance, Shams et al. (2020) [44], introducing goals such as running hospitals and organizing survivor camps, alongside norms dictating actions like building shelters and halting water distribution under specific circumstances. Similarly, Gasparini et al. (2016) [48] delved into the monitoring of restricted areas by coalitions of agents, outlining imperatives regarding continuous monitoring and interception of unauthorized boats.

Conversely, other examples served as illustrative toy scenarios, emphasizing particular functionalities essential for policy languages/frameworks. For instance, Chang and Meneguzzi (2016) [47] presented a norm-based simulation involving immigration policies, elucidating penalties for unauthorized actions within a complex enforcement system. Jiang and Aldewereld (2016) [49] explored conflicting norms within regulated organizations, shedding light on challenges in normative conflict resolution through scenarios like book returns and customs inspections.

The comparison of these examples underscored the varying degrees of complexity and realism inherent in policy specification. While some scenarios faithfully mirrored real-world dilemmas, others served as simplified illustrations aimed at elucidating specific functionalities crucial for effective policy representation and enforcement within policy frameworks. This diversity in scenario complexity and realism provided valuable insights for evaluating the robustness and applicability of policy languages/frameworks across different contexts and requirements.

Chapter 5

Task 2: Comparison and Analysis

Task 2 involved a rigorous examination and comparison of the $\mathcal{AOPL}$ [3] with the $\mathcal{EPA}$ framework and other logic-based policy frameworks. The objective was to identify key criteria for evaluation, create a detailed comparison table, and draw insightful conclusions based on the strengths and weaknesses of each framework.

5.1 Methodology

Paper Selection Continuity:

Building upon the papers identified in Task 1, we maintained consistency in our approach. Leveraging the 10 carefully chosen papers, which encapsulated a rich array of policies and roles, we embarked on an insightful exploration into the realms of $\mathcal{AOPL}$ and $\mathcal{EPA}$.

Criterion Selection:

The criteria for the comparison table were carefully chosen to address gaps and opportunities for enhancement within $\mathcal{AOPL}$, aiming to evaluate existing features and envision its evolution into a more robust policy framework. These selected criteria explore the expressive power of policy statements, nuances in policy analysis, and role handling.

Criteria for evaluating policy frameworks were derived from our core literature set, identifying features that enhance expressive power and policy analysis capabilities. The finalized criteria were selected for their potential to improve the expressive and analytical capabilities of policy frameworks. These criteria include:

1. **Authorization Policies**: This criterion addressed the policy frameworks' ability to express policies that dictate the actions an agent is allowed to execute.

2. **Obligation Policies**: This criterion addressed the policy frameworks' ability to express policies that dictate the actions an agent is obligated to execute.

3. **Categoricity**: This criterion assessed the policy frameworks' ability to provide clarity and unambiguity of a policy statement. A policy is considered categorical when it leaves no room for interpretation or ambiguity, being unequivocal and clear beyond any reasonable doubt.

4. **Modality Conflicts**: This criterion assessed instances where conflicts arise between authorization and obligation policies within a given policy framework. In such cases, there is a conflict between an obligation policy, which dictates a certain action to be performed, and an authorization policy, which restricts the agent from executing that action.

5. **Roles**: This criterion assessed the policy frameworks' ability to assign specific sets of policies to particular roles or positions. Agents assigned to a particular role would only execute the policies associated with that role, ensuring a clear delineation of responsibilities and permissions

6. **Separation of Duty**: This criterion assesses the policy frameworks' capability to clearly define and manage roles, particularly in situations where roles may overlap. It addresses the potential ambiguity that arises when multiple roles have similar or overlapping responsibilities, and evaluates how the policy framework manages and resolves such situations.

7. **Temporal Restriction**: This criterion evaluated the policy frameworks' capacity to handle and enforce temporal policies governing agents' actions. It assesses whether the frameworks can manage and enforce policies that impose time constraints on permissions and obligations effectively.

8. **Penalties**: This criterion assessed the policy frameworks' ability to impose consequences on agents for failing to fulfill obligatory policies or for violating permissions enforced by authorization policies.

9. **Prohibitions**: This criterion evaluated the policy frameworks' ability to explicitly forbid an agent from executing a specific action.

10. **Real World Scenarios**: This criterion evaluated whether the paper utilized real world scenarios when showcasing how the policy framework handles specific domains.

11. **Group Norms**: This criterion evaluated the capability of the policy framework to group a set of policies and apply them to a group of agents who already have specific policies without conflicting with their individual policies.

12. **Enforcers**: This criterion assessed the capability of the policy framework to express a super user whose job is to manage the agents in the domain and ensure they are punished when they commit any violations.

13. **Storage of Non-Compliance**: This criterion assesses the capability of the policy framework to store instances of violation by an agent for further analysis.

14. **Nested Structure of Norms**: This criterion evaluates the capability of the policy framework to create a hierarchical structure of policies.

5.2 Comparison

Creation of the Comparison Table:

Our comparison table encapsulates insights from 12 papers: the ten (10) papers included in the *Research Paper Benchmark* from Task 1, plus the original papers on $\mathcal{AOPL}$ and $\mathcal{EPA}$. Each paper delineated strengths and weaknesses of $\mathcal{AOPL}$ as compared to other frameworks. Please refer to Table 5.1 for details.

In instances where a definitive *yes* or *no* answer was elusive, we introduced nuanced categories to enhance clarity:

- "Not explicitly" signifies a lack of strong representation, though inference can be drawn from other defined rules.
- "Implied" suggests that a criterion's end result is implied by the enforcement of certain rules, even if not explicitly stated.
- "Briefly acknowledges" indicates a passing mention without a detailed explanation.

Please refer to Table 5.2 for details. The classification process involved an in-depth assessment of the context provided in each paper, considering the framework's inherent capabilities and the implications of the criterion under scrutiny. This method ensured that each criterion was evaluated within its specific contextual framework, accounting for any implicit treatment or underlying assumptions.

For instance, in our examination of the paper "Authorization and Obligation Policies in Dynamic Systems" by Gelfond and Lobo [3], we looked at its treatment of roles within the $\mathcal{AOPL}$ framework. The paper briefly acknowledged the potential use of roles when comparing $\mathcal{AOPL}$ to other related work. Gelfond and Lobo outlined the basic structure of a typical policy in Role-Based Access Control (RBAC), which included object constants for users (referred to as subjects), their roles (such as job functions or titles), operations (e.g., read or write), and resources (e.g., files, disks, or databases). They further provided the syntax used to express roles in RBAC:

"*has_permission(R1, P)* **if** *has_permission(R2, P), R1* $\leq$ *R2*"

This syntax demonstrated how RBAC handles roles. Additionally, they highlighted the expression of a typical permission policy in $\mathcal{AOPL}$:

"*permitted(execute(S, O, D))* **if** *plays_role(S, R), has_ permission(R, O, D)*"

This syntax illustrates the translation of permission policies in $\mathcal{AOPL}$.

They then concluded by suggesting that the RBAC approach appeared to be a narrow special case of the methodology proposed in their paper for specifying and reasoning about policies.

While these excerpts suggest that $\mathcal{AOPL}$ showed promises for the capability to encode roles, they also imply that $\mathcal{AOPL}$ may offer more robust role-handling capabilities compared to traditional Role-Based Access Control (RBAC) methodologies. The mention of roles in $\mathcal{AOPL}$ serves primarily as a comparison to RBAC, indicating the potential breadth and flexibility of $\mathcal{AOPL}$'s methodology for specifying and reasoning about policies.

This classification as "briefly acknowledged" acknowledges the mention of roles within the paper but recognizes the lack of detailed exploration or explicit treatment of role-related functionalities within the $\mathcal{AOPL}$ framework. Further analysis may be required to fully assess the extent of role handling capabilities within $\mathcal{AOPL}$ and their implications for policy specification and reasoning.

After we applied this methodology across all 12 papers, we aimed to provide a comprehensive and insightful comparison between $\mathcal{AOPL}$ and the other selected papers. This approach enabled us to offer valuable insights into the nuanced treatment of criteria within policy frameworks, thereby contributing to a deeper understanding of their practical implications and potential areas for improvement.

Table 5.1: Comparison of Criteria in Selected Papers

Criterion	[3]	[4]	[44]	[45]	[51]	[52]	[47]	[48]	[46]	[49]	[53]	[54]
Authorization Policies	Y	Y	N	N	N	Y	Y	//	Y	N	Y	Y
Obligation Policies	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Categoricity	Y	N	N	N	N	N	N	N	N	Y	N	N
Modality Conflicts	N	Y	N	N	N	^	^	N	^	N	Y	*
Roles	^	Y	N	N	Y	N	Y	N	*	Y	N	*
Separation of Duty	N	Y	N	N	N	N	N	N	N	*	N	N
Temporal Restriction	*	Y	Y	Y	Y	N	N	Y	Y	*	Y	Y
Penalties	*	N	^	Y	Y	N	Y	*	N	Y	N	Y
Prohibitions	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Real World Scenarios	Y	*	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
Group Norms	*	*	N	N	*	N	N	Y	*	*	*	*
Enforcers	N	N	^	^	Y	N	Y	N	N	N	N	N
Storage of Non-Compliance	N	N	N	N	N	N	Y	Y	N	N	N	N
Nested Structure of Norms	*	N	Y	N	Y	N	N	N	*	Y	Y	Y
Code Repository	N	N	N	N	N	N	N	N	*	N	N	N

Summary

$\mathcal{AOPL}$ offers several advantages in representing obligations and authorizations within normative reasoning frameworks. Its use of defeasible rules and soft policy enforcement mechanisms provides remarkable flexibility and adaptability in navigating complex scenarios. This adaptability allows $\mathcal{AOPL}$ to effectively handle situations where policies may have exceptions or conditions that need to be taken into account. Additionally, its readability and writability serve as notable strengths,

Table 5.2: Key for Comparison Table

Symbol	Meaning
Y	Indicates that the criterion is directly addressed or supported by the framework.
N	Indicates that the criterion was not applicable or not mentioned in the paper
//	Implied
^	Briefly Acknowledges
*	Not explicitly

making it easily accessible to a wider audience and reducing the likelihood of errors during policy encoding. However, $\mathcal{AOPL}$ lacks certain features present in other frameworks, such as robust support for temporal constraints, modality conflicts, and explicit handling of roles. Enhancements in these areas could significantly increase $\mathcal{AOPL}$'s expressive power, making it more competitive in diverse application domains and enabling it to tackle a broader range of scenarios effectively.

Roles appear in a limited number of papers, with only three explicitly addressing this criterion. This indicates that roles are not extensively explored within the policy frameworks literature, suggesting a potential gap in research and development. However, nearly all papers address other fundamental criteria such as authorization policies, obligation policies, and prohibitions, indicating a strong focus on these aspects of policy specification. This disparity highlights the relative novelty and underemphasis of roles as a feature within the frameworks analyzed. Despite this, the inclusion of roles in a few papers underscores their importance in real-world scenarios and the need for further exploration and development in this area. By addressing this gap, researchers and developers can enhance the capabilities of policy frameworks like $\mathcal{AOPL}$ to better accommodate the complexities of role-based policies and improve their applicability in diverse contexts.

Chapter 6

Task 3: Extension of $\mathcal{AOPL}$ with Roles

In this chapter, we discussed our proposed extension of $\mathcal{AOPL}$ with capabilities for representing policies connected to different roles. We began by examining the significance of roles and their necessity. Next, we delved into the differences between roles and groups, followed by a summary of key role concepts we identified during Task 2 analysis. Finally, we introduced our innovative $\mathcal{AOPL}$ approach for incorporating roles.

6.1 Roles vs Groups Norms

Roles

Roles in policy frameworks are typically defined as sets of permissions or obligations assigned to specific agents in a particular domain.

In policy frameworks, roles serve as a way to organize and manage access to resources or actions within the domain. Each role is associated with a set of permissions or obligations that determine what actions the entity assigned to that role can perform.

Roles provide a mechanism for enforcing security, access control, and other policy requirements by defining clear boundaries and responsibilities for different agents within the domain. They help streamline policy management and enforcement by allowing policies to be applied consistently across different contexts and users.

The analysis conducted via our Comparison Table 5.1 revealed a notable gap in research concerning the incorporation of roles within policy frameworks. This identified gap prompted us to explore the feasibility of integrating roles into the $\mathcal{AOPL}$ framework. Notably, while $\mathcal{EPA}$ directly supported roles, $\mathcal{AOPL}$ did not, making it a targeted area for enhancement.

Group Norms

On the other hand, group norms are typically defined as sets of rules or standards that dictate acceptable behavior for a specific group of agents within a given context.

Group norms play a crucial role in regulating the behavior of agents within a domain, ensuring adherence to established standards and promoting consistency in actions. These norms outline the expected conduct, responsibilities, and interactions of members within the group.

By establishing clear guidelines and expectations, group norms facilitate cohesion, collaboration, and effective decision-making within the group. They provide a framework for maintaining order, resolving conflicts, and promoting a sense of unity and identity among group members.

Roles and Group Norms in a University

To discuss the difference between roles and groups, let us consider the context of a hypothetical university. In our hypothetical university scenario, roles are akin to specific positions or job titles assigned to individuals, each carrying a set of distinct responsibilities and expectations. For example, the role of a "Professor" encompasses teaching courses, conducting research, and providing academic leadership within their department. Roles establish a formal structure within the academic community, defining the professional scope and expectations associated with each position.

On the other hand, group norms transcend individual roles and encapsulate shared expectations and behaviors within the broader university community. Norms like punctuality and collaborative research practices guide the collective behavior of faculty, staff, and students, fostering a cohesive culture within the institution.

Additionally, a group is simply a collection of individuals, whereas what defines a role is the set the permissions and obligations associated with it. While groups may have associated norms, this is not necessary. Roles, on the other hand, do not exist independently of the rights and responsibilities that define them.

Illustrative Scenario

Consider a scenario where a Professor is responsible for teaching a course on "Database Management." As part of the Professor's role-specific responsibilities, the Professor is tasked with designing the course syllabus, delivering lectures, and assessing student performance through assignments and exams.

Additionally, within the university's academic community, there is a collaborative norm that encourages interdisciplinary research endeavors. Despite their specific roles as professors or students, members of the university community are encouraged to engage in joint research projects spanning various disciplines. This norm fosters an environment where individuals from diverse backgrounds can come together to address complex research questions and contribute to the advancement of knowledge.

In this scenario:

1. Professors fulfill their role-specific responsibilities by teaching courses and mentoring students.

2. Students enrolled in the course adhere to their role-specific responsibilities by submitting assignments, and actively participating in discussions.

6.2 Roles: Considerations from Related Work

Comparison and Analysis of Papers

In exploring the dynamics of roles and group norms across diverse research frameworks, subtle yet significant distinctions emerge, shedding light on the nuanced treatment and representation of these concepts.

Within policy frameworks like $\mathcal{EPA}$ by Craven et al. [4], roles assume central significance, intricately woven into the fabric of authorization and obligation policies. In contrast, research by King et al. [51] and Stephan Chang and Felipe Meneguzzi [47] positions roles within monitoring systems and simulation environments, elucidating their function as key agents responsible for capturing, transmitting, and managing information and actions within complex systems.

Gasparini et al.'s COIR [48] and Aldewereld et al.'s [46] exploration of group norms delve into theoretical underpinnings, offering insights into collective obligations and normative structures. Gasparini et al. [48] present a formalism for representing collective obligations, illuminating patterns and conditions necessary for fulfillment within intricate systems. Meanwhile, Aldewereld et al. [46] navigate the complexities of group norms, highlighting the imperative for agents to reason about their relation to norms and coordinate actions effectively.

Management of Conflicting Permissions

The analysis of the examined papers suggests that resolving conflicting permissions between roles for the same action involves nuanced considerations within systems.

In policy frameworks like $\mathcal{EPA}$ by Craven et al. [4], conflicts between permissions across roles may arise, requiring mechanisms to prioritize or resolve such conflicts. Similarly, in scenarios outlined by Aldewereld et al. [46], group norms may specify roles responsible for ensuring norm compliance, influencing precedence in conflicting permission scenarios.

Exploring examples from the papers underscores the importance of robust conflict resolution mechanisms and effective management of conflicting permissions to maintain system coherence and adherence to established norms and policies.

6.3 Implementation of $\mathcal{AOPL}$ with Roles

To incorporate roles into $\mathcal{AOPL}$, we utilized our university scenario, as described previously, to showcase our syntax.

Domain Representation

Policies, including role policies, depend on a particular domain that needs to be encoded prior to encoding its relevant policies. In this subsection, we show how the university domain (including pertient roles) would be encoded in $\mathcal{ASP}$. Note that $\mathcal{AOPL}$ is a language solely dedicated to the representation of policies. Thus, the underlying representation of the dynamic domain needs to be done another language, such as $\mathcal{ASP}$ or an action language.

Our first step in defining the university domain involved defining the sorts/types of objects of the domain and categorizing roles, namely professor, student, and enforcer:

Listing 6.1: ASP Code for Roles

```
person(X) :- enforcer(X).
role(professor).
role(student).
role(enforcer).
```

Notably, the enforcer role assumes a specialized responsibility for role assignment and unassignment within this domain.

Following this, we encoded factual data pertinent to students and professors, encompassing courses, assignments, and discussions, ensuring a comprehensive representation of the domain.

Listing 6.2: ASP Code for Person Definitions

```
person(abi).
person(sam).
person(steve).
```

Listing 6.3: ASP Code for Enforcer Definition

```
enforcer(mike).
```

Listing 6.4: ASP Code for Course Definition

```
course(maths).
course(english).
```

Listing 6.5: ASP Code for Assignments

```
assignment(a_m1, maths).
assignment(a_m2, maths).
assignment(a_m3, maths).
assignment(a_e1, english).
assignment(a_e2, english).
```

Listing 6.6: ASP Code for Discussions

```
discussion(d_e1, english).
discussion(d_e2, english).
```

Note that each course has a few assignments and discussion sessions defined (e.g., a_m1, a_m2, a_m3 and d_m1, d_m2, respectively, for the $maths$ course). These will be referenced in some of our policy rules below.

Next, we defined actions relevant to our domain and to roles. We started with the definitions of those actions that involve roles: $assign\_role(P0, R, P)$ and $unassign\_role(P0, R, P)$ –

Listing 6.7: ASP Code Defining Actions for Assigning and Unassigning Roles

```
action(assign_role(P0, R, P)) :- enforcer(P0),
                                 role(R), person(P).

action(unassign_role(P0, R, P)) :- enforcer(P0),
                                   role(R), person(P).
```

The above action syntax specifies that only an enforcer can assign or unassign roles. This forms the core mechanism for role implementation within $\mathcal{AOPL}$, offering potential for further expansions beyond the scope of Gelfond and Lobo's work [3].

Similarly, we defined possible actions associated with the $student$ role, such as enrolling in courses, submitting assignments, and participating in discussions.

Listing 6.8: ASP Code for Student Actions

```
action(enrolls(S, C)) :- person(S), course(C).

action(submits(S, A)) :- person(S), assignment(A).

action(participates(S, D)) :- person(S), discussion(D).
```

Likewise, actions linked to the professor role, like mentoring students and teaching courses, were specified.

Listing 6.9: ASP Code for Professor Actions

```
action(mentors(P, S)) :- person(P), person(S).

action(teaches(P, C)) :- person(P), course(C).
```

To track individuals' roles, we introduced the fluent *has_role*:

Listing 6.10: ASP Code for the Fluent has_role

```
fluent(inertial, has_role(P, R)) :- person(P), role(R).
```

Additionally, we indicated how this fluent changes as a result of role assignment or removal, i.e., as a result of the actions $assign\_role$ and $unassign\_role$.

Listing 6.11: ASP Code for the "holds" Predicate on Enforcer Actions

```
holds(has_role(P, R), I+1) :-occurs(assign_role(P0, R, P),I),
                              step(I).

-holds(has_role(P, R), I+1) :-occurs(unassign_role(P0, R, P),I),
                              step(I).
```

Similarly, we monitored students' actions, such as submitting assignments or participating in discussions, using fluents like *has_submitted*, *has_enrolled*, and *has_participated*.

Listing 6.12: ASP Code for "holds" Predicates on Student Actions

```
holds(has_submitted(S, A), I+1) :- occurs(submits(S, A), I),
                                   step(I).

holds(has_enrolled(S, C), I+1) :- occurs(enrolls(S, C), I),
                                  step(I).

holds(has_participated(S,D),I+1) :-occurs(participates(S,D),I),
                                   step(I).
```

We monitored whether professors had mentored students or taught courses, employing the fluents *has_taught* and *has_mentored* respectively.

Listing 6.13: ASP Code for "holds" Predicates on Professor Actions

```
holds(has_taught(P, C), I+1) :- occurs(teaches(P, C), I),
                                step(I).

holds(has_mentored(P, S), I+1) :- occurs(mentors(P, S), I),
                                  step(I).
```

We then added constraints that ensure that a person cannot be assigned the same role or enroll in the same class twice.

Listing 6.14: ASP Code for Executability Constraints

```
-occurs(assign_role(P0, P, R), I) :- holds(has_role(P, R), I),
                                     step(I).

-occurs(enrolls(S, C), I) :- holds(has_enrolled(S, C), I),
                             step(I).
```

Lastly, the prohibitions shown below ensure that no actions associated with a particular role that a person does not have can be executed. This ensures that for instance only a student can perform student actions and only professors can perform professor actions ensuring a level of separation of duty [55].

Listing 6.15: ASP Code for Executability Constraints (Continued)

```
-occurs(teaches(P, C), I) :- -holds(has_role(P, professor), I),
                              person(P), course(C),
                              step(I).

-occurs(mentors(P, S), I) :- -holds(has_role(P, professor), I),
                             -holds(has_role(S, student), I),
                              person(P), person(S),
                              step(I).

-occurs(enrolls(S, C), I) :- -holds(has_role(S, student), I),
                              person(P), course(C),
                              step(I).

-occurs(submits(S, A), I) :- -holds(has_role(S, student), I),
                              person(S), assignment(A, C),
                              step(I).
```

```
-occurs(participates(S, D), I) :- -holds(has_role(S,student),I),
                                  person(S), discussion(D, C),
                                  step(I).
```

$\mathcal{AOPL}$ Syntax for Roles

Following the domain encoding, we proceeded to encode the policies associated with our roles, namely professors and students in the policy-specification language of $\mathcal{AOPL}$. For a detailed explanation of $\mathcal{AOPL}$ Syntax, please refer to the background section of our thesis 2.3.1.

In our approach, we introduced an extension to $\mathcal{AOPL}$ utilizing the **has_role** fluent. This extension empowered the encoding of policies linked to particular roles, thereby enriching the expressive capacity of $\mathcal{AOPL}$. Moreover, it facilitated a more holistic methodology for incorporating role-based policies into the framework.

The policies associated with the role of professor was encoded with our proposed extension of $\mathcal{AOPL}$ follows:

normally $obl(teaches(P, C))$ **if** $has\_role(P, professor), has\_course(P, C).$

normally $obl(mentor(P, S))$ **if** $has\_role(P, professor), has\_role(S, student),$ $has\_course(P, C), has\_enrolled(S, C).$

These rules stipulate that a professor is obliged to teach a course if they hold the role of professor and are assigned the course *C*; similarly, a professor is obliged to mentor a student if they possess the role of professor, person *S* possess the role of student, and *S* is enrolled in one of *C*'s courses. Additionally, as a result of including the key word **normally** we accounted for exceptional circumstances:

$\neg obl(mentor(P, S))$ **if** $family\_emergency(P).$

Here, *family_emergency(P)* denotes a condition under which a professor *P* is exempted from mentoring a student due to a family emergency.

Additionally, for the role of student, the syntax would be as follows:

normally $obl(submits(S, A))$ **if** $has\_role(S, student), has\_enrolled(S, C),$ $assignment(A, C).$

normally $obl(participates(S, D))$ **if** $has\_role(S, student), has\_enrolled(S, C),$ $discussion(D, C).$

where $assignment(A, C)$ means that A is an assignment for course C, and $discussion(D, C)$ is a discussion session for course C.

These rules indicate that all actions are normally obligatory for students executing them, with exemptions considered similarly to those outlined for professors.

From an implementation perspective, it's important to highlight that the translation of $\mathcal{AOPL}$ policies into $\mathcal{ASP}$ remains unchanged even with the extension of the **has_role** fluent. This is because the **has_role** fluent, while new to $\mathcal{AOPL}$, can already be represented in $\mathcal{ASP}$. The process involves translating $\mathcal{AOPL}$ into $\mathcal{ASP}$ and integrating the $\mathcal{ASP}$ representation of the domain alongside specific compliance rules. This resultant $\mathcal{ASP}$ program is then executed using a solver such as **Clingo** [29].[1] Incorporating the new choice rules syntax in $\mathcal{AOPL}$ does not complicate the existing translation process, as choice rules are already inherent in $\mathcal{ASP}$.

Challenges with the $\mathcal{AOPL}$ Extension for Roles

In our university scenario, students are required to enroll in a number of courses. We would like to express this requirement as:

normally $obl(enrolls(S, C))$ **if** $has\_role(S, student)$.

However, the rule above imposes a constraint where a student must enroll in all available courses, which may not align with our desired behavior. The syntax of $\mathcal{AOPL}$ does not inherently support the expression of such nuanced rules. To address this limitation, we proposed introducing a custom syntax inspired by choice rules in $\mathcal{ASP}$, although not directly available in $\mathcal{AOPL}$. The proposed syntax is as follows:

normally $1\{obl(enrolls(S, C)) : course(C)\}3$ **if** $has\_role(S, student)$.

This expression signifies that a student is typically obligated to enroll in one to three courses from the set of available courses. The minimum and maximum number of courses that a student has to enroll in can be adjusted by modifying the numbers outside of the curly braces. Omitting one of these numbers will signify that there is no restriction in that direction. For example, the policy:

normally $1\{obl(enrolls(S, C)) : course(C)\}$ **if** $has\_role(S, student)$.

indicates that each student must enroll in at least one course, but there is no maximum limit on the number of courses a student can be enrolled in.

By introducing this syntax, we can tailor the rule to better suit the requirements of our university scenario. Additionally, the syntax allows for the specification of both minimum and maximum constraints on the number of courses a student can enroll in, providing flexibility in policy formulation.

[1] `https://potassco.org/clasp/`

This increased flexibility enhances the expressive power of $\mathcal{AOPL}$, enabling the representation of more complex policies. However, it is essential to note that this enhanced expressiveness comes at the cost of reduced readability and writability of the syntax. Understanding the syntax for choice rules is now necessary for comprehending policies fully.

On the implementation side, note that policy compliance with respect to $\mathcal{AOPL}$ policies is determined by translating $\mathcal{AOPL}$ into $\mathcal{ASP}$ and adding the $\mathcal{ASP}$ representation of the domain, as well as specific rules for determining compliance. The resulting $\mathcal{ASP}$ program is run using a solver such as **Clingo** [29].[2] Adding the new syntax for choice rules in $\mathcal{AOPL}$ would not complicate the existing translation of policies into $\mathcal{ASP}$ (defined by Gelfond and Lobo) as choice rules are already present in $\mathcal{ASP}$.

In summary, the additions made to represent roles are as follows:

- We introduced recommendations regarding sorts, objects, fluents, and actions to use in the formalization of roles. These could be seen as *library concepts* and could be represented as a library module in a high-level logic-based language (e.g. $\mathcal{ALM}$ [32, 33]). In particular, we introduced the sort $role$, the special object $enforcer$, the fluent has_role, and actions $assign\_role$ and $unassign_role$.
- We expanded the syntax of $\mathcal{AOPL}$ by introducing choice atoms borrowed from $\mathcal{ASP}$.

[2] `https://potassco.org/clasp/`

Chapter 7

Validation

The validation process aims to assess the proposed methodology for encoding roles in $\mathcal{AOPL}$, including the introduced syntax additions. This section evaluates the expressiveness (i.e., what can be expressed; how many different types of scenarios can be expressed/represented) and usability of the methodology (i.e., readability and writability), crucial factors in knowledge representation frameworks.

7.1 Experimental Design

The experimental design for validating the proposed methodology involves a systematic selection of scenarios from Task 1 based on specific criteria. Scenarios were chosen if they explicitly mentioned "roles" or presented situations where roles played a significant part in the decision-making process. The criteria for inclusion ensured a diverse representation of role-related scenarios to comprehensively evaluate the proposed methodology.

Selected Scenarios:

1. Shams et al. (2020) [44]: Our benchmark incorporates normative reasoning frameworks applied in disaster recovery missions, emphasizing the dynamic nature of decision-making amidst conflicting goals and norms.

    - *"(1) Running a hospital to help wounded people: fulfilled when, between time-steps 3 5 and 8, medics are present to offer help and they have access to water and medicines."*
    - *"(2) Organizing a survivors' camp: fulfilled when the camp area is secured and a shelter is built by timestep 15"*

    Norms to impose that the agents must consider are as follows:

    - *"(1) Prohibition against building shelters within 3 time units of detecting shocks."*
    - *"(2) Obligation to halt water distribution for 2 time units upon detecting contamination."*
    - *"(3) Prohibition against halting water distribution within 3 time units of building a shelter."*

2. Aldewereld et al. (2015) [46]: Exploring group norms, we examine obligations like school attendance for children and the associated roles and responsibilities of parents/guardians.

- *"Obligation for children under the age of 16 to attend school."*
- *"Groups of more than 3 children are forbidden to be in a shop"*
- *"The chairperson of a meeting is obliged to have the secretary circulating the minutes"*
- *"Soldiers are forbidden to enter area (x, y)"*

3. Chang and Meneguzzi (2016) [47]: Norm-based simulations reveal the consequences of actions, such as penalties for theft, within complex environments governed by monitoring and enforcement roles.

   There are two norms that can be extracted from this scenario, which are defined below [47]:

   - *"All immigrants holding valid passports must be accepted. Failure to comply may result in the loss of 5 credits."*
   - *"All immigrants holding passports that are not valid must not be accepted. Failure to comply may result in the loss of 10 credits and suspension from work activities for up to 10 time steps."*

7.2 Results

The provided scenarios from above were encoded using $\mathcal{AOPL}$ and the results are as follows:

1. Shams et al. (2020) [44]:

   a) *" Prohibition against building shelters within 3 time units of detecting shocks."*

   b) *" Obligation to halt water distribution for 2 time units upon detecting contamination."*

   c) *" Prohibition against halting water distribution within 3 time units of building a shelter."*

   We first decided what objects, roles, and variables we will be using: objects $shelters$, $water$, $contamination$, $camp$, and $shocks$; relevant role $organizer$; and variables T (representing the timestep) and P (representing a person).

   The fluents and actions would be as follows:

   - actions: $to\_build(P, shelters, T)$ and $to\_halt(P, water, T)$
   - fluents: $has\_role(P, organizer)$, $has\_secured(P, camp)$, $has\_built(P, shelters, T)$, $has\_detected(P, shocks, T)$ and $has\_detected(P, contamination, T)$.

   The first policy of the scenario, (a), was encoded as follows:

normally $\neg permitted(to\_build(P, shelters, T1))$
if $has\_role(P, organizer)$,
$has\_detected(P, shocks, T)$,
$T1 >= T$,
$T1 <= T + 3$.

The second policy of the scenario, (b), was encoded as follows:

normally $obl(to\_halt(P, water, T1))$ **if** $has\_role(P, organizer)$,
$has\_detected(P, contamination, T)$,
$T1 >= T$,
$T1 <= T + 2$.

The last policy of the scenario, (c), was encoded as follows:

normally $\neg permitted(to\_halt(P, water, T1))$ **if** $has\_role(P, organizer)$,
$has\_built(P, shelters, T)$,
$T1 >= T$,
$T1 <= T + 3$.

2. Aldewereld et al. (2015) [46]:

   a) *"Obligation for children under the age of 16 to attend school."*

   We first defined a sort age and the role of a $child$. We used variables P to refer to a person and A for an age. The fluents and actions would be as follows:

   - action: $to\_attend(P, school)$;
   - fluents: $has\_role(P, child)$ and $has\_age(P, A)$.

   The obligation of policy of the scenario would be encoded as follows:

normally $obl(to\_attend(P, school))$ **if** $has\_role(P, child)$,
$has\_age(P, A)$,
$A < 16$.

   b) *"Groups of more than 3 children are forbidden to be in a shop"*

   We additionally defined a sort $group$, as compared to the previous policy. The fluents and actions would be as follows:

   - action: $to\_enter(G, shop)$ – a group of people G enters a shop;

– fluent: $member(P, G)$ – person P is a member of group G.

The authorization policy of the scenario would be encoded as follows:

normally $\neg permitted(to\_enter(G, shop))$ **if** $member(P1, G)$, $member(P2, G)$, $member(P3, G)$, $member(P4, G)$, $P1! = P2, P1! = P3$, $P1! = P4, P2! = P3$, $P2! = P4, P3! = P4$.

This syntax is very long and not very efficient. Ideally, $\mathcal{AOPL}$ would include aggregate constructs like the $count$, which are present in $\mathcal{ASP}$. Then the rule would be written much more concisely using the following syntax:

normally $\neg permitted(to\_enter(G, shop))$ **if** $\#count\{P : member(P, G)\} > 3$.

We thus proposed the introduction of aggregate constructs borrowed from $\mathcal{ASP}$ into $\mathcal{AOPL}$.

c) *"The chairperson of a meeting is obliged to have the secretary circulating the minutes"*

We first defined our roles of $chairperson$ and $secretary$. Variables P and $P1$ refer to persons. The fluents and actions would be as follows:

– action: $to\_circulate\_minutes(P)$, $to\_ask(P, P1, to\_circulate\_minutes(P1)$;
– fluents: $has\_role(P, chairperson)$, $has\_role(P, secretary)$, and $was\_asked(P1, P, to\_circulate\_minutes(P1))$ (i.e., $P1$ was asked by P to circulate minutes).

The obligation of policy of the scenario would be encoded as follows:

normally $obl(to\_ask(P, P1, to\_circulate\_minutes(P1))$
if $has\_role(P, chairperson)$, $has\_role(P1, secretary)$.

normally $obl(to\_circulate\_minutes(P1))$
if $has\_role(P, chairperson)$, $has\_role(P1, secretary)$, $was\_asked(P1, P, to\_circulate\_minutes(P1))$.

The original text of the policy was thus captured by two rules: one saying that the chairperson has the obligation to ask the secretary to circulate the minutes, while the second policy says that the secretary has the obligation to actually circulate the minutes, if asked so by the chairperson.

d) *"Soldiers are forbidden to enter area (x, y)"*

We first defined the role of $soldier$, and the sort $coord$ for an integers that represent the coordinates (X, Y) of an area. The variable P below refers to a person, while $area(X, Y)$ is a symbol denoting the area with coordinates X and Y. The fluents and actions would be as follows:

- actions: $to\_enter(P, area(X, Y))$
- fluent: $has\_role(P, soldier)$

The authorization policy of the scenario would be encoded as follows:

normally $\neg permitted(to\_enter(P, area(x, y))$ **if** $has\_role(P, soldier)$.

where x and y are particular coordinates.

3. Chang and Meneguzzi (2016) [47]:

a) *"All immigrants holding valid passports must be accepted. Failure to comply may result in the loss of 5 credits."*

b) *"All immigrants holding passports that are not valid must not be accepted. Failure to comply may result in the loss of 10 credits and suspension from work activities for up to 10 time steps."*

We first defined the roles of sorts $enforcer$ and $immigrant$, and the $timestep$ sort. We decided to use the variables T (representing timesteps); and P and $P1$ (represening persons). The fluents and actions would be as follows:

- actions: $to\_accept(P, P1, passport)$,
- fluents: $has\_valid(P, passport), has\_role(P, immigrant), has\_role(P, enforcer)$

The obligation policy (a) was encoded as follows:

$d_1(P)$: **normally** $obl(to\_accept(P, P1, passport))$
if $has\_role(P, enforcer)$,
$has\_role(P1, immigrant)$,
$has\_valid(P1, passport)$.

As there is no built-in mechanism to represent penalties in $\mathcal{AOPL}$, we suggested utilizing the labels for defeasible policy rules available in $\mathcal{AOPL}$ as a means to connect penalties to the rules they pertain to. We proposed associating penalties with these labels in case of violations, as outlined below:

$penalty(d_1(P), P, 5, 0)$ **if** $person(P)$.

The penalty rule above indicates that in violation of the defeasible policy $d_1(P)$, person P incurs a loss of 5 credits with no suspension (i.e., a suspension of 0 time steps).

The authorization policy (b) was encoded as follows:

$d_2(P)$: **normally** $\neg permitted(to\_accept(P, P1, passport))$
if $has\_role(P, enforcer)$,
$has\_role(P1, immigrant)$,
$\neg has\_valid(P1, passport)$.
$penalty(d_2(P), P, 10, 10)$.

The penalty rule above indicates that in violation of the defeasible policy $d_2(P)$, person P incurs a loss of 10 credits with 10 days suspension.

Note that only defeasible rules have labels in $\mathcal{AOPL}$. In order for our proposed solution for penalties to function correctly in $\mathcal{AOPL}$, all rules, including strict policy rules, would be required to have a label assigned as suggested in previous work by Inclezan [53].

The scenarios encoded above underwent evaluation by the researcher, considering their power of expression, readability & writability, and conciseness. This assessment was conducted on a scale ranging from one (1) to five (5), as illustrated in Table 7.1. For further details on the specific criteria ranges, please refer to Table 7.2.

Table 7.1: Evaluation of Scenarios

Scenarios	**Power of Expression**	**Readability & Writability**	**Conciseness**
Scenario 1 a)	4	4	4
Scenario 1 b)	4	4	4
Scenario 1 c)	4	4	4
Scenario 2 a)	4	4	4
Scenario 2 b)	3	3	2
Scenario 2 c)	4	3	3
Scenario 2 d)	5	5	5
Scenario 3 a)	4	3	4
Scenario 3 b)	4	3	4
Average	4.00	3.67	3.77

Table 7.2: Criteria Scoring (1-5)

Criterion	Description
Power of Expression 1-5	1: Unclear and incomplete 2: Partially captures requirements 3: Adequate representation 4: Effective clarity and accuracy 5: Exceptional clarity and precision
Readability & Writability 1-5	1: Convoluted and difficult 2: Somewhat readable but complex 3: Reasonably readable and writable 4: Clear and straightforward 5: Exceptionally readable and writable
Conciseness 1-5	1: Overly verbose 2: Somewhat verbose 3: Balanced clarity and brevity 4: Concise and to the point 5: Exceptionally concise

In Table 7.1, the evaluation of scenarios based on the criteria of Power of Expression, Readability & Writability, and Conciseness provided valuable insights into the performance of $\mathcal{AOPL}$ in representing various scenarios.

Scenario 2b stands out as particularly challenging, receiving the lowest scores across all three evaluation criteria. This scenario posed significant difficulties in translation into $\mathcal{AOPL}$ policies, primarily due to the absence of aggregate constructs within $\mathcal{AOPL}$, a feature readily available in $\mathcal{ASP}$.

Integrating aggregate constructs like 'count' into $\mathcal{AOPL}$ offers substantial benefits across all three evaluation criteria: Power of Expression, Readability & Writability, and Conciseness. By enabling more sophisticated and nuanced policy specifications, these constructs enhance the Power of Expression. Moreover, they simplify the representation of complex logical computations, thereby improving Readability & Writability. Additionally, by reducing the need for extensive rule sets and conditional statements, aggregate constructs promote Conciseness, leading to clearer and more concise representations of policy requirements.

The average scores for each category in Table 7.1 reveal important trends. While the scenarios generally achieved an adequate level of representation in $\mathcal{AOPL}$, as indicated by the average score for Power of Expression (4.00), some were more challenging to read and write in $\mathcal{AOPL}$, as suggested by the slightly lower average score for Readability & Writability (3.67). This underscores the trade-off between expressive power and usability.

Conciseness, with an average score of 3.77, reflects a balance between clarity and brevity in scenario representation. While generally concise and to the point, some instances required verbosity

to accurately convey policy intricacies.

Overall, the evaluation highlights the nuanced interplay between expressive capabilities and usability in $\mathcal{AOPL}$ across diverse scenarios. Integrating aggregate constructs like 'count' holds promise for enhancing $\mathcal{AOPL}$'s effectiveness and usability, ultimately improving its overall performance across all evaluation criteria.

However, since the evaluation was solely conducted by the researcher, there existed a limitation in our assessment due to the absence of external validation. To address this, we introduced a quantitative evaluation based on the character count and lexicon count. The character count in our evaluation examined the total number of characters required to express a particular policy, excluding spaces. Conversely, the lexicon count referred to the total number of syntax elements a reader or user must understand to fully comprehend what a policy framework aims to express.

For example, in scenario 2d, where "*Soldiers are forbidden to enter area (x, y)*", when expressed in $\mathcal{AOPL}$, we obtained the following expression:

normally $\neg permitted(to\_enter(P, area(x, y))$ **if** $has\_role(P, soldier)$.

This expression consisted of sixty-two (62) characters (excluding spaces) and had a lexicon count of six (6) (including the standard syntax of $\mathcal{AOPL}$ such as 'normally', 'permitted/obligation', 'if', 'condition', as well as '¬ permitted' and 'has_role'). Utilizing this evaluation methodology we created our quantitative table 7.3.

Table 7.3: Evaluation Table of AOPL against Existing Work

	AOPL		**Original Representation**	
Scenario	**# of characters**	**Lexicon Count**	**# of characters**	**Lexicon Count**
1a	106	8	33	5
1b	102	8	38	5
1c	100	8	31	5
2a	70	6	84	7
2b	80	7	55	7
2c	225	6	51	5
2d	62	6	N/A	N/A
3a	144	8	60	8
3b	143	8	65	8
AVERAGE	**114.7**	**7.2**	**52.1**	**6.3**

Based on the evaluation table, it is evident that $\mathcal{AOPL}$ tends to require the use of more characters to express scenarios compared to the existing works from which those scenarios were pulled. On average, $\mathcal{AOPL}$ policies required approximately 114.7 characters, whereas the original representations needed only about 52.1 characters. Similarly, the lexicon count for $\mathcal{AOPL}$ was slightly higher, averaging 7.2 compared to 6.3 for the original representations.

Despite $\mathcal{AOPL}$'s tendency to use more characters and lexicons, this could be indicative of $\mathcal{AOPL}$'s approach to expressing policies in a manner closer to natural language than its counter-

parts. This suggested a potential weak correlation between readability and expressive power, where $\mathcal{AOPL}$'s emphasis on detailed expression may enhance clarity and precision.

However, the outlier scenario 2c, where there was a significant spike in the number of characters required by $\mathcal{AOPL}$ compared to the original representation. This exceptional case suggests that there might be a threshold beyond which an excessive number of characters decreases readability. Thus, while a higher number of characters may generally indicate enhanced expressiveness, there exists a point at which it could negatively impact readability.

7.3 Discussion

The methodology employed in this thesis encompasses three distinct tasks, each contributing to the overarching goal of exploring, evaluating, and extending the $\mathcal{AOPL}$ framework for normative reasoning.

Research Question 1

Task 1 initiated with a meticulous process of paper selection and benchmark creation, laying the foundation for subsequent analyses. This preliminary step was pivotal for addressing our first research question:

RQ1: *How do the $\mathcal{AOPL}$, $\mathcal{EPA}$, and other policy frameworks compare, particularly in their ability to support (a) policy analysis; and (b) encoding policies for different roles?*

Through our benchmark creation process, we uncovered that while these policy frameworks exhibit similarities in certain aspects, each demonstrates a tendency to excel in handling specific types of scenarios. Specifically, $\mathcal{AOPL}$ was tailored to address obligations and authorizations, providing it with certain policy analysis capabilities such as managing defeasible rules and implementing soft policy enforcement through the 'normally' keyword. However, this specialization resulted in only a brief acknowledgment regarding its ability to handle roles, a feature present in some of the other papers. $\mathcal{EPA}$, as one of these frameworks, emerged as a much more robust framework than $\mathcal{AOPL}$, capable of performing more extensive policy analysis. It offers functionalities such as handling modality conflicts, addressing coverage gaps, and facilitating policy comparison. Additionally, $\mathcal{EPA}$ provides explicit documentation on handling roles in scenarios where $\mathcal{AOPL}$ falls short. Among the ten (10) additional papers examined, only three (3) explicitly stated their ability to handle roles, while the remainder either do not support roles or do not explicitly state their support. Regarding their policy analysis capabilities, we found that they were all comparable to those found in recent papers on $\mathcal{AOPL}$ [19, 53].

Navigating the Digital Bibliography & Library Project (DBLP) platform, presented its own set of challenges. Although the platform boasts a vast repository of scholarly articles, the sheer volume of available literature made it difficult to pinpoint relevant papers. In our search, we sifted through approximately over two hundred (200) papers. Additionally, the stringent criteria for paper selection, which included factors like relevance, temporal alignment, and thematic congruence, further complicated the process.

Moreover, creating the core literature set itself posed its own set of challenges, particularly in integrating examples from the selected papers. Each example required careful consideration to ensure it accurately represented the policies, planning domains, and scenarios relevant to the research objectives. Additionally, aligning the examples with the extended $\mathcal{AOPL}$ framework demanded meticulous attention to detail, further complicating the benchmark creation process. However, despite these challenges, a curated set of ten papers meeting the defined criteria was successfully compiled, forming the basis for core literature set.

Answer to RQ1: While Task 1 outlined the process of evaluating $\mathcal{AOPL}$ and other policy frameworks against the selected criteria, it served mainly as a precursor to Task 2, which directly addressed **RQ1** by analyzing how well these frameworks support policy analysis and the encoding of policies for different roles. Table 5.1 provided a comprehensive overview of the selected criteria and their relevance to the evaluation of $\mathcal{AOPL}$.

Research Question 2

Task 2 involved creating a detailed comparison table to examine and compare $\mathcal{AOPL}$ with $\mathcal{EPA}$ and other frameworks, using insights from the selected papers in Task 1. This table played a pivotal role in addressing our second research question:

RQ2: *What extensions of $\mathcal{AOPL}$ are required to enhance its expressive power?*

Through the analysis facilitated by this table, we identified several extensions necessary for $\mathcal{AOPL}$ to enhance its expressive power.

While the criteria for comparison were chosen by delineating the nuances of each framework within the table proved challenging. Classifying the treatment of criteria within each paper required careful consideration of context and underlying assumptions, ensuring a nuanced evaluation of the frameworks.

Expressive power refers to a policy framework's ability to effectively represent a scenario's policies and rules. This encompasses factors such as the accuracy and comprehensiveness of rule representation, the framework's capacity for nuanced policy descriptions, and its ability to handle complex scenarios with clarity.

Upon reviewing our comparison table (Table 5.1), we noted limitations in $\mathcal{AOPL}$'s ability to represent temporal restrictions, modality conflicts, roles, and penalties. These aspects were found to be prevalent in approximately fifty percent (50%) of the papers used in our benchmark, indicating a need for enhancements in $\mathcal{AOPL}$'s expressive capabilities regarding these criteria.

Answer to RQ2: $\mathcal{AOPL}$ required the ability to express temporal constraints, modality conflicts, penalties and roles.

Research Question 3

In Task 3, our focus shifted to extending $\mathcal{AOPL}$ to incorporate roles within a hypothetical university scenario. This task was pivotal in addressing our third and final research question:

RQ3: *How could $\mathcal{AOPL}$ be extended to encode policies for different roles?*

Through this task, we successfully proposed an extension of $\mathcal{AOPL}$ with roles, which not only maintained the framework's readability and writability but also enhanced its expressive power. This extension involved the introduction of a new fluent called 'has_role', which was implemented in every scenario. This fluent was responsible for tracking roles assigned by a designated superuser or admin, referred to as the $enforcer$ in our scenario. The enforcer was tasked with managing the assignment and removal of all roles within the system. Additionally, each role restricted agents to perform only actions permitted by that role. This ensured a certain level of separation of duties, akin to common institutional practices, further amplifying $\mathcal{AOPL}$'s expressive capacity.

Following the validation of the extended $\mathcal{AOPL}$ framework with roles against our benchmark scenarios, which demonstrated notable conciseness and readability & writability, we encountered challenges during the encoding process that highlighted areas for improvement, particularly in handling scenarios involving roles with the consequence of penalties.

Answer to RQ3: We extended $\mathcal{AOPL}$ by introducing a novel fluent called **has_role**. This addition doesn't directly alter the language itself but proposes a new methodology for encoding policy enforcement within $\mathcal{AOPL}$ going forward.

Chapter 8

Conclusion

In this thesis, we focused on the comparative analysis of various policy frameworks, and proposed an extension of the $\mathcal{AOPL}$ framework with the ability to support roles. Our research was structured into three main tasks, each contributing uniquely to our overarching goal. Task 1 involved a meticulous process of paper selection and benchmark creation, establishing a solid foundation for subsequent analyses. Through this task, we identified relevant literature and compiled a benchmark of scenarios, laying the groundwork for our further investigations. Task 2 focused on examining and comparing $\mathcal{AOPL}$ with other policy frameworks, particularly $\mathcal{EPA}$. We created a detailed comparison table based on insights gleaned from the selected papers in Task 1. This comparative analysis highlighted key areas where $\mathcal{AOPL}$ could be extended to enhance its expressive power. Task 3 delved into extending $\mathcal{AOPL}$ with roles, a critical aspect for encoding policies for different roles within scenarios. We proposed an extension to $\mathcal{AOPL}$ with roles, maintaining readability and writability while enhancing its expressive power. This extension was validated against benchmark scenarios, demonstrating its effectiveness in managing scenarios with roles.

List of Contributions

- Establishment of a benchmark of scenarios for $\mathcal{AOPL}$ evaluation.
- Creation of a detailed comparison table between $\mathcal{AOPL}$ and other policy frameworks.
- Proposal and validation of an extension to $\mathcal{AOPL}$ with roles, enhancing its expressive power.

Future Work

Future work should focus on extending $\mathcal{AOPL}$ to address the missing criteria highlighted in this thesis. The most important additions would include:

- a thorough representation of **penalties** (not just in terms of an amount/credit to pay, but also in terms of other negative consequences)
- detection and resolution of **modality conflicts** (i.e., situations when authorization policies conflict with obligation policies)
- means for representing **temporal constraints** (i.e., when restrictions are not permanent, but rather apply for a certain time period)

Additionally, further exploration into the integration of $\mathcal{AOPL}$ with other emerging technologies and frameworks could lead to more comprehensive and versatile normative reasoning solutions.

References

[1] Michael Gelfond and Yulia Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.

[2] John Meyer and Daniela Inclezan. APIA: An architecture for policy-aware intentional agents. *Electronic Proceedings in Theoretical Computer Science*, 345:84–98, sep 2021.

[3] Michael Gelfond and Jorge Lobo. Authorization and obligation policies in dynamic systems. *Theory and Practice of Logic Programming*, 8(3):275–318, 2008.

[4] Robert Craven, Jorge Lobo, Jiefei Ma, Alessandra Russo, Emil Lupu, and Arosha Bandara. Expressive policy analysis with enhanced system dynamicity. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 239–250. ACM, 2009.

[5] Michael Wooldridge. *An introduction to multi-agent systems*. John Wiley & Sons, 2009.

[6] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson Education, 2010.

[7] Ankit Jain and MP Singh. Designing intelligent agents: A comprehensive approach. *International Journal of Computer Science and Network Security*, 11(7):137–143, 2011.

[8] Marcello Balduccini and Michael Gelfond. The AAA architecture: An overview. In *AAAI Spring Symposium: Emotion, Personality, and Social Behavior*, pages 1–6, 2008.

[9] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16):193–210, 1998.

[10] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[11] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.

[12] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten

Schaub. ASP-core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, December 2019.

[13] Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. Abstract gringo. *Theory and Practice of Logic Programming*, 15(4-5):449–463, 2015.

[14] Justin Blount. *An architecture for intentional agents*. PhD thesis, Texas Tech University, December 2013.

[15] Marcello Balduccini and Sara Girotto. *ASP as a Cognitive Modeling Tool: Short-Term Memory and Long-Term Memory*, pages 377–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[16] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002. Knowledge Representation and Logic Programming.

[17] Van Nguyen, Stylianos Loukas Vasileiou, Tran Cao Son, and William Yeoh. Explainable Planning Using Answer Set Programming. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, pages 662–666, 9 2020.

[18] Tran Cao Son, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: A survey. *CoRR*, abs/2202.05793, 2022.

[19] Charles Harders and Daniela Inclezan. Plan selection framework for policy-aware autonomous agents. In Sarah Alice Gaggl, Maria Vanina Martinez, and Magdalena Ortiz, editors, *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, volume 14281 of *Lecture Notes in Computer Science*, pages 638–646. Springer, 2023.

[20] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract Meaning Representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[21] Somak Aditya, Chitta Baral, Nguyen Ha Vo, Joohyung Lee, Jieping Ye, Zaw Naung, Barry Lumpkin, Jenny Hastings, Richard B. Scherl, Dawn M. Sweet, and Daniela Inclezan. Recognizing social constructs from textual conversation. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1293–1298. The Association for Computational Linguistics, 2015.

[22] Yuliya Lierler, Daniela Inclezan, and Michael Gelfond. Action languages and question answering. In *IWCS 2017 - 12th International Conference on Computational Semantics - Short papers, Montpellier, France, September 19 - 22, 2017*, 2017.

[23] Qinglin Zhang and Daniela Inclezan. An application of ASP theories of intentions to understanding restaurant scenarios. In *Proceedings of PAoASP'17*, 2017.

[24] Daniela Inclezan, Qinglin Zhang, Marcello Balduccini, and Ankush Israney. An ASP methodology for understanding narratives about stereotypical activities. *Theory and Practice of Logic Programming*, 18(3–4):535–552, 2018.

[25] Qinglin Zhang, Chris Benton, and Daniela Inclezan. An application of asp theories of intentions to understanding restaurant scenarios: Insights and narrative corpus. *Theory and Practice of Logic Programming*, 20(2):273–293, 2020.

[26] Gaurab Luitel, Matthew Stephan, and Daniela Inclezan. Model level design pattern instance detection using answer set programming. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering, MiSE@ICSE 2016, Austin, Texas, USA, May 16-17, 2016*, pages 13–19. ACM, 2016.

[27] Marcello Balduccini and Sara Girotto. ASP as a cognitive modeling tool: Short-term memory and long-term memory. In Marcello Balduccini and Tran Cao Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Computer Science*, pages 377–397. Springer, 2011.

[28] Daniela Inclezan. An application of answer set programming to the field of second language acquisition. *Theory Pract. Log. Program.*, 15(1):1–17, 2015.

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

[30] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. pages 141–151, 01 2004.

[31] Michael Gelfond and Daniela Inclezan. Some properties of system descriptions of ALd. *J. Appl. Non Class. Logics*, 23(1-2):105–120, 2013.

[32] Daniela Inclezan. *Modular action language ALM for dynamic domain representation*. PhD thesis, Texas Tech University, Lubbock, USA, 2012.

[33] Daniela Inclezan and Michael Gelfond. Modular Action Language ALM. *Theory and Practice of Logic Programming*, 16(2):189—235, 2016.

[34] Khair Eddin Sabri and Nadim Obeid. A temporal defeasible logic for handling access control policies. *Applied Intelligence*, 44(1):30–42, 2016.

[35] Jamal Abdali, Karim El Guemhioui, and Luigi Logrippo. A metamodel for hybrid access control policies. *Journal of Software*, 10:784–797, 07 2015.

[36] Anthony Opara, Haan Mo Johng, Tom Hill, and Lawrence Chung. A framework for representing internet of things security and privacy policies and detecting potential problems. pages 198–201, 04 2022.

[37] Robert Kowalski and Marek Sergot. *A Logic-Based Calculus of Events*, pages 23–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[38] Michael Thielscher. *Nondeterministic Actions in the Fluent Calculus: Disjunctive State Update Axioms*, pages 327–345. Springer Netherlands, Dordrecht, 2000.

[39] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1):109–169, 2000.

[40] Tae-Won Kim, Joohyung Lee, and Ravi Palla. Circumscriptive event calculus as answer set programming. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, page 823–829, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[41] Joaquín Arias, Manuel Carro, Zhuo Chen, and Gopal Gupta. Modeling and reasoning in event calculus using goal-directed constraint answer set programming. *CoRR*, abs/2106.14566, 2021.

[42] Tassilo Pellegrini, Giray Havur, Simon Steyskal, Oleksandra Panasiuk, Anna Fensel, Victor Mireles-Chavez, Thomas Thurner, Axel Polleres, Sabrina Kirrane, and Andrea Schönhofer. Dalicc: A license management framework for digital assets. 02 2019.

[43] Domenico Corapi, Alessandra Russo, Marina De Vos, Julian Padget, and Ken Satoh. Normative design using inductive learning. *Theory and Practice of Logic Programming*, 11(4-5):783–799, 2011.

[44] Zohreh Shams, Marina De Vos, Nir Oren, and Julian Padget. Argumentation-based reasoning about plans, maintenance goals, and norms. *ACM Trans. Auton. Adapt. Syst.*, 14(3), feb 2020.

[45] Zohreh Shams, Marina De Vos, Julian Padget, and Wamberto W. Vasconcelos. Practical reasoning with norms for autonomous software agents (full edition), 2017.

[46] Huib Aldewereld, Virginia Dignum, and Wamberto Vasconcelos. Reasoning with group norms in software agent organisations. In Virginia Dignum, Pablo Noriega, Murat Sensoy, and Jaime Simão Sichman, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems XI*, pages 1–21, Cham, 2016. Springer International Publishing.

[47] Stephan Chang and Felipe Meneguzzi. Simulating normative behaviour in multi-agent environments using monitoring artefacts. volume 9628, pages 59–77, 07 2016.

[48] Luca Gasparini, Timothy Norman, Martin Kollingbaum, Liang Chen, and John-jules Meyer. CÒir: Verifying normative specifications of complex systems. volume 9628, pages 134–153, 07 2016.

[49] Jie Jiang and Huib Aldewereld. Modeling and detecting norm conflicts in regulated organizations. volume 9628, pages 173–190, 07 2016.

[50] European Commission. Authorised economic operator. `http://ec.europa.eu/taxation_customs/customs/policy_issues/customs_security/aeo/`, 2013.

[51] Thomas C. King, Marina De Vos, Virginia Dignum, Catholijn M. Jonker, Tingting Li, Julian Padget, and M. Birna van Riemsdijk. Automated multi-level governance compliance checking. *Autonomous Agents and Multi-Agent Systems*, 31(6):1283–1343, November 2017.

[52] Marina De Vos, Sabrina Kirrane, Julian Padget, and Ken Satoh. Odrl policy modelling and compliance checking. In *Rules and Reasoning: Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings*, page 36–51, Berlin, Heidelberg, 2019. Springer-Verlag.

[53] Daniela Inclezan. An ASP framework for the refinement of authorization and obligation policies. *Theory and Practice of Logic Programming*, 23(4):832–847, 2023.

[54] Monica Palmirani, Guido Governatori, Antonino Rotolo, Said Tabet, Harold Boley, and Adrian Paschke. Legalruleml: Xml-based rules and norms. In Frank Olken, Monica Palmirani, and Davide Sottara, editors, *Rule-Based Modeling and Computing on the Semantic Web*, pages 298–312, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[55] R.T. Simon and M.E. Zurko. Separation of duty in role-based environments. In *Proceedings 10th Computer Security Foundations Workshop*, pages 183–194, 1997.