A Framework for Defining Behavior Modes in Policy-Aware Autonomous Agents

Daniela Inclezan[1], Charles Harders[1] and Vineel S. K. Tummala[1]

[1]Miami University, Oxford, OH USA

Abstract

Autonomous agents operating in policy-governed environments can exhibit varying degrees of policy compliance, from full adherence to complete non-conformance. This paper introduces a framework that enables controllers of autonomous agents to define these different compliance attitudes, referred to as "behavior modes." The framework demonstrates its utility by simulating agent decision-making processes when selecting plans to achieve their goals. It leverages the policy specification language AOPL by Gelfond and Lobo and is implemented using Answer Set Programming (ASP). Experimental results in two example domains showcase the potential of this framework to simulate diverse agent attitudes.

Keywords

ASP, policies/norms, autonomous agents, behavior modes

1. Introduction

Autonomous agents are becoming pervasive in a variety of aspects of daily life, including healthcare and elderly care, manufacturing, or self-driving transportation. In order to ensure that such agents exhibit a safe and secure behavior that is appropriate for the environment in which they act, for their interactions with humans, and the tasks that they are supposed to solve, it is expected that they operate within the boundaries of given policies (or norms). Policies may be specified by the creators or controllers of autonomous agents and can include specification about actions that the autonomous agents are *required* to perform (or not perform) in specific situations, and actions they are *allowed* (or not) to perform. The former are referred to as *obligation policies*, while the latter are *authorization policies*. An autonomous agent may be built so that it can decide whether to abide by these policies or not, at different points in time, depending on the priorities set for it by its controller and the situation at hand. In normal scenarios, compliance with policies would take precedence over all other aspects, but situations may arise when agents may need to switch to a less compliant behavior, due to higher level objectives, such as when the agent participates in a rescue operation. The issue of different agent attitudes towards policy-compliance and how that affects the plan selection process is also relevant to policy makers. Simulating human agents with a range of behavior modes towards policy-compliance can help policy makers to better assess the consequences of their policies.

In this paper, we propose a framework for the specification of behavior modes for an autonomous agent that guide the plan selection process. In our framework, agent behaviors are defined by setting preferences and constraints for several metrics related to policy compliance and plans. Reasoning about compliance and planning with behavior modes is achieved using Answer Set Programming (ASP).

In order to reason about policy compliance, an autonomous agent needs to have a model of the changing environment in which it acts, a representation of policy statements, and reasoning algorithms for determining policy compliance. We use action language AL_d [1, 2] for the encoding of the agent's environment. AL_d has a concise syntax and incorporates established solutions to the ramification and qualification problems in its semantics, which is defined via a translation into ASP. For the specification of a policy, we employ language *AOPL* [3]. *AOPL* allows the description of complex policies, which are

Original Submission – ASPOCP 2024

inclezd@miamioh.edu (D. Inclezan); harderc2@miamioh.edu (C. Harders); tummalvs@miamioh.edu (V. S. K. Tummala)
0000-0002-4534-9658 (D. Inclezan)

not limited to role-based access control. Policy statements of *AOPL* can be either strict or *defeasible*, thus allowing for exceptions to be specified. The policy compliance of an agent action is computed by translating the *AOPL* policy into ASP, adding compliance checking modules, and checking the solutions (i.e., answer sets) of the resulting logic program.

In the original work by Gelfond and Lobo, only plans consisting of actions at the same level of compliance were considered (e.g., plans consisting only of actions explicitly known to be compliant). Harders and Inclezan [4] introduced a finer-grained qualification of plans consisting of a mix of actions at different levels of compliance. In the current work, we define how behavior modes can be specified by an agent's controller, based on a variety of metrics. To illustrate the importance of defining different behavior modes and analyzing the resulting plans, let's consider the following example:

Example 1. *We assume a configuration in which there are six rooms, $r_1, \ldots, r_6$, with the layout shown in Figure 1. The autonomous agent a is in room r_1 and wants to get to room r_3. The agent is able to go from one room to an adjacent room (i.e., perform action enter(a, r) – agent a enters room r). Some of the agent's possible actions are indicated by arrows in Figure 1. Assume that the agent is governed by a consistent and unambiguous policy and that all actions are compliant with respect to obligations. However, with respect to authorization policies, some actions are known to be compliant (called* strongly-compliant *and labeled by "s" in the picture); others are unknown to be compliant or non-compliant (called* underspecified *actions labeled by "u"), and finally some are known to be non-compliant (labeled by "n").*

Let's consider four of the plans that would accomplish the goal:

$\alpha_1 = \langle enter(a, r_4), enter(a, r_5), enter(a, r_6), enter(a, r_3)\rangle,$
$\alpha_2 = \langle enter(a, r_4), enter(a, r_5), enter(a, r_2), enter(a, r_3)\rangle,$
$\alpha_3 = \langle enter(a, r_2), enter(a, r_3)\rangle,$ *and*
$\alpha_4 = \langle enter(a, r_2), enter(a, r_5), enter(a, r_6), enter(a, r_3)\rangle.$

According to Gelfond and Lobo's definitions [3] included in Section 2, α_1 is strongly-compliant while $\alpha_1, \alpha_2, \alpha_3$ are weakly-compliant (i.e., consisting solely of actions not known to be non-compliant). For consistent and unambiguous policies, weakly-compliant actions include strongly-compliant and underspecified actions as shown by Inclezan [5]. Nothing can be said about α_4 because its actions are not at the same level of compliance. Note that α_1 is both strongly- and weakly-compliant because the class of weakly-compliant actions includes strongly-compliant actions in consistent and unambiguous policies [5].

This classification of plans does not allow the controller of an agent enough flexibility to specify desirable behavior modes for its agent. Instead, we want to allow a controller to specify behavior modes in terms of preferences such as "prioritize compliance over plan length." In that case α_1 would be the best plan, followed by α_2 and α_3. Plan α_4 may not even be considered to be an option due to the inclusion of a non-compliant action. If a behavior mode is defined in which plan length is prioritized over compliance (which may be useful in emergency rescue operations assuming that plan length correlates to plan execution in real time), then α_3 would be the best, followed by α_1 and α_2. Note that α_1 and α_2 have the same length, but all actions of α_1 are guaranteed to be compliant (i.e., they are strongly-compliant), whereas one action of α_2 is not explicitly stated to be compliant but it is not non-compliant either (i.e., it is underspecified).

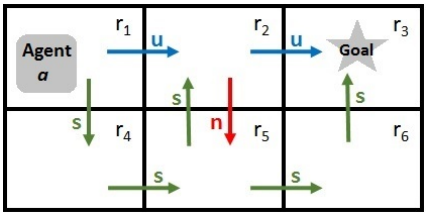

Figure 1: Moving between Rooms (Authorization: s - strongly-compliant action; u - underspecified action; n - non-compliant action. Obligation: all actions are compliant.)

The main contributions of this paper are as follows:

1. The formalization of a framework (leveraging ASP, action language AL_d, and policy language *AOPL*) in which a controller can specify behavior modes based on a collection of metrics related to policy compliance and other aspects
2. A description of its implementation in ASP
3. Experimental results on the application of this framework to example domains.

In what follows, we start with preliminary information about the policy specification language *AOPL*. We analyze issues related to policy compliance relevant to planning in Section 3 and introduce our framework in Section 4. We discuss our ASP implementation in Section 5 and report experimental results in Section 6. We investigate related work in Section 7 and end with conclusions.

2. Background: Policy Specification Language *AOPL*

We assume that readers are familiar with language ASP [6, 7, 8, 2] and ASP solvers such as Clingo[1] or DLV[2] [9, 10], including concepts such as *choice rules* and *aggregates*. In what follows, we give a brief introduction to the policy specification language *AOPL*.

While several logic-based languages for the specification of policies (or norms) exist [11, 12, 13], we use Gelfond and Lobo's *Authorization and Obligation Policy Language* (*AOPL*) [3] in our work due to its close connection and seamless coupling with ASP and action languages [14]. *AOPL* is designed for the specification of policies that should govern the behavior of an intelligent agent acting in a dynamic environment. Policies are statements about permissions and prohibitions (called *authorizations* in *AOPL*), as well as obligations and dispensations (simply called *obligations*).

AOPL works in conjunction with a dynamic system description of the agent's environment written in an action language, such as AL_d [1, 2]. The signature of the dynamic system description includes: *sorts* (i.e., types, classes) of objects in the domain; *fluents* (i.e., domain properties that may be changed by actions); and *(elementary) actions*.

Strict policies of *AOPL* are specified using predicates *permitted* for authorization policies, *obl* for obligation policies, and statements of the form:

$$\begin{array}{rll} permitted(e) & \textbf{if} & cond \\ \neg permitted(e) & \textbf{if} & cond \\ obl(h) & \textbf{if} & cond \\ \neg obl(h) & \textbf{if} & cond \end{array} \tag{1}$$

where e is an elementary action; h is a happening (i.e., an elementary action or its negation[3]); and *cond* is a (possibly empty) collection of atoms of the signature, except *prefer* atoms. In addition to the *strict* policy statements, *AOPL* supports *defeasible* statements and *priorities* between them:

$$d : \textbf{normally } permitted(e) \quad \textbf{if } cond \tag{2a}$$
$$d : \textbf{normally } \neg permitted(e) \quad \textbf{if } cond \tag{2b}$$
$$d : \textbf{normally } obl(h) \quad \textbf{if } cond \tag{2c}$$
$$d : \textbf{normally } \neg obl(h) \quad \textbf{if } cond \tag{2d}$$
$$prefer(d_i, d_j) \tag{2e}$$

where d_i and d_j in (2e) are labels for defeasible rules (similar to the label d in statements of the form (2a) - (2d)) and statement (2e) says that rule d_i overrides rule labeled d_j. We call *permitted* and *obl* predicates appearing to the left of the keyword **if** the *head* of a policy rule. Strict rules override defeasible rules with the opposite head; *prefer* predicates are used between defeasible rules with opposite heads to describe exceptions to the overridden defeasible rule (e.g., exceptions to d_j in (2e)).

[1] https://potassco.org/clingo/
[2] https://www.dlvsystem.it/dlvsite/
[3] If $obl(\neg e)$ is true, then the agent must not execute e.

Reasoning about the compliance of an agent to a policy is defined via an ASP translation of the policy and dynamic system description. The lp translation function is straightforward for atoms, literals, and strict rules. In (3), (4), and (5) we indicate the lp translation for defeasible rules (2a), (2b), and preference rule (2e) respectively:

$$permitted(e) \leftarrow lp(cond),\ \text{not}\ ab(d),\ \text{not}\ \neg permitted(e) \tag{3}$$

$$\neg permitted(e) \leftarrow lp(cond),\ \text{not}\ ab(d),\ \text{not}\ permitted(e) \tag{4}$$

$$ab(d_j) \leftarrow lp(cond_i) \tag{5}$$

where *ab* stands for abnormal w.r.t. a default statement and is a common way of representing exceptions to defaults in ASP methodology [2]. Similarly for defeasible obligation policies of forms (2c) and (2d).

As an example, a policy for the scenario in Example 1 may consist of the statements:

$$\begin{array}{lll} d_1: & \textbf{normally} & obl(\neg enter(a,r)) \ \textbf{if}\ fire(r) \\ d_2: & \textbf{normally} & \neg obl(\neg enter(a,r)) \ \textbf{if}\ fire(r),\ wearing\_prot\_equip(a) \end{array}$$

$$prefer(d_2, d_1)$$

The first policy prohibits entering a room where there is an active fire; the second statement together with the preference rule cancel the first policy out whenever the agent is wearing protective equipment.

Given a policy P and a state σ of the (transition diagram associated with the) dynamic system D,

$$lp(P,\sigma) =_{def} lp(P) \cup lp(\sigma)$$

Gelfond and Lobo define a policy as *consistent* if, for every state σ of D, the logic program $lp(P,\sigma)$ is consistent (i.e., has an answer set). A policy is *categorical* if lp has *exactly one answer set* for every state σ of D, i.e., it is unambiguous. We show below the definitions of compliance for actions and paths, introduced by Gelfond and Lobo [3]. Note that *ca* denotes a compound action, while *e* refers to an elementary action. By $l \in P(\sigma)$ we indicate that $lp(P,\sigma) \models l$; by $l \notin P(\sigma)$ we mean $lp(P,\sigma) \not\models l$.

Definition 1 (Compliance for Authorizations). *(Definitions 4 and 5 in the original paper)*

- *An event* $\langle \sigma, ca \rangle$ *is* strongly compliant *with authorization policy P if for every* $e \in ca$ *we have that* $permitted(e) \in P(\sigma)$ *(i.e., the logic program* $lp(P,\sigma)$ *entails* $permitted(e)$, $lp(P,\sigma) \models permitted(e)$*).*
- *An event* $\langle \sigma, ca \rangle$ *is* weakly compliant *with authorization policy P if for every* $e \in ca$ *we have that* $\neg permitted(e) \notin P(\sigma)$ *(i.e., the logic program* $lp(P,\sigma)$ *does not entail* $\neg permitted(e)$, $lp(P,\sigma) \not\models \neg permitted(e)$*).*
- *An event* $\langle \sigma, ca \rangle$ *is* non-compliant *with authorization policy P if for every* $e \in ca$ *we have that* $\neg permitted(e) \in P(\sigma)$ *(i.e., the logic program* $lp(P,\sigma)$ *entails* $\neg permitted(e)$, $lp(P,\sigma) \models \neg permitted(e)$*).*
- *A path* $\langle \sigma_0, ca_0, \sigma_1, \ldots, \sigma_{n-1}, ca_{n-1}, \sigma_n \rangle$ *is* strongly (weakly) compliant *with authorization policy P if for every* $0 \leq i < n$ *the event* $\langle \sigma_i, ca_i \rangle$ *is strongly (weakly) compliant with P.*

Definition 2 (Compliance for Obligations). *(Definition 9 in the original)*

An event $\langle \sigma, ca \rangle$ *is* compliant *with obligation policy P if*

- *For every* $obl(e) \in P(\sigma)$ *we have that* $e \in ca$*, and*
- *For every* $obl(\neg e) \in P(\sigma)$ *we have that* $e \notin ca$*.*

Definition 3 (Compliance for Authorization and Obligations). *An event* $\langle \sigma, ca \rangle$ *is* strongly (weakly) compliant *with arbitrary policy P (i.e., a policy that may contain both authorization and obligation statements) if it is strongly (weakly) compliant with the authorization component of the policy and compliant with the obligation component of P.*

Note that compliance for paths is only defined for authorization policies (Definition 1), but it could be extended to obligation policies in a similar manner. However, such definitions are coarse-grained and do not allow distinguishing, for example, between two weakly-compliant paths. Also, *AOPL* does not discuss interactions between authorization and obligation policies referring to the same action, for instance situations when both $obl(e)$ and $\neg permitted(e)$ are entailed by $lp(P,\sigma)$.

3. Planning and Policy Compliance

Planning problems have as an input: (1) a description of the dynamic system in which the autonomous agent is acting, including information about actions the agent can execute, (2) an initial state, and (3) a desired set of fluents (or goal) to be achieved by the agent. A solution to a planning problem is a sequence of actions to be performed by the agent in order to achieve the goal state. *Answer Set Planning* [15] refers to the use of ASP to solve planning problems by reducing them to the computation of answer sets of a logic program. Each answer set corresponds to a possible plan. An ASP planning problem is defined as a triple $\langle D, \Gamma, \Delta \rangle$ where D is the ASP encoding of the dynamic system, Γ is a collection of fluent literals that hold in the initial state, and Δ is the set of fluent literals representing the goal.

The dynamic system description may be non-deterministic, in which case a computed plan *may* take the agent to the desired goal state after executing the plan, but this is not guaranteed [16]. Information about the initial state may also be incomplete. This defines the so-called class of *conformant* planning problems [16, 17, 18], which has a higher complexity than classical planning [19]. Since we focus on simulating an agent's behavior modes and plan selection w.r.t. to *compliance to policies*, we limit ourselves to *deterministic* dynamic system descriptions and *complete knowledge* about the initial state. This removes some of the complexities that are orthogonal to policy-compliance behavior modes. If a planning problem $\langle D, \Gamma, \Delta \rangle$ satisfies these properties, then Γ is a state of D (not just a subset of a state) and a solution to the planning problem is a sequence of agent actions $\alpha = \langle a_0, \dots, a_{n-1} \rangle$ that *guarantees* to reach a state σ_n such that $\Delta \subseteq \sigma_n$.

3.1. Authorizations

Inclezan [5] showed that (1) a division of paths into strongly- and weakly-compliant is too coarse to compare plans; (2) the label "weakly-compliant" is not specific enough to create a relative priority order between plans, because all strongly-compliant events are also weakly-compliant; and (3) there are interactions between authorizations and obligations that require more attention. In what follows, we will make use of the definitions below borrowed from work by Inclezan.

Definition 4. *An event $\langle \sigma, ca \rangle$ is* underspecified *with respect to authorization policy P if for every $e \in ca$ the logic program $lp(P, \sigma)$ entails both "not permitted(e)" and "not $\neg$permitted(e)" (i.e., permitted(e) and $\neg$permitted(e) are absent from every answer set of $lp(P, \sigma)$).*

In what follows we consider *categorical* (i.e., unambiguous) policies (the program $lp(P, \sigma)$ has exactly one answer set for every state σ of D), since categoricity is a property that policies should strive towards [3]. This will be relevant to plan selection, as it will facilitate creating an ordering between plans. Given a categorical policy, Inclezan [5] proved that the actions in a plan can be divided into three disjunctive sets with respect to authorization: strongly-compliant, underspecified, and non-compliant events, in this order in terms of desirability. For two plans with the same length (and equivalent in terms of their obligation compliance), requirements can be imposed on the number of actions in each category, or an ordering can be introduced based on the percentage of actions of each kind.

3.2. Obligations

From the point of view of obligation policies, the cases that require attention are captured in Table 1. Note that we continue to focus on *categorical* policies. Moreover, we assume that the obligation policies are *non-conflicting*, i.e., $lp(P, \sigma)$ does not entail both $obl(e)$ and $obl(\neg e)$ (i.e., obligation to execute and to not execute e) for an action e. Cases when $lp(P, \sigma)$ entails $\neg obl(e)$ or $\neg obl(\neg e)$ represent the absence of an obligation, and therefore it is not relevant whether action e is planned to be executed in state σ or not. Out of the combinations that count towards compliance, situations when $lp(P, \sigma)$ entails $obl(\neg e)$ and e is not planned to be executed in state σ tend to abound, as this is the case at every time step when an action other than e is planned to be executed. Thus, we will track instead the occurrence of non-compliant actions with respect to obligations.

Event $\langle \sigma, ca \rangle$ s.t.	$lp(P,\sigma) \models obl(e)$	$lp(P,\sigma) \models obl(\neg e)$
$e \in ca$	compliant	non-compliant
$e \notin ca$	non-compliant	compliant

Table 1
Compliant vs non-compliant situations w.r.t. obligations

Inclezan [5] discussed possible interactions between obligations and authorizations that require further attention from the policy writer, and which were not considered by Gelfond and Lobo. One example is when there is an event $\langle \sigma, ca \rangle$ such that $e \in ca$ and $lp(P,\sigma) \models \{obl(e), \neg permitted(e)\}$ or $lp(P,\sigma) \models \{obl(\neg e), permitted(e)\}$. Such events are called *modality ambiguous* [11], as they reflect an ambiguity that arises at the intersection between two modalities, obligation and authorization. Including a *modality ambiguous* event $\langle \sigma, ca \rangle$ in a plan should be avoided. Such situations can be accounted for by counting how frequently they occur and either prohibiting them altogether or minimizing their occurrence.

4. Behavior Mode Specification Framework for Planning

At the foundation of our behavior mode specification framework lie a series of metrics (i.e., functions) that can be prioritized in different ways by the planning agent.

- the plan **length**, $l(\alpha) = n$
- the **number of modality ambiguous events**, $n\_ma(\alpha)$
- the **number of non-compliant events w.r.t. obligation**, $n\_no(\alpha)$
- the **number of strongly-compliant events w.r.t. authorization**, $n\_sa(\alpha)$
- the **percentage of strongly-compliant events w.r.t. authorization**, $p\_sa(\alpha) =_{def} n\_sa(\alpha)/l(\alpha)$
- the **number of underspecified events w.r.t. authorization**, $n\_ua(\alpha)$
- the **percentage of underspecified events w.r.t. authorization**, $p\_ua(\alpha) =_{def} n\_ua(\alpha)/l(\alpha)$
- the **number of non-compliant events w.r.t. authorization**, $n\_na(\alpha)$
- the **percentage of non-compliant events w.r.t. authorization**, $p\_na(\alpha) =_{def} n\_na(\alpha)/l(\alpha)$.

To allow a human controller to specify preferences between the metrics above, we introduce a language *BMSL* (*Behavior Mode Specification Language*). *Expressions* of *BMSL* have the form:

$$\begin{array}{ll} \textbf{maximize} & m \\ \textbf{minimize} & m \end{array} \tag{6}$$

where m is one of the metrics stated above, $m \in \{l, n\_ma, n\_no, n\_sa, p\_sa, n\_ua, p\_ua, n\_na, p\_na\}$. The first expression is a directive to maximize the value of metric m, while the second is a directive to minimize m. *Statements* of *BMSL* have the form:

$$expr_1 < \cdots < expr_k \tag{7}$$

$$m \circ v \tag{8}$$

where $expr_i$, $1 \leq i \leq k$ are expressions of the form (6), v is a numeric value, and $\circ$ is a comparison operator such that $\circ \in \{=, <, \leq, >, \geq\}$. Statement (7) denotes that $expr\_i$ is preferred to $expr\_j$ (i.e., $expr\_i$ should be prioritized). Statement (8) is a strict requirement for a metric to compare in a specific way to a given value. A *specification S* of *BMSL* is a collection of statements of the form (7) and (8).

In practice, we want *BMSL* specifications to meet certain expectations, which we outline in the definition below.

Definition 5. *Let S be a specification in BMSL. Let $G(S)$ be a directed graph whose nodes are expressions of the type (6) in S and arcs are pairs of the form $(expr_i, expr_j)$ for every statement of the form $expr_i < expr_j$ in S.*

- *S is* **well-defined** *if $G(S)$ is a directed acyclic graph.*
- *If S is well-defined, then by $TS(S)$ we denote the* topological sort *of S (obtained by breaking ties lexicographically if needed).*
- *S is* **consistent** *if it is well-defined and there is no contradiction between statements of the form (8) in S.*

Predefined Behavior Modes

In our framework, we define some built-in behavior modes by providing their description in language *BMSL*. Note that the specifications for all of these predefined behavior modes are consistent according to Definition 5. Other behaviors and preferences can be similarly specified in *BMSL* by the controller of an autonomous agent.

Paranoid Behavior

A *paranoid agent* only selects plans that are strongly-compliant w.r.t. authorization (by requiring that the number of non-compliant and underspecified events w.r.t. authorization, *n_na* and *n_ua* respectively, be 0), compliant w.r.t. obligation, and contain no modality ambiguous events. Among qualifying plans, the plan(s) with minimal length are selected. The *BMSL* specification is:

$\mathbf{S_{paranoid}} =_{def} \{n\_na = 0,\ \ n\_ua = 0,\ \ n\_no = 0,\ \ n\_ma = 0,\ \ \mathbf{minimize}\ l\}$

An agent exhibiting paranoid behavior would select plan α_1 from Example 1, as it is the only plan that contains only strongly-compliant actions w.r.t. authorization.

Cautious Behavior

A *cautious agent* does not accept plans with non-compliant actions. It accepts plans containing underspecified and modality ambiguous events. However it maximizes the percentage of strictly-compliant actions (vs. underspecified) w.r.t. authorization; it minimizes the number of modality ambiguous events; and looks for shortest plans, in this order. Here is the *BMSL* specification for this mode:

$\mathbf{S_{cautious}} =_{def} \{n\_na = 0,\ \ n\_no = 0,\ \ \mathbf{maximize}\ p\_sa < \mathbf{minimize}\ n\_ma < \mathbf{minimize}\ l\}$

A cautious agent would also select plan α_1 from Example 1. While α_2 also meets the imposed constraints, α_1 has a higher percentage of strongly-compliant actions than α_2.

Pragmatic Behavior

A *pragmatic agent* does not accept non-compliant plans either. It accepts plans containing underspecified and modality ambiguous events. Contrary to the cautious agent, a pragmatic agent looks for the shortest plans first, then it maximizes the percentage of strictly-compliant actions (vs. underspecified) w.r.t. authorization, followed by minimizing the number of modality ambiguous events. See the *BMSL* specification here:

$\mathbf{S_{pragmatic}} =_{def} \{n\_na = 0,\ \ n\_no = 0,\ \ \mathbf{minimize}\ l < \mathbf{maximize}\ p\_sa < \mathbf{minimize}\ n\_ma\}$

A pragmatic agent would choose plan α_3 from Example 1 as the optimal plan because of its minimal length.

Lazy Behavior

A *lazy agent* accepts all types of plans, as long as they comply with obligation policies. It prioritizes plans with the smallest percentage of non-compliant actions w.r.t. authorization, then it it looks for shortest plans, and finally it maximizes the percentage of strictly-compliant actions. The *BMSL* specification looks as follows:

$\mathbf{S_{lazy}} =_{def} \{n\_no = 0,\ \ \mathbf{minimize}\ p\_na < \mathbf{minimize}\ l\ <\ \mathbf{maximize}\ p\_sa\}$

A lazy agent would also choose plan α_3 as the best.

Non-Conforming Behavior

A *non-conforming agent* acts as an agent that is not aware or does not intend to comply to policies. It only cares about finding minimal length plans. The *BMSL* specification for this behavior mode is simply:

$\mathbf{S_{non\_conforming}} =_{def} \{\textbf{minimize } l\}$

Although we call this type of behavior *non-conforming*, it may be desirable in certain situations, depending on the goal of the agent. If the agent is tasked to save the life of a human in an emergency situation, it may be preferable for the agent to look for the shortest plan and ignore authorization and obligation policies, if complying with such policies may entail a longer plan that does not guarantee the timeliness of the rescue operation. In other situations, a better name for this type of behavior is *rebel* agents.

5. ASP Plan Selection Based on Behavior Modes

We reduce the problem of finding optimal plans w.r.t. policy behavior modes, to the task of computing answer sets of a logic program. To do so, we expand existing techniques in answer set planning with an encoding of the *AOPL* policy and an encoding of the *BMSL* specification. The resulting logic program consists of the *ASP encodings* of components:

1. dynamic system description D
2. policy P
3. initial state Γ and goal Δ of the planning problem
4. a planning module
5. behavior specification S

We describe each of these components next.

5.1. ASP Encoding of System Description *D*

Following established methods for encoding dynamic system descriptions in ASP, a high-level action language description given in AL_d is translated into ASP by adding predicates $holds(f, i)$ for fluent f and time step i, and $occurs(a, i)$ for action a and time step i. For the example in Figure 1, we have the static relation $adjacent(r_1, r_2)$ saying that rooms r_1 and r_2 are adjacent; the fluent $in(x, r)$ which means that entity/agent x is in room r; and action $enter(x, r)$ which means that x enters room r. Direct effects of actions are encoded by *dynamic causal laws*, as in this example saying that, as an effect of entering a room, the agent will be in that room:

$holds(in(X, R), I + 1) \quad \leftarrow \quad occurs(enter(X, R), I)$

Executability conditions are rules that specify when an action cannot be performed. For example, we can say that X cannot enter a room in which they currently are, nor one that is not adjacent to their current location via:

$\neg occurs(enter(X, R), I) \quad \leftarrow \quad holds(in(X, R_1), I), \quad \neg adjacent(R, R_1)$

We specify relationships between fluents via *state constraints*, e.g., the agent is in one room at a time:

$\neg holds(in(X, R), I) \leftarrow holds(in(X, R_1), I), R \neq R_1$

The ASP encoding of the system description will also include standard rules to describe that fluents that are not affected directly or indirectly by actions maintain their previous values (Inertia Axioms).

5.2. ASP Encoding of Policy *P*

The ASP encoding lp of policy P follows the approach briefly outlined in Section 2, but extends all literals (with the exception of those obtained from static properties of dynamic system D) with an additional parameter i for time step. This is necessary in order to be able to reason over paths/trajectories in the

dynamic system, where an action may be permitted at one time step, but not at a later one. Thus, the rules in (3)-(5) are re-written as:

$$\begin{array}{rcl} permitted(e,I) & \leftarrow & lp(cond,I),\ \text{not } ab(d,I),\ \ \text{not } \neg permitted(e,I) \\ \neg permitted(e,I) & \leftarrow & lp(cond,I),\ \text{not } ab(d,I),\ \ \text{not } permitted(e,I) \\ ab(d_j,I) & \leftarrow & lp(cond_i,I) \end{array}$$

Similarly for other rules of P.

5.3. ASP Encoding of Initial State Γ and Goal Δ

As in answer set planning, an initial state Γ is encoded in ASP via facts of the form

$$\begin{array}{rl} holds(f,0) & \text{for every fluent } f \in \Gamma \\ \neg holds(f,0) & \text{for every fluent literal } \neg f \in \Gamma \end{array}$$

The goal $\Delta = \{f_1, \ldots, f_m, \neg f_{m+1}, \ldots, \neg f_n\}$ is encoded in ASP as the fact:

$$goal(I) \quad \leftarrow \quad holds(f_1,I), \ldots, holds(f_m,I),\ \ \neg holds(f_{m+1},I), \ldots, \neg holds(f_n,I)$$

5.4. ASP Planning Module

A planning module starts by defining a *horizon* (see constant n below), which means a maximum number of time steps by which we want the goal to be met, and specifying that time steps range from 0 to n:

$\#\text{const } n = 10$

$step(0..n)$

Next, *success* is defined as achieving the goal, and it is required to be reached:

$$\begin{array}{rcl} success & \leftarrow & goal(I) \\ & \leftarrow & \text{not } success \end{array}$$

The agent is required to execute an action at each time step via a choice rule:

$$1\{occurs(A,I) : agent\_action(A)\}1 \quad \leftarrow \quad step(I)$$

When planning, an agent can only make decisions about its own actions. We specify these by introducing sort $agent(ag)$ (ag is our agent) and sort $agent\_action(a)$ (a is an action of our agent that can be used in planning). For instance, we define which *enter* actions belong to our agent as follows:

$$agent\_action(enter(X,R)) \quad \leftarrow \quad agent(X),\ \ room(R)$$

Furthermore, in order to be able to compare plans based on their length, we use an approach similar to that outlined by Son and Pontelli [20] (see action *noop*). We introduce an agent action called *wait* that has no effect on the state of the dynamic system, by adding the fact:

$agent\_action(wait)$.

We also require that *wait* actions can only be scheduled at the end of a plan (after the goal was met) and not simultaneously with non-*wait* agent actions:

$$\begin{array}{rcl} & \leftarrow & occurs(A,I), A \neq wait, goal(I) \\ & \leftarrow & occurs(wait,I),\ \text{not } goal(I) \end{array}$$

5.5. ASP Encoding of Behavior Mode Specification S

To encode a behavior mode specification S, we must first start by encoding the calculation of the metrics described in Section 4. First, we need to introduce new predicates to track each action that is included in the plan in terms of its classification with respect to authorization and obligation policy compliance. Predicates *strg_perm*, *underspec*, and *n_perm* relate to authorization and refer to a planned action that is strongly-compliant, underspecified, and non-compliant, respectively. Predicate *n_obl_compl* means that a planned action is not compliant with respect to obligations. Finally, predicate *mod_ambg* means that the action is modality ambiguous.

$$
\begin{array}{lcl}
strg\_perm(A,I) & \leftarrow & occurs(A,I),\ \ permitted(A,I) \\
underspec(A,I) & \leftarrow & occurs(A,I),\ \ \text{not } permitted(A,I),\ \ \text{not } \neg permitted(A,I) \\
n\_perm(A,I) & \leftarrow & occurs(A,I),\ \ \neg permitted(A,I) \\
n\_obl\_cmpl(A,I) & \leftarrow & occurs(A,I),\ \ obl(\neg A,I) \\
n\_obl\_cmpl(A,I) & \leftarrow & \text{not } occurs(A,I),\ \ obl(A,I) \\
mod\_ambg(A,I) & \leftarrow & occurs(A,I),\ \ obl(A,I),\ \ \neg permitted(A,I) \\
mod\_ambg(A,I) & \leftarrow & occurs(A,I),\ \ obl(\neg A,I),\ \ permitted(A,I)
\end{array}
$$

Note that we are able to define predicates *strong_perm*, *underspec*, and *not_perm* as in the rules above because of the restrictions we made w.r.t. policy P in Section 3 where we required that it should be categorical (i.e., $lp(P,\sigma)$ has exactly one answer set for every state σ).

Next, we show how we calculate the metrics in Section 4 using aggregates of Clingo. We start by introducing a new metric, l, representing the plan length. If n is the horizon of the planning problem and N_1 represents the number of *wait* actions in the plan, then l is $n+1-N_1$. The rule below uses the aggregate *#count* of Clingo to count the number of time steps I at which action *wait* occurs.

$$
l(N) \ \leftarrow \ \#count\{I \ : \ occurs(wait,I)\} = N_1,\ \ N = n + 1 - N_1
$$

Similarly, we calculate the other metrics as follows:

$$
\begin{array}{lcl}
n\_ma(N) & \leftarrow & \#count\{A,I \ : \ mod\_ambg(A,I)\} = N \\
n\_no(N) & \leftarrow & \#count\{A,I \ : \ n\_obl\_cmpl(A,I)\} = N \\
n\_sa(N) & \leftarrow & \#count\{A,I \ : \ strg\_perm(A,I)\} = N \\
n\_ua(N) & \leftarrow & \#count\{A,I \ : \ underspec(A,I)\} = N \\
n\_na(N) & \leftarrow & \#count\{A,I \ : \ n\_perm(A,I)\} = N \\
p\_sa(N) & \leftarrow & n\_sa(N_1),\ l(N_2), N = (N_1 * 100)/N_2 \\
p\_ua(N) & \leftarrow & n\_ua(N_1),\ l(N_2), N = (N_1 * 100)/N_2 \\
p\_na(N) & \leftarrow & n\_na(N_1),\ l(N_2), N = (N_1 * 100)/N_2
\end{array}
$$

To model a *BMSL* specification S, we take advantage of the topological sort $TS(S)$ from Definition 5. If S is a well-defined specification, then $TS(S)$ is an ordering $\langle expr_1, \ldots, expr_n \rangle$ of expressions in S from the highest priority to least priority (ties are assumed to be broken lexicographically when needed). We translate an expression $expr_i$ of the form "**maximize** m" into ASP as:

$$
\#maximize\{N@j \ : \ m(N)\}
$$

where m is one of the metrics outlined in Section 4, $m \in \{l, n\_ma, n\_no, n\_sa, p\_sa, n\_ua, p\_ua, n\_na, p\_na\}$, and $j = n + 1 - i$. This statement maximizes the metric m and assigns priority j to this optimization request. Statements with higher priorities take precedence over the ones with lower priorities. Similarly for "**minimize** m":

$$
\#minimize\{N@j \ : \ m(N)\}
$$

Statements of S of the type "$m \circ v$" where $\circ$ is a comparison operator, are translated as constraints:

$$
\leftarrow m(N), \text{not } N \circ v
$$

For example, the translation of specification $\mathbf{S_{cautious}}$ into ASP looks as follows:

$$
\begin{array}{l}
\leftarrow n\_na(N), \text{not } N = 0 \\
\leftarrow n\_no(N), \text{not } N = 0 \\
\#maximize\{N@3 \ : \ p\_sa(N)\} \\
\#minimize\{N@2 \ : \ n\_ma(N)\} \\
\#minimize\{N@1 \ : \ l(N)\}
\end{array}
$$

When putting together the ASP encoding of the components described in Section 5, we obtain a logic program $\Pi(D,\Gamma,\Delta,P,S)$. Answer sets of this program represent optimal solutions to the planning problem $\langle D,\Gamma,\Delta \rangle$ with respect to the behavior mode S and policy P.

6. Experimental Results

We empirically evaluated our implementation on a couple of domains that are more elaborate relative to Example 1.

Traffic Norms Domain: The first domain captures some of the traffic regulations and may be pertinent to a self-driving car. We consider a (section) of a city with a grid street plan, different traffic signs (*do not enter, stop sign*), traffic lights, speed limits, school buses, and pedestrians crossing. Some of the policies in this domain are: *The agent is obligated to stop if pedestrians are crossing. Normally, the agent is permitted to drive if the traffic light is green or yellow, but it is obligated to stop if it is red. In areas where the speed limit is less than 55 mph, the agent cannot surpass the speed limit by more than 5 mph.*

As an example, the last policy mentioned above is written in *AOPL* as:

$$r1(L1, L2, S, S1): \quad \textbf{normally}\ \neg permitted(drive(L1, L2, S))$$
$$\textbf{if}\ \ speed\_limit(L1, L2, S1), S1 < 55, S > S1 + 5$$

The complete description of the domain, the full collection of policies, and experimental details are available at: https://tinyurl.com/5n7ayekf.

We tested the ASP implementation of our framework on eleven planning scenarios. For each scenario, we ran the planning agent in the five different modes listed in Section 4 for paranoid, cautious, pragmatic, lazy, and non-conforming agents. Table 2 shows average times and standard deviation over 10 runs, as well as the length of the optimal plan. All experiments were performed on a machine with an Intel(R) Core(TM) i5-1335U CPU 1.30 GHz RAM 8GB.

Table 2
Experimental results for the *Traffic Norms* domain (*T* - average time in s; *SD* -standard deviation; *L* - optimal plan length)

	Paranoid		**Cautious**			**Pragmatic**			**Lazy**			**Non-Conforming**		
Scenario #	*T* (s)	*SD*	*T* (s)	*SD*	*L*	*T* (s)	*SD*	*L*	*T* (s)	*SD*	*L*	*T* (s)	*SD*	*L*
1	2.1	0.3	2.2	0.2	6	2.5	0.8	6	2.9	0.7	6	3.1	1.1	4
2	3.3	0.7	3.2	0.6	2	2.8	0.5	2	3.7	1.0	2	2.8	0.7	2
3	3.0	0.7	3.0	0.7	4	2.7	0.4	4	2.8	0.4	4	2.6	1.0	4
4	2.4	0.9	4.3	0.5	3	4.7	0.5	3	3.8	0.9	3	3.8	0.3	2
5	4.3	0.6	4.5	0.5	6	4.0	0.4	6	4.6	0.5	6	4.4	0.5	5
6	4.2	0.5	4.5	0.3	6	4.3	0.4	6	2.0	1.2	6	3.2	0.8	5
7	9.9	1.0	9.6	1.9	6	10.3	0.8	6	10.5	0.7	6	9.9	1.4	5
8	10.4	0.9	10.6	0.8	6	10.5	0.8	6	9.7	2.4	6	8.6	2.0	5
9	10.1	3.2	9.1	3.0	6	6.5	1.4	6	4.0	1.2	6	3.2	1.8	5
10	9.9	3.3	9.7	4.1	6	9.5	3.9	6	8.6	2.4	6	8.3	2.5	5
11	24.7	3.9	30.3	4.0	9	27.6	4.3	9	10.3	8.9	9	6.5	0.6	6

Given the nature of the policies, there were no plans that satisfied the paranoid behavior mode (i.e., plans consisting only of strongly-compliant actions). This is expected in regular domains, where the paranoid mode may not be useful, as policies tend to specify what is not allowed instead of every single action that is permitted. We can envision however domains with high levels of security in which this behavior mode may actually be desirable. In terms of performance, as expected, the non-conforming mode is the fastest, followed by the lazy mode. Generally, computing plans in the pragmatic mode is slightly more efficient than the cautious mode. In fact, while the plans computed for the cautious, pragmatic, and lazy modes have the same plan length, they differ in terms of performance, with the lazy mode being the most efficient overall. We observe that as average time increases in more complex scenarios #9-11, the standard deviation increases as well. We also notice an impact from the range of the driving speeds to select from in different scenarios on the average run time. It is worthy to explore more efficient ways of encoding the domain to reduce the impact on run time.

Discussion: This domain is important because it illustrates the need of simulating different behavior modes as a fundamental step in the policy refinement process, similar to the use of *use cases* by Corapi et al. [21]. Some of the cautious and pragmatic plans, although correct w.r.t. those behavior modes, consist of actions that have the agent drive substantially below the speed limit (e.g., at 5 mph on a street with a 25 mph speed limit), which would cause traffic jams and potentially unsafe actions from other agents in real-life situations. The domain also illustrates the flexibility of the Behavior Mode Specification Language *BMSL*, as it allows introducing new metrics to address this issue, such as real time needed to accomplish an action. New behavior modes, specific to this domain, can be created by the controller by refining the predefined modes, and adding requirements or priorities about total real time to complete the plan (as done in other work we currently have underway).

Rooms Domain: We also tested our framework on a domain proposed by Harders and Inclezan [4], using the same experimental setup as above. This domain assumes that an agent operates in a building with several rooms. Rooms are connected by doors that allow unidirectional access from one room to another. Doors can be locked and unlocked by the agent using either a key or a badge. The agent is located in one of the rooms and wants to get to another room. The agent has information about extreme situations such as an active fire or contamination in a room. The agent may have special protective equipment on or not. A selection of policies in this domain are: *Normally, the agent is obligated not to enter a room where there is an active fire. However, the agent is allowed to enter (i.e., not obligated to not enter) a room in which there is an active fire if it has a special protective equipment on. The agent is not permitted to use its badge more than 3 times.*

Due to space limitations, in Table 3 we show the results for five of the 14 tested scenarios, for the cautious, pragmatic, and non-conforming modes. The selected scenarios are the ones that best indicate the differences between these three behavior modes, as they produce plans of different lengths for different modes. Results for additional scenarios and behavior modes, as well as more details about the domain and its policies, are available at https://tinyurl.com/5n7ayekf (also see Harders [22]).

	Cautious		**Pragmatic**		**Non-Conforming**	
Scenario #	T (s)	L	T (s)	L	T (s)	L
1	2.7	3	2.9	3	2.6	3
3	2.4	8	2.6	6	2.7	5
5	2.9	7	3.1	7	2.9	2
8	2.5	10	2.6	4	2.8	4
14	3.3	6	3.1	4	3.1	4

Table 3
Sample experimental results for the *Rooms* domain (T - average time in s; L - optimal plan length)

Discussion: As expected, optimal plans in the non-conforming mode are the fastest to compute and the shortest in length. The cautious mode tends to find longer plans as it prioritizes strong-compliance w.r.t. authorizations over other factors. This means that sometimes the agent goes out of its way to execute actions that are explicitly known to be compliant (i.e., strongly-compliant actions), as if it was looking to be awarded for good behavior. This happens in scenario #8, for example, in which the agent takes a longer path to the goal only to unlock a door – an action that is deemed strongly-compliant by the policy. We believe this behavior mode is still relevant to explore, especially by policy makers, as it may illustrate a possibility on the spectrum of human behavior w.r.t. policies. In practice, the pragmatic mode seemed to be the closest to general human behavior.

7. Related Work

The closest work to ours is the *APIA* architecture for policy-aware *intentional* agents by Meyer and Inclezan [23]. One substantial difference between their work and ours is that *APIA* agents operate with *activities* instead of simple plans, following the AIA architecture by [24]. By allowing sub-activities,

APIA agents can reason about complex scenarios, for example serendipitous cases, where a goal is achieved because of someone else's actions and thus the agent can drop its current plan. However, *APIA* can only compare plans (or paths) according to the coarse definitions by Gelfond and Lobo [3], and cannot characterize or compare plans that contain a mix of actions at different levels of compliance. Additionally, it does not allow the agent's controller to easily set preferences about what to prioritize (e.g., compliance versus plan length), which we do in our framework. An architecture that allows simulations of an agent's actions when its controller (or the agent itself) is allowed to switch between behavior modes during plan execution, for instance if an emergency arises, is reported by Glaze [25]. Such switches incur a small overhead to the runtime, but contribute to the flexibility of simulating evolving agent behaviors, which is important especially when analyzing human behavior: an agent may start out with a cautious behavior mode but switch to a non-conforming mode when the circumstances involve saving someone's life, for instance. The question of emergency situations in relation to policies was also studied by Alves and Fernández [26] in the context of access control policies. In contrast, our work is based on *AOPL*, which can express not only access control policies (i.e., authorizations), but also obligations, both strict and defeasible, and preferences between policy statements.

Another aspect relevant to our work is answer set planning. A survey paper by Son et al. [19] summarizes the state-of-the-art in this domain and the different special avenues of explored research: classical planning, conformant planning, conditional planning, and planning with preferences. The closest area of research to ours is that of planning with preferences. Son and Pontelli [20] introduced a language *PP* for specifying basic preferences (state desires and goal preferences) and general preferences among basic ones. Our work differs from Son et al.'s in that our preferences are set between different *metrics that are aggregates*, for instance percentages, which is a more complex case. We are not sure that maximization of percentage metrics can be achieved within the *PP* framework. Additionally, our framework is domain-specific, as it focuses on policy compliance, and hence a simpler specification language like the one proposed in Section 4 suffices.

Craven et al. [11] discussed issues that may arise from the analysis of a policy. Some of these are relevant to planning, specifically *modality conflicts* which occur when there are seemingly contradictory statements in the obligation and authorization policy, for instance when an agent is obligated to perform an action that it is not permitted to execute. Craven et al. employ Event Calculus [27] for the description of an agent's changing environment. In our work we use ASP and leverage existing research in the ASP community on representing and reasoning about action and change, policy compliance, planning, and autonomous agents.

8. Conclusions and Future Work

In this work, we introduced a framework that allows the specification of different behavior modes of an autonomous agent in terms of plan selection w.r.t. to policy compliance. We described various metrics that the controller can employ. The framework allows imposing constraints on these metrics and establishing preferences in terms of which metrics should be optimized, according to the attitude towards policy compliance that is meant to be captured. Additionally, new metrics can be included and incorporated in the definition of new behavior modes. We described an implementation of this framework in ASP. Experimental results for non-trivial domains were presented as well. This framework can be useful to policy makers in practice, as a way to run simulations of different types of (human) agents and refine policies based on any unintended consequences observed in such simulations.

In future work, we plan to extend our framework to autonomous *intentional* agents, which are driven by goals and sub-goals associated with activities, by leveraging work by Meyer and Inclezan [23]. Additionally, we intend to explore other metrics that can be added to our framework. For instance, Meyer and Inclezan proposed that less compliant actions should be scheduled later in a plan, as there is a chance that those actions may not need to be executed if the goal is serendipitously met by someone else's actions. Finally, we plan to improve the efficiency of the framework and thank the anonymous reviewers for their suggestions in this direction.

References

[1] M. Gelfond, D. Inclezan, Some properties of system descriptions of AL_d, J. Appl. Non Class. Logics 23 (2013) 105–120. doi:10.1080/11663081.2013.798954.

[2] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents, Cambridge University Press, 2014. doi:10.1017/CBO9781139342124.

[3] M. Gelfond, J. Lobo, Authorization and Obligation Policies in Dynamic Systems, in: M. Garcia de la Banda, E. Pontelli (Eds.), Logic Programming, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2008, pp. 22–36. doi:10.1007/978-3-540-89982-2_7.

[4] C. Harders, D. Inclezan, Plan selection framework for policy-aware autonomous agents, in: S. A. Gaggl, M. V. Martinez, M. Ortiz (Eds.), Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings, volume 14281 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 638–646. doi:10.1007/978-3-031-43619-2_43.

[5] D. Inclezan, An ASP framework for the refinement of authorization and obligation policies, Theory and Practice of Logic Programming (2023) 1–16. doi:10.1017/S147106842300011X.

[6] M. Gelfond, V. Lifschitz, The Stable Model Semantics for Logic Programming, in: Proceedings of the International Conference on Logic Programming (ICLP88), 1988, pp. 1070–1080.

[7] M. Gelfond, V. Lifschitz, Classical Negation in Logic Programs and Disjunctive Databases, New Generation Computing 9 (1991) 365–386. doi:10.1007/BF03037169.

[8] V. W. Marek, M. Truszczynski, Stable models and an alternative logic programming paradigm, in: K. R. Apt, V. W. Marek, M. Truszczynski, D. S. Warren (Eds.), The Logic Programming Paradigm - A 25-Year Perspective, Artificial Intelligence, Springer, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[9] M. Gebser, R. Kaminski, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, S. Thiele, Potassco user guide, 2 ed., University of Potsdam, 2015.

[10] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 input language format, Theory Pract. Log. Program. 20 (2020) 294–309. doi:10.1017/S1471068419000450.

[11] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, A. Bandara, Expressive policy analysis with enhanced system dynamicity, in: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security, ASIACCS '09, Association for Computing Machinery, New York, NY, USA, 2009, p. 239–250. doi:10.1145/1533057.1533091.

[12] D. Corapi, A. Russo, M. De Vos, J. Padget, K. Satoh, Normative design using inductive learning, Theory and Practice of Logic Programming 11 (2011) 783–799. doi:10.1017/S1471068411000305.

[13] T. Pellegrini, G. Havur, S. Steyskal, O. Panasiuk, A. Fensel, V. Mireles-Chavez, T. Thurner, A. Polleres, S. Kirrane, A. Schönhofer, Dalicc: A license management framework for digital assets, 2019.

[14] M. Gelfond, V. Lifschitz, Action languages, Electronic Transactions on AI 3 (1998) 193–210. URL: http://www.ep.liu.se/ej/etai/1998/007/.

[15] V. Lifschitz, Answer set planning, in: D. D. Schreye (Ed.), Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999, MIT Press, 1999, pp. 23–37.

[16] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, A logic programming approach to knowledge-state planning, II: the DLV^K system, Artif. Intell. 144 (2003) 157–211. doi:10.1016/S0004-3702(02)00367-3.

[17] T. C. Son, P. H. Tu, M. Gelfond, A. R. Morales, Conformant planning for domains with constraints-a new approach, in: M. M. Veloso, S. Kambhampati (Eds.), Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA, AAAI Press / The MIT Press, 2005, pp. 1211–1216. URL: http://www.aaai.org/Library/AAAI/2005/aaai05-192.php.

[18] P. H. Tu, T. C. Son, M. Gelfond, A. R. Morales, Approximation of action theories and its application to conformant planning, Artif. Intell. 175 (2011) 79–119. doi:10.1016/j.artint.2010.04.007.

[19] T. C. Son, E. Pontelli, M. Balduccini, T. Schaub, Answer set planning: A survey, Theory Pract. Log.

Program. 23 (2023) 226–298. doi:10.1017/S1471068422000072.
[20] T. C. Son, E. Pontelli, Planning with preferences using logic programming, Theory and Practice of Logic Programming 6 (2006) 559–607. doi:10.1017/S1471068406002717.
[21] D. Corapi, M. D. Vos, J. A. Padget, A. Russo, K. Satoh, Norm refinement and design through inductive learning, in: M. D. Vos, N. Fornara, J. V. Pitt, G. A. Vouros (Eds.), Coordination, Organizations, Institutions, and Norms in Agent Systems VI - COIN 2010 International Workshops, COIN@AAMAS 2010, Toronto, Canada, May 2010, COIN@MALLOW 2010, Lyon, France, August 2010, Revised Selected Papers, volume 6541 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 77–94. doi:10.1007/978-3-642-21268-0_5.
[22] C. Harders, Plan Choosing For Policy-Aware Autonomous Agents, Master's thesis, Miami University, 2023.
[23] J. Meyer, D. Inclezan, APIA: an architecture for policy-aware intentional agents, in: A. Formisano, Y. A. Liu, B. Bogaerts, A. Brik, V. Dahl, C. Dodaro, P. Fodor, G. L. Pozzato, J. Vennekens, N. Zhou (Eds.), Proceedings 37th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021, volume 345 of *EPTCS*, 2021, pp. 84–98. doi:10.4204/EPTCS.345.23.
[24] J. Blount, M. Gelfond, M. Balduccini, A theory of intentions for intelligent agents - (Extended Abstract), in: F. Calimeri, G. Ianni, M. Truszczynski (Eds.), Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings, volume 9345 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 134–142. doi:10.1007/978-3-319-23264-5_12.
[25] S. C. Glaze, Modifying Behavior Modes of Policy-Aware Autonomous Agents, Master's thesis, Miami University, 2024.
[26] S. Alves, M. Fernandez, A graph-based framework for the analysis of access control policies, Theoretical Computer Science 685 (2017) 3–22. doi:10.1016/j.tcs.2016.10.018.
[27] R. Kowalski, M. Sergot, A Logic-Based Calculus of Events, Springer Berlin Heidelberg, Berlin, Heidelberg, 1989, pp. 23–55. doi:10.1007/978-3-642-83397-7_2.