TPLP **26** (3-4): 428–457, 2026. © The Author(s), 2026. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

doi:10.1017/S1471068425100380 First published online 2 January 2026

Autonomous Agents and Policy Compliance: A Framework for Reasoning About Penalties

VINEEL TUMMALA and DANIELA INCLEZAN
Miami University, Oxford, OH, USA
(*e-mails:* tummalvs@miamioh.edu, inclezd@miamioh.edu)

submitted 21 March 2025; revised 7 September 2025; accepted 6 December 2025

Abstract

This paper presents a logic programming-based framework for policy-aware autonomous agents that can reason about potential penalties for noncompliance and act accordingly. While prior work has primarily focused on ensuring compliance, our approach considers scenarios where deviating from policies may be necessary to achieve high-stakes goals. Additionally, modeling noncompliant behavior can assist policymakers by simulating realistic human decision-making. Our framework extends Gelfond and Lobo's Authorization and Obligation Policy Language ($\mathcal{AOPL}$) to incorporate penalties and integrates Answer Set Programming (ASP) for reasoning. Compared to previous approaches, our method ensures well-formed policies, accounts for policy priorities, and enhances explainability by explicitly identifying rule violations and their consequences. Building on the work of Harders and Inclezan, we introduce penalty-based reasoning to distinguish between noncompliant plans, prioritizing those with minimal repercussions. To support this, we develop an automated translation from the extended $\mathcal{AOPL}$ into ASP and refine ASP-based planning algorithms to account for incurred penalties. Experiments in two domains demonstrate that our framework generates higher-quality plans that avoid harmful actions while, in some cases, also improving computational efficiency. These findings underscore its potential for enhancing autonomous decision-making and informing policy refinement.

KEYWORDS: norms, policy, dynamic domains, ASP

1 Introduction

In this paper, we explore autonomous agents operating in dynamic environments governed by *policies* or *norms*, including cultural conventions and regulations. Our focus is on agents whose knowledge bases and reasoning algorithms are encoded in logic programming, specifically Answer Set Programming (ASP) (Gelfond and Lifschitz 1991; Marek and Truszczynski 1999). We introduce a framework that enables these policy-aware agents to assess potential penalties for noncompliance and generate suitable plans for their goals.

Research on norm-aware autonomous agents predominantly focuses on compliance (e.g., (Oren *et al.* 2011; Alechina *et al.* 2012)). However, studying noncompliant agents is equally important for two key reasons. First, autonomous agents may be tasked with

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

high-stakes objectives (e.g., assisting in rescue operations) that can only be achieved through selective noncompliance. In such cases, identifying optimal noncompliant plans that accomplish the goal while minimizing repercussions is crucial. Second, our framework can support policymakers by enabling policy-aware agents to model human behavior. Humans do not always adhere to norms and often seek to minimize penalties for noncompliance. By simulating different compliance attitudes, policymakers can identify potential weaknesses in policies and refine them accordingly.

To enable autonomous agents to reason about policies, evaluate penalties for noncompliance, and generate optimal plans based on their circumstances and given norms, we must first encode these policies. In our proposed framework, policies are specified in the *Authorization and Obligation Policy Language* ($\mathcal{AOPL}$) by Gelfond and Lobo (2008), whose semantics are defined via a translation into ASP. We expand $\mathcal{AOPL}$ to enable the representation of, and reasoning about, penalties that may be incurred for noncompliance with a policy. We develop an automated translation of the extended $\mathcal{AOPL}$, referred to as $\mathcal{AOPL}$-$\mathcal{P}$, into ASP and refine ASP-based planning algorithms to incorporate penalty considerations.

Using a high-level language such as $\mathcal{AOPL}$ for representing policies and penalties offers several advantages over alternative approaches, which may include encoding policies as soft constraints in ASP (Calimeri *et al.* 2020). One key benefit is that $\mathcal{AOPL}$ ensures the policy is well-structured, with built-in mechanisms to detect issues like inconsistencies, ambiguities, and underspecification (Inclezan 2023). Additionally, $\mathcal{AOPL}$'s syntax and semantics support the representation of priorities between policy statements, allowing for the determination of which statements apply at different points in time. Moreover, our $\mathcal{AOPL}$-based approach enhances explainability by identifying which policy rules an agent violated and how these violations led to accumulated penalties – something that would be difficult to achieve using soft constraints.

In previous work on policy-aware autonomous agents, Harders and Inclezan (2023) introduced *behavior modes* to capture different attitudes toward policy compliance, allowing noncompliant actions within the Risky mode. While this framework established a foundation for reasoning about noncompliance, it ranked plans in the Risky mode solely by their length, without accounting for the number or proportion of noncompliant actions or the severity of the violated rules. However, other behavior modes – excluding noncompliant ones – did consider the proportion of explicitly known compliant actions, for example.

In this work, we extend that approach by introducing a more nuanced evaluation of noncompliance, enabling agents to weigh trade-offs between policy violations and their objectives more effectively. Our framework distinguishes between noncompliant plans using penalties, allowing agents to achieve their goals while minimizing repercussions. Experimental results show that our approach generates higher-quality plans. For example, in a Traffic Norms domain, it selects an optimal driving speed and avoids actions potentially harmful to humans. Such factors are overlooked by the previous framework, which accounts only for plan length while ignoring execution time (and thus driving speed) and gives no special consideration to preventing human harm. Additionally, for certain domains, our method demonstrates improved efficiency. A preliminary version of this work appears in (Tummala and Inclezan 2024).

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

The rest of the paper starts with background information in Section 2, followed by a motivating example in Section 3. We present our framework in Section 4, which includes the extension of $\mathcal{AOPL}$ with penalties ($\mathcal{AOPL}$-$\mathcal{P}$), an automated translator from $\mathcal{AOPL}$-$\mathcal{P}$ to ASP, penalty-aware planning, and revised behavior modes. We present experimental results in Section 5. We discuss related work in Section 6 and end with conclusions and future work in Section 7.

2 Background

We now provide background on the norm-specification language $\mathcal{AOPL}$ and our previous work on policy-aware planning agents. We assume readers are familiar with ASP or can consult external resources (e.g., (Gelfond and Lifschitz 1991; Gelfond and Kahl 2014; Calimeri *et al.* 2020)) as needed.

2.1 *Policy-specification language* $\mathcal{AOPL}$

Gelfond and Lobo (2008) designed the Authorization and Obligation Policy Language $\mathcal{AOPL}$ for specifying policies for an intelligent agent acting in a dynamic environment. A policy is a collection of authorization and obligation statements. An *authorization* indicates whether an agent's action is permitted or not, and under which conditions. An *obligation* specifies whether an agent is required or not required to perform a particular action or to abstain from it under given conditions. An $\mathcal{AOPL}$ policy assumes that the agent's environment is described using an action language, a high-level language designed to concisely and accurately represent action preconditions as well as the direct and indirect effects of actions. Over time, various action languages have been developed (e.g., $\mathcal{A}$ (Gelfond and Lifschitz 1993), $\mathcal{B}$ (Gelfond and Lifschitz 1998), $\mathcal{C}$ (Giunchiglia *et al.* 2004), $\mathcal{H}$ (Chintabathina *et al.* 2005), $\mathcal{AL}_d$ (Gelfond and Inclezan 2013), etc.), including modular action languages such as MAD (Lifschitz and Ren 2006) and $\mathcal{ALM}$ (Inclezan 2012). They all incorporate means for dealing with the ramification and qualification problems, as well as the law of inertia (i.e., domain properties not affected by actions remain unchanged).

A system description written in an action language defines the domain's transition diagram whose states are complete and consistent sets of static (i.e., immutable) and fluent literals (i.e., properties of the domain that may be changed by actions), and whose arcs are labeled by actions. The signature of the dynamic system description (which includes predicates denoting *sorts*, *statics*, *fluents*, and *actions*) is included in the signature of an $\mathcal{AOPL}$ policy for that dynamic domain. Additionally, the signature of an $\mathcal{AOPL}$ policy includes predicates *permitted* for authorizations, *obl* for obligations, and *prefer* for specifying preferences between authorizations or between obligations. A *prefer* atom is created from the predicate *prefer*; similarly, for *permitted* and *obl* atoms. A literal is an atom or its negation.

An $\mathcal{AOPL}$ *policy* $\mathcal{P}$ is a finite collection of statements of the form:

$$permitted\,(e) \quad \textbf{if} \;\; cond \tag{1a}$$

$$\neg permitted\,(e) \quad \textbf{if} \;\; cond \tag{1b}$$

$$obl\,(h) \quad \textbf{if} \;\; cond \tag{1c}$$

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

$$\neg obl\,(h) \quad \textbf{if } cond \tag{1d}$$
$$d : \textbf{normally } permitted(e) \quad \textbf{if } cond \tag{1e}$$
$$d : \textbf{normally } \neg permitted(e) \quad \textbf{if } cond \tag{1f}$$
$$d : \textbf{normally } obl(h) \quad \textbf{if } cond \tag{1g}$$
$$d : \textbf{normally } \neg obl(h) \quad \textbf{if } cond \tag{1h}$$
$$prefer(d_i, d_j) \tag{1i}$$

where e is an elementary action; h is a happening (i.e., an elementary action or its negation[1]); *cond* is a set of literals of the signature, except for *prefer* literals; d appearing in (1e)–(1h) denotes a defeasible rule label; and d_i, d_j in (1i) refer to distinct *defeasible* rule labels from $\mathcal{P}$. Rules (1a)–(1d) encode *strict* policy statements, while rules (1e)–(1h) encode *defeasible* statements (i.e., statements that may have exceptions). Rule (1i) captures *priorities* between defeasible statements only. It specifies that a defeasible rule labeled d_i overrides a defeasible rule labeled d_j, rendering the latter inapplicable when the condition of the former (i.e., of d_i) is satisfied. Strict rules, whenever their condition *cond* is satisfied, always override the defeasible rules they conflict with. Unlike deontic logic, the $\mathcal{AOPL}$ language described by Gelfond and Lobo does not assume an equivalence between rules for $\neg permitted(e)$ and $obl(\neg e)$. We believe this choice was made to allow for different interpretations and to accommodate other types of logics. Such an equivalence can be implemented by adding the following rules to an $\mathcal{AOPL}$ policy $\mathcal{P}$:

$$\neg permitted(e) \quad \textbf{if } obl(\neg e)$$
$$obl(\neg e) \quad \textbf{if } \neg permitted(e)$$

However, the lack of a clear relationship between permissions and obligations can give rise to *modality conflicts*, a term introduced by Craven *et al.* (2009), as noted by Inclezan (2023). For example, an $\mathcal{AOPL}$ policy may derive both $\neg permitted(e)$ and $obl(e)$ for the same elementary action e in a given state, which is an undesirable outcome, as it appears contradictory.

The semantics of an $\mathcal{AOPL}$ policy determine a mapping $\mathcal{P}(\sigma)$ from states of a transition diagram $\mathcal{T}$ into a collection of *permitted* and *obl* literals, obtained from the policy statements that are applicable in state σ (i.e., have a satisfied *cond* and are not overridden). To formally describe the semantics of $\mathcal{AOPL}$, a translation of a policy $\mathcal{P}$ and a state σ of the transition diagram into ASP is defined as $lp(\mathcal{P}, \sigma)$. Properties of an $\mathcal{AOPL}$ policy $\mathcal{P}$ are defined in terms of the answer sets of the logic program $lp(\mathcal{P}, \sigma)$ expanded with appropriate rules. Gelfond and Lobo define a policy as *consistent* if, for every state σ of $\mathcal{T}$, the logic program $lp(\mathcal{P}, \sigma)$ is consistent (i.e., has an answer set). A policy is *categorical* if $lp(\mathcal{P}, \sigma)$ has *exactly one answer set* for every state σ of $\mathcal{T}$.

In our work, we adopt established definitions for classifying events as strongly-compliant (i.e., explicitly permitted), underspecified (i.e., neither explicitly permitted nor explicitly prohibited), or noncompliant (i.e., explicitly prohibited) with respect to authorizations, and as compliant or noncompliant with respect to obligations. Formal definitions for these concepts, adapted from Gelfond and Lobo (2008) and Inclezan (2023),

[1] If $obl(\neg e)$ is true, then the agent must not execute e.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

are provided below. Note that *ca* denotes a compound action, while *e* refers to an elementary action. An event $\langle \sigma, ca \rangle$ is a pair consisting of a state σ and a (possibly compound) action *ca* occurring in that state. If l is a literal, then $lp(\mathcal{P}, \sigma) \models l$, read as "the logic program $lp(\mathcal{P}, \sigma)$ entails l," denotes that l belongs to every answer set of $lp(\mathcal{P}, \sigma)$. Similarly, $lp(\mathcal{P}, \sigma) \not\models l$ read as "the logic program $lp(\mathcal{P}, \sigma)$ does not entail l," denotes that l does not appear in any of the answer sets of $lp(\mathcal{P}, \sigma)$.

Definition 1
(Compliance for Authorizations).

- *An event* $\langle \sigma, ca \rangle$ *is strongly-compliant with respect to the authorizations in policy* $\mathcal{P}$ *if, for every* $e \in ca$, *we have that* $lp(\mathcal{P}, \sigma) \models permitted(e)$.
- *An event* $\langle \sigma, ca \rangle$ *is underspecified with respect to the authorizations in policy* $\mathcal{P}$ *if, for every* $e \in ca$, *we have that* $lp(\mathcal{P}, \sigma) \not\models permitted(e)$ *and* $lp(\mathcal{P}, \sigma) \not\models \neg permitted(e)$.
- *An event* $\langle \sigma, ca \rangle$ *is noncompliant with respect to the authorizations in policy* $\mathcal{P}$ *if, for every* $e \in ca$, *we have that* $lp(\mathcal{P}, \sigma) \models \neg permitted(e)$.

Definition 2
(Compliance for Obligations). *An event* $\langle \sigma, ca \rangle$ *is compliant with respect to the obligations in policy* $\mathcal{P}$ *if*

- *For every e such that* $lp(\mathcal{P}, \sigma) \models obl(e)$ *we have that* $e \in ca$, *and*
- *For every e such that* $lp(\mathcal{P}, \sigma) \models obl(\neg e)$ *we have that* $e \notin ca$.

Definition 3
(Compliance for Authorization and Obligations). *An event* $\langle \sigma, ca \rangle$ *is strongly compliant with arbitrary policy* $\mathcal{P}$ *(that may contain both authorizations and obligations) if it is strongly compliant with the authorization component and compliant with the obligation component of* $\mathcal{P}$.

Inclezan (2023) showed that in categorical (i.e., unambiguous) policies, events $\langle \sigma, ca \rangle$ such that *ca* consists of a single elementary action can be categorized with respect to authorization as either strongly-compliant (i.e., explicitly permitted), noncompliant (i.e., explicitly prohibited), or underspecified (i.e., neither explicitly permitted nor explicitly prohibited). In the case of noncategorical consistent policies, there will be events that lie outside of this categorization (i.e., do not fit in any of these three categories). A typical example of a noncategorical consistent policy is one that consists of two defeasible rules, both applying in a state σ, one deriving $permitted(e)$ and the other deriving $\neg permitted(e)$ with no preference stated between them. In such a case, $lp(\mathcal{P}, \sigma)$ has two answer sets where one answer set contains $permitted(e)$, while the other contains $\neg permitted(e)$; hence none of the categories in Definition 1 apply to event $\langle \sigma, e \rangle$. Similarly, in the case of compound actions (i.e., when the agent executes more than one action at a time). A compound action may include elementary actions belonging to different categories (e.g., one compliant and another underspecified), and thus the compound action itself does not fit into any single category.

In what follows, we assume categorical policies. Handling noncategorical policies is nontrivial and left for future work. However, a well-defined policy is generally expected to be categorical, while a noncategorical (i.e., ambiguous) policy typically reflects a flaw in its specification or design.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

2.2 Agent behavior modes with respect to policy compliance

In previous work, Harders and Inclezan (2023) introduced an ASP framework for plan selection in policy-aware autonomous agents, where policies were specified in $\mathcal{AOPL}$. They proposed that agents could adopt different attitudes toward norm compliance, influencing the selection of the "best" plan. These attitudes, termed *behavior modes*, were defined using various metrics to capture different compliance strategies. These metrics included plan length as well as the number and percentage of different types of *elementary* actions: strongly compliant actions (explicitly permitted), underspecified actions (neither explicitly permitted nor prohibited), and noncompliant actions (violating authorizations or obligations).

The following predefined agent behavior modes were introduced by Harders and Inclezan:

- Safe Behavior Mode – prioritizes actions that are explicitly known to be compliant (i.e., maximizes the percentage of strongly-compliant elementary actions first and then plan length) and does not execute noncompliant actions;
- Normal Behavior Mode – prioritizes plan length and then actions explicitly known to be compliant (i.e., minimizes plan length first and then maximizes the percentage of strongly-compliant elementary actions), while not executing noncompliant actions; and
- Risky Behavior Mode – disregards policies, but does not go out of its way to be noncompliant either. This may result in the inclusion of noncompliant actions if they contribute to minimizing plan length.

Harders and Inclezan (2023) encoded these behavior modes in the ASP-variant that constitutes the input language for the Clingo solver (Gebser *et al*. 2019) using constraints, employing the $\#maximize$ and $\#minimize$ constructs to express priorities. For instance, in the Safe behavior mode, the planning module included the following rules:

$$\#maximize\ \{N@2\ :\ p_sa(N)\}$$
$$\#minimize\ \{N@1\ :\ l(N)\}$$
$$\leftarrow n_na(N),\ \text{not}\ N=0$$
$$\leftarrow n_no(N),\ \text{not}\ N=0$$

where p_sa is the percentage of strongly-compliant elementary actions (with respect to authorizations); l is the length of the plan; n_na and n_no denote the number of noncompliant elementary actions with respect to authorizations and obligations, respectively.

3. Motivating example: Traffic norms domain

To illustrate the need for penalties in enabling more nuanced planning, especially allowing certain levels of noncompliance in high-stakes situations, consider a dynamic domain where a self-driving agent navigates a simplified city environment. There are certain norms that the self-driving agent must be aware of when driving, which may represent traffic regulations or cultural conventions. We limit ourselves to one agent, a few traffic

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

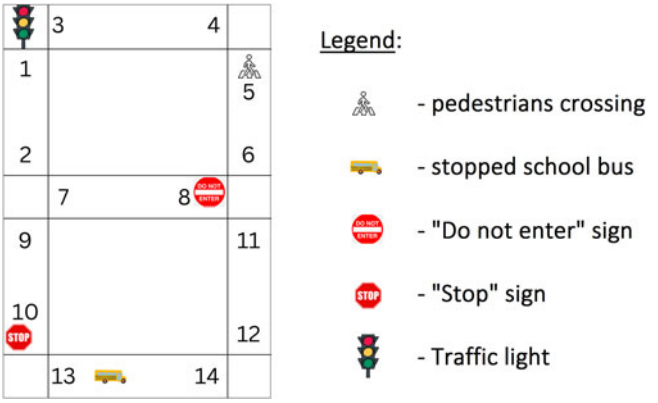


Fig 1. Layout of the traffic norms domain.

signs, and a grid street layout. A schematic view of this *Traffic Norms Domain* is in Figure 1.

To model this dynamic domain, we consider fourteen locations labeled from 1 to 14; a set of driving speeds; traffic light colors red, yellow, and green; and two traffic signs "Stop" and "Do not enter." One example of a fluent in this domain is $pedestrians_are_crossing(L)$ saying that people are crossing the street at location L. The agent can execute actions: $drive(L_1, L_2, S)$ to drive between two (connected) locations L_1 and L_2 at a speed $S > 0$; and $stop(L)$ to stop at L.

We consider the set of policies in Figure 2. Note that some of these rules represent cultural norms (e.g., rules 1 and 2), while others reflect traffic regulations (e.g., rule 5). For illustration purposes, we consider a system of penalties on a scale from 1 to 3 and discuss potential issues with this choice in Section 4.7. In practice, such penalties would be determined by experts in ethics and traffic regulation; here, we adopt a 3-point scale solely to illustrate the framework. Policy rules 7 and 8 do not have penalties associated with them, as they describe permissions.

Harders and Inclezan (2023) introduced the Risky agent behavior mode for emergency situations. Agents in this mode only look for the shortest plan while ignoring policies. As a result, a Risky agent in the Traffic Norms domain who starts in location 6 and needs to get to location 1, where the speed limit between locations 2 and 1 is 45 mph, may come up with several plans (we don't specify driving speeds unless relevant to our discussion), including:

1. Drive from 6 to 5, from 5 to 4 without stopping for the pedestrians crossing the street, then from 4 to 3, and finally from 3 to 1 (possibly on a red light).
2. Drive from 6 to 8 and thus enter a "Do not enter" street, drive from 8 to 7, from 7 to 2, and finally from 2 to 1 at 65 mph.
3. Same as plan 2, but drive from 2 to 1 at 45 mph.

According to the definition of Risky agents, all these plans are treated as equivalent, as they consider only plan length without factoring in the severity of infractions, such as the number or gravity of noncompliant actions. Instead, we aim to distinguish between plans containing noncompliant actions and prioritize those that incur the least penalty.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

1. The agent is constrained by a defeasible rule that does *not permit* exceeding the speed limit by more than 5 mph if the speed limit is under 55 mph.
   Penalty: 1-point penalty if exceeding the speed limit by less than 10 mph; 2-point if exceeding the speed limit by 10 to 19 mph; and a 3-point penalty is imposed if exceeding the speed limit by 20 mph or more.
2. The agent is constrained by a defeasible rule that does *not permit* exceeding the speed limit by more than 10 mph when the speed limit is 55 mph or higher.
   Penalty: a 2-point penalty if exceeding the speed limit by 10 to 19 mph, and a 3-point penalty if exceeding the speed limit by 20 mph or more.
3. The agent is strictly *obligated not* to enter the roads marked as "Do not enter."
   Penalty: 3-point penalty.
4. The agent is strictly *not permitted* to roll over a stop sign.
   Penalty: 2-point penalty.
5. The agent is strictly *obligated not* to move when a school bus is stopped.
   Penalty: 3-point penalty. (*Note: we will refine this penalty in Section 4.7.*)
6. The agent is strictly *obligated* to stop when pedestrians are crossing.
   Penalty: 3-point penalty. (*Note: we will refine this penalty in Section 4.7.*)
7. The agent is *permitted* by a defeasible rule to proceed through an intersection when the traffic light turns green.
8. The agent is *permitted* by a defeasible rule to drive when the traffic light is yellow.
9. The agent is strictly *obligated not* to cross an intersection on a red traffic light.
   Penalty: 3-point penalty.
10. Rule 1 overrides rules 7 and 8.
11. Rule 2 overrides rules 7 and 8.

Fig 2. Policies and penalties for the traffic norms domain.

In scenarios where a fully compliant plan is unavailable, an agent may need to choose a noncompliant plan to achieve a high-stakes goal. To enable this, we introduce penalties as a means of guiding plan selection.

4 Penalization framework for policy-aware agents

In our framework, we assume that penalties for a domain are set by domain experts in collaboration with specialists in ethics. We consider penalties that are specified as numbers on a given scale. For illustration purposes, we start by considering a scale from 1 to 3, where a 3-point penalty corresponds to situations with a high gravity. We also account for interactions between penalties and other planning-relevant metrics that should be optimized, such as the total execution time of a plan (as opposed to plan length).

In the remainder of this section, we present the implementation of our framework. We first introduce an adaptation of the $\mathcal{AOPL}$ version first presented by Inclezan (2023), which we refer to as $\mathcal{AOPL}'$. We then extend $\mathcal{AOPL}'$ with penalties, resulting in $\mathcal{AOPL}$-$\mathcal{P}$ (i.e., $\mathcal{AOPL}$ with penalties), and expand the corresponding ASP translation. An automated translator from $\mathcal{AOPL}$-$\mathcal{P}$ into ASP has been implemented and is described next. We then show how reasoning about penalties in planning is handled in our framework, followed by the incorporation of additional metrics, in particular plan execution time.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

We also revisit behavior modes in this context and discuss a key refinement needed to prevent harm to humans. The section concludes with a high-level overview of the framework.

4.1 $\mathcal{AOPL}'$ and its ASP translation

In our work, we build on the $\mathcal{AOPL}$ version introduced by Inclezan (2023), which assumes that all rules – including strict ones – are labeled, unlike in the original definition of the language by Gelfond and Lobo (see Section 2.1). While Gelfond and Lobo assigned labels only to defeasible rules to allow expressing preferences among them, we require labels on both strict and defeasible rules to track which policy rules are violated by an agent. The distinction between defeasible and strict rules specifies only whether a rule can have exceptions to its applicability. It does not determine which rules an agent may violate, as both strict and defeasible applicable rules can be violated. We further extend this version of $\mathcal{AOPL}$ by refining its semantics to support reasoning over trajectories in dynamic systems, a key requirement for planning. We refer to this refined version as $\mathcal{AOPL}'$.

In $\mathcal{AOPL}'$, rules of a policy may have one of the forms below:

$$r: \quad permitted\,(e) \textbf{ if } cond \tag{2a}$$

$$r: \neg permitted\,(e) \textbf{ if } cond \tag{2b}$$

$$r: \quad obl\,(h) \textbf{ if } cond \tag{2c}$$

$$r: \quad \neg obl\,(h) \textbf{ if } cond \tag{2d}$$

$$d: \textbf{normally} \quad permitted(e) \textbf{ if } cond \tag{2e}$$

$$d: \textbf{normally} \quad \neg permitted(e) \textbf{ if } cond \tag{2f}$$

$$d: \textbf{normally} \quad obl(h) \textbf{ if } cond \tag{2g}$$

$$d: \textbf{normally} \quad \neg obl(h) \textbf{ if } cond \tag{2h}$$

$$prefer(d_i, d_j) \tag{2i}$$

As in the original $\mathcal{AOPL}$ language, the rules of $\mathcal{AOPL}'$ use the following notation: e denotes an elementary action, h a happening (i.e., an elementary action or its negation), and *cond* a set of literals from the signature, excluding those containing the predicate *prefer*. Rules (2a)–(2d) represent strict policy rules, now labeled with a label denoted by r. Defeasible rules of types (2e)–(2h) are likewise labeled, as in the earlier version, and contain the keyword **normally**. As before, preference relationships such as rule (2i) can be specified only between defeasible rules, while strict rules always override the defeasible rules with which they conflict.

The semantics of $\mathcal{AOPL}'$ are given in terms of a translation into ASP. We adapt the translation defined by Inclezan (2023) in two ways: first, by introducing discrete time steps to reason over trajectories of multiple steps for planning purposes; and second, by modifying the original translation from the Clingo input language to the ASP-Core-2 standard input language for ASP (Calimeri *et al.* 2020), thereby ensuring compatibility with other solvers such as DLV2 (Alviano *et al.* 2017). This translation is denoted by $rei_lp(\mathcal{P})$ for an $\mathcal{AOPL}'$ policy $\mathcal{P}$.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

The **signature** of $rei_lp(\mathcal{P})$ for a policy $\mathcal{P}$ applying in a dynamic domain described by a transition diagram $\mathcal{T}$ contains the sorts, statics, fluents, and actions of $\mathcal{T}$; predicates *action*, *static*, and $fluent$; the sort *step* representing time steps; a predicate $holds(s)$ for every static s; and a predicate $holds(f, i)$ for every fluent f and time step i.

To simplify the presentation of other components of the signature of $rei_lp(\mathcal{P})$, we generalize the syntax of $\mathcal{AOPL}'$ rules of type (2a)–(2h) as:

$$r : [\textbf{normally}]\ hd\ \textbf{if}\ cond \tag{3}$$

which refers to both strict and defeasible, authorization and obligation rules from $\mathcal{P}$. The square brackets "[ ]" indicate an optional component of a rule (in this case the keyword "**normally**" denoting a defeasible rule). We use the term *head* of policy rule r to refer to the hd part in (3), where $hd \in HD$,

$$HD = \bigcup_{e \in E} \{permitted(e), \neg permitted(e), obl(e), obl(\neg e), \neg obl(e), \neg obl(\neg e)\}$$

and E is the set of all elementary actions in $\mathcal{T}$. We refer to the *cond* part of a policy rule r as its *body*. The signature $rei_lp(\mathcal{P})$ contains functions *permitted* and *obl*, as well as a function *neg* that encodes the negation "¬" when it is present in a policy rule. We introduce the following transformation **lp** that replaces the negation "¬" by the function *neg*:

- If x is a static, fluent, or elementary action, then $\mathbf{lp}(x) = x$ and $\mathbf{lp}(\neg x) = neg(x)$
- If e is an elementary action then:

$$\begin{aligned}
&\mathbf{lp}(permitted(e)) = permitted(e)\\
&\mathbf{lp}(\neg permitted(e)) = neg(permitted(e))\\
&\mathbf{lp}(obl(e)) = obl(e)\\
&\mathbf{lp}(\neg obl(e)) = neg(obl(e))\\
&\mathbf{lp}(obl(\neg e)) = obl(neg(e))\\
&\mathbf{lp}(\neg obl(\neg e)) = neg(obl(neg(e)))
\end{aligned}$$

The signature of $rei_lp(\mathcal{P})$ also includes the following functions:

- $b(r)$ for every rule r to denote the condition *cond* of r (i.e., its body);
- $ab(r)$ for each defeasible rule r, representing an exception to its application (i.e., the rule being overridden by another defeasible rule, as specified by a *prefer* relation).

Additionally, the signature of $rei_lp(\mathcal{P})$ contains the predicates:

- $rule(r)$ – where r is a rule label (referred shortly as "rule" below)
- $type(r, ty)$ – where $ty \in \{strict, defeasible, prefer\}$ is the type of rule r
- $head(r, \mathbf{lp}(hd))$ – to denote the head hd of rule r
- $body(r, b(r))$ – to associate a rule r with its body denoted by function $b(r)$
- $mbr(b(r), \mathbf{lp}(l))$ – for every l in the condition *cond* of rule r, where the condition is represented by $b(r)$ (*mbr* stands for "member")
- $prefer(d_1, d_2)$ – where d_1 and d_2 are defeasible rule labels

To reason about which policies are applicable (i.e., active) at each time step, the signature of $rei_lp(\mathcal{P})$ also includes the following predicates:

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

- $holds(x, i)$ – where i is a time step and x may be a rule r; $\mathbf{lp}(\text{hd})$ for the head hd of a rule; the function $b(r)$ representing the body of a rule; or the function $ab(r)$ for every defeasible rule r
- $opp(r, \mathbf{lp}(\overline{hd}))$ – where r is a defeasible rule and $\overline{hd} \in HD$ (opp stands for "opposite")

The predicate $holds$ determines which policy rules are applicable, based on the truth values of statics and fluents in a state and the interactions among policy rules. Note that $holds$ with arity two is an overloaded predicate, also applying to pairs of fluents and time steps when it is used to describe states of a trajectory in $\mathcal{T}$. The predicate $opp(r, \mathbf{lp}(\overline{hd}))$ indicates that $\overline{hd}$ is the logical complement of r's head hd.

The ASP **translation** of a policy $\mathcal{P}$ denoted by the ASP program $rei_lp(\mathcal{P})$ consists of:

1. A collection $\mathcal{E}(\mathcal{P})$ of facts (or rules) representing the encoding of the policy rules in $\mathcal{P}$ using the predicates $rule$, $type$, $head$, mbr, and $prefer$
2. The set of policy-independent ASP rules $\mathcal{R}$ shown below in (4), which define predicates $holds(x, i)$ and $opp(r, \mathbf{lp}(\overline{hd}))$. In these ASP rules, variable F represents a fluent, S a static, E an elementary action, H a happening (i.e., an elementary action or its negation), and I a time step.

$$
\begin{array}{ll}
body(R, b(R)) & \leftarrow rule(R) \\
holds(R, I) & \leftarrow type(R, strict), holds(b(R), I) \\
holds(R, I) & \leftarrow type(R, defeasible), holds(b(R), I), \\
 & \quad opp(R, O), \text{not } holds(O, I), \text{not } holds(ab(R), I) \\
\neg holds(B, I) & \leftarrow body(R, B), mbr(B, F), fluent(F), \neg holds(F, I). \\
\neg holds(B, I) & \leftarrow body(R, B), mbr(B, neg(F)), fluent(F), holds(F, I). \\
\neg holds(B, I) & \leftarrow body(R, B), mbr(B, S), static(S), \neg holds(S), step(I). \\
\neg holds(B, I) & \leftarrow body(R, B), mbr(B, neg(S)), static(S), holds(S), step(I). \\
holds(B, I) & \leftarrow body(R, B), not \neg holds(B, I), step(I). \\
holds(ab(R_2), I) & \leftarrow prefer(R_1, R_2), holds(b(R_1), I) \\
holds(Hd, I) & \leftarrow holds(R, I), head(R, Hd) \\
opp(R, permitted(E)) & \leftarrow head(R, neg(permitted(E))) \\
opp(R, neg(permitted(E))) & \leftarrow head(R, permitted(E)) \\
opp(R, obl(H)) & \leftarrow head(R, neg(obl(H))) \\
opp(R, neg(obl(H))) & \leftarrow head(R, obl(H))
\end{array}
\tag{4}
$$

Thus, for a policy $\mathcal{P}$, $rei_lp(\mathcal{P}) = \mathcal{E}(\mathcal{P}) \cup \mathcal{R}$.

Example 1

(ASP Encoding $\mathcal{E}(\mathcal{P})$ of a Policy $\mathcal{P}$). Let's give an example of the encoding $\mathcal{E}(\mathcal{P})$ of a policy $\mathcal{P}$ consisting of a unique policy rule:

$$r6(1) : obl(stop(1)) \textbf{ if } pedestrians_are_crossing(1)$$

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

where 1 refers to location 1 in Figure 1, $stop(1)$ is an action, and $pedestrians_are_crossing(1)$ is a fluent. $\mathcal{E}(\mathcal{P})$ for this policy will consist of the ASP facts:

$rule(r6(1)).$
$type(r6(1),\ strict).$
$head(r6(1),\ obl(stop(1))).$
$mbr(b(r6(1)),\ pedestrians_are_crossing(1)).$

We discuss next the role of the set of rules $\mathcal{R}$ given in (4). For a given trajectory in the system description to which policy $\mathcal{P}$ belongs, if $\mathcal{P}$ is categorical, atoms of the form $holds(\mathbf{lp}(hd), i)$ indicate which literals hd from HD (i.e., literals formed by predicates *permitted* and *obl*) hold at time step i. Atoms of the form $holds(r, i)$, where r is a rule and i is a time step, indicate which rules are applicable at each step. If a rule is applicable, the agent's actions must be checked for compliance. Strict rules apply in all states where their condition (i.e., body) is satisfied, but not in states that do not satisfy it – see the second rule in (4). For instance, in the sample policy from Figure 2, rule 6 applies only if pedestrians are crossing at the location where the agent is currently situated; crossings at other locations or the absence of pedestrians are irrelevant, and the rule would not apply in those situations. Strict rules with satisfied bodies cannot be rendered inapplicable by rules with the opposite (complement) head. However, a defeasible rule may be overridden by another rule, rendering it inapplicable in a state that would otherwise satisfy its condition – see the third rule in (4). The part "$opp(R, O),\ \text{not}\ holds(O, I)$" in this ASP rule specifies that an applicable (strict) rule with the opposite head makes the defeasible rule R inapplicable. The part "not $holds(ab(R), I)$" indicates that a preference relationship can render a defeasible rule R inapplicable if a more preferred defeasible rule applies in the state. This is encoded as an exception to the default case through the use of $ab(R)$, following the standard treatment of defaults (see the ninth rule in (4)). Ultimately, only the policy rules applicable at a given time step i need to be checked for the agent's compliance at that step.

The reified translation $rei_lp(\mathcal{P})$ of $\mathcal{AOPL}'$ into ASP preserves the intended semantics of $\mathcal{AOPL}$. Proposition 1 discusses the correspondence between the original ASP translation of $\mathcal{AOPL}$ and the reified translation $rei_lp(\mathcal{P})$ introduced here for $\mathcal{AOPL}'$. To establish this correspondence, we first present the ASP translation of a trajectory, which will be referenced in the proposition. If $t = \langle \sigma_0, ca_0, \sigma_1, ca_1, \ldots, ca_n, \sigma_{n+1} \rangle$ is a trajectory in transition diagram $\mathcal{T}$, where $\sigma_0, \ldots, \sigma_{n+1}$ are states and $ca_0, \ldots, ca_n$ are actions, then the ASP encoding of t denoted by $\mathcal{E}(t)$ is defined as follows:

$$\begin{aligned}\mathcal{E}(t) =_{def}\ & \{holds(f,i) : f \text{ is a fluent},\ f \in \sigma_i,\ 0 \le i \le n+1\} \cup \\ & \{\neg holds(f,i) : f \text{ is a fluent},\ \neg f \in \sigma_i\ \ 0 \le i \le n+1\} \cup \\ & \{holds(s) : s \text{ is a static},\ s \in \sigma_i,\ 0 \le i \le n+1\} \cup \\ & \{\neg holds(s) : s \text{ is a static},\ \neg s \in \sigma_i,\ 0 \le i \le n+1\} \cup \\ & \{occurs(e,i) : e \text{ is an elementary action},\ e \in ca_i,\ 0 \le i \le n\}\end{aligned}$$

Proposition 1.
Let $\langle \sigma_i, ca_i \rangle$ be an event in a trajectory $t = \langle \sigma_0, ca_0, \sigma_1, ca_1, \ldots, ca_n, \sigma_{n+1} \rangle$ in the transition diagram $\mathcal{T}$. For every $i \in \{0, \ldots, n\}$, let $\mathcal{A}_i$ be the collection of answer sets of $lp(\mathcal{P}, \sigma_i)$ and let $\mathcal{B}_i$ be the collection of answer sets of $rei_lp(\mathcal{P}) \cup \mathcal{E}(\langle \sigma_i, ca_i \rangle)$.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

There is a one-to-one correspondence $map : \mathcal{A}_i \rightarrow \mathcal{B}_i$ *such that if* $map(A) = B$ *then for every* $hd \in HD$ *where* $\mathbf{lp}(hd) \in A$, *we have that* $\exists holds(\mathbf{lp}(hd), i) \in B$.

Challenges: The $rei_lp(\mathcal{P})$ translation described above presents specific challenges when it comes to automating the translation process. The description of $\mathcal{E}(\mathcal{P})$ above assumes that rule labels are ground terms, as in Example 1. In practical applications, it is more common for rule labels to contain variables, thus representing a schema for a collection of ground rule labels. If that is the case, then $\mathcal{E}(\mathcal{P})$ would not contain facts, but rather a collection of rules qualifying the variables.

Let's consider rule 6 from Figure 2 stating that there is a strict obligation to stop when pedestrians are crossing. Assuming that the dynamic domain includes action $stop(l)$ and fluent $pedestrians_are_crossing(l)$ where l is a location, the corresponding $\mathcal{AOPL}'$ rule would look as follows:

$$r6(L) : obl(stop(L)) \quad \textbf{if} \quad pedestrians_are_crossing(L) \tag{5}$$

Note that the rule label, $r6(L)$ is not ground. Hence the representation of this rule in $\mathcal{E}(\mathcal{P})$ would be as follows:

$$\begin{aligned}
&rule(r6(L)) \leftarrow action(stop(L))\\
&type(r6(L), strict) \leftarrow rule(r6(L))\\
&head(r6(L), obl(stop(L))) \leftarrow rule(r6(L))\\
&mbr(b(r6(L)), pedestrians_are_crossing(L)) \leftarrow rule(r6(L))
\end{aligned}$$

where the body "$action(stop(L))$" of the first rule above is derived from the name of the action referenced in policy rule 6.

Another challenge posed by the reified translation of $\mathcal{AOPL}'$ into ASP is dealing with arithmetic comparisons. To illustrate this, let's consider rule 1 from Figure 2. The label of the $\mathcal{AOPL}'$ statement for this policy rule, $r1(L_1, L_2, S, S_1)$, needs to keep track of four variables: L_1, L_2, and S associated with the *drive* action from location L_1 to location L_2 at speed S; and the speed limit S_1 for the road section from L_1 to L_2:

$$\begin{aligned}
r1(L_1, L_2, S, S_1) : \quad &\textbf{normally} \quad \neg permitted(drive(L_1, L_2, S))\\
&\textbf{if} \quad speed_limit(L_1, L_2, S_1),\ S > S_1 + 5,\ S_1 < 55
\end{aligned} \tag{6}$$

The ASP encoding of rule 1 from Figure 2 would contain a statement:

$$mbr(b(r1(L_1, L_2, S, S_1)),\ S > S_1 + 5) \leftarrow rule(r1(L_1, L_2, S, S_1))$$

Since this syntax is not allowed by ASP solvers, we replace arithmetic comparisons with our own, for example gt for ">", as in:

$$mbr(b(r1(L_1, L_2, S, S_1)), gt(S, S_1 + 5)) \leftarrow rule(r1(L_1, L_2, S, S_1))$$

and define these new symbols (e.g., gt, gte) via ASP rules added to the $\mathcal{R}$ part of $rei_lp(\mathcal{P})$, which corresponds to policy-independent ASP rules (see (4)).

The process of obtaining these translations is described further in Section 4.3. But first, let's extend $\mathcal{AOPL}'$ with means for representing penalties.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

4.2 Extending $\mathcal{AOPL}'$ with penalties: $\mathcal{AOPL}$-$\mathcal{P}$

In our framework we extend the $\mathcal{AOPL}'$ syntax (i.e., statements of type (2a)-(2i)) by a new type of statement for penalties:

$$penalty(r, p) \textbf{ if } cond_p \tag{7}$$

where r is the label of the prohibition or obligation rule for which the penalty is specified, p stands for the number of penalty points imposed if the policy rule r applies and the agent is noncompliant with it, and $cond_p$ is a collection of static literals. The "**if** $cond_p$" part is omitted if $cond_p$ is empty. We denote this extension of $\mathcal{AOPL}'$ as $\mathcal{AOPL}$-$\mathcal{P}$.

For instance, the penalty associated with rule 6 from Figure 2 is encoded in $\mathcal{AOPL}$-$\mathcal{P}$ as:

$$penalty(r6(L), 3) \tag{8}$$

This says that the agent will incur a 3-point penalty at each time step in which this rule applies and the agent's action is noncompliant with it.

Multiple penalty values may be associated with the same rule, reflecting different gravity levels, as in rule 1 of Figure 2. If the rule applies at a given time step and the agent's action is noncompliant, the agent incurs one of the specified penalties according to the gravity level. The various levels of penalties assigned to rule 1 are stated in $\mathcal{AOPL}$-$\mathcal{P}$ as:

$$\begin{array}{lll} penalty(r1(L_1, L_2, S, S_1), 1) & \textbf{if} & S - S_1 < 10 \\ penalty(r1(L_1, L_2, S, S_1), 2) & \textbf{if} & S - S_1 >= 10,\ S - S1 < 20 \\ penalty(r1(L_1, L_2, S, S_1), 3) & \textbf{if} & S - S_1 >= 20 \end{array} \tag{9}$$

Recall from the previous section that the semantics of the policy language are given in terms of a translation into ASP. We need to expand this translation to cover the new type of statement in (7). Given a policy $\mathcal{P}$, recall that $rei_lp(\mathcal{P}) = \mathcal{E}(\mathcal{P}) \cup \mathcal{R}$, where $\mathcal{E}(\mathcal{P})$ is the encoding of the statements in $\mathcal{P}$ and $\mathcal{R}$ is the set of policy-independent rules described in (4). We expand the $\mathcal{E}(\mathcal{P})$ part of $rei_lp(\mathcal{P})$ with the ASP translation of the new type of $\mathcal{AOPL}$-$\mathcal{P}$ statement (7) for penalties:

$$penalty(r, p) \leftarrow rule(r),\ cond_p$$

The rules in $\mathcal{R}$ remain as they are. As an example, statement (8) is translated into ASP as:

$$penalty(r6(L), 3) \leftarrow rule(r6(L))$$

Penalty statements (9) are translated as:

$$\begin{array}{lll} penalty(r1(L_1, L_2, S, S_1), 1) & \leftarrow & rule(r1(L_1, L_2, S, S_1)),\ S - S_1 < 10 \\ penalty(r1(L_1, L_2, S, S_1), 2) & \leftarrow & rule(r1(L_1, L_2, S, S_1)),\ S - S_1 \geq 10,\ S - S_1 < 20 \\ penalty(r1(L_1, L_2, S, S_1), 3) & \leftarrow & rule(r1(L_1, L_2, S, S_1)),\ S - S_1 \geq 20 \end{array}$$

We developed a Python-based translator from the $\mathcal{AOPL}$-$\mathcal{P}$ into ASP that creates the ASP policy encoding $\mathcal{E}(\mathcal{P})$ for a policy $\mathcal{P}$. We describe this translator next.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

```
1  rule(r1(L1, L2, S, S1)) :-
2      action(drive(L1, L2, S)),
3      holds(speed_limit(L1, L2, S1)).
4  type(r1(L1, L2, S, S1), defeasible) :- rule(r1(L1, L2, S, S1)).
5  head(r1(L1, L2, S, S1), neg(permitted(drive(L1, L2, S)))) :-
6      rule(r1(L1, L2, S, S1)).
7  mbr(b(r1(L1, L2, S, S1)), speed_limit(L1, L2, S1)) :-
8      rule(r1(L1, L2, S, S1)).
9  mbr(b(r1(L1, L2, S, S1)), gt(S, S1+5)) :- rule(r1(L1, L2, S, S1)).
10 mbr(b(r1(L1, L2, S, S1)), lt(S1, 55)) :- rule(r1(L1, L2, S, S1)).
11 prefer(r1(L1, L2, S, S1), r7(L1, L2, S)) :-
12     rule(r1(L1, L2, S, S1)), rule(r7(L1, L2, S)).
13 penalty(r1(L1, L2, S, S1),1) :- rule(r1(L1, L2, S, S1)), S - S1 < 10.
14 penalty(r1(L1, L2, S, S1),2) :- rule(r1(L1, L2, S, S1)), S - S1 >= 10,
15     S - S1 < 20.
16 penalty(r1(L1, L2, S, S1),3) :- rule(r1(L1, L2, S, S1)), S - S1 >= 20.
```

Fig 3. ASP translation for policy rule 1 from Figure 2.

4.3 Translator from $\mathcal{AOPL}$-$\mathcal{P}$ to ASP

To automate the translation of $\mathcal{AOPL}$-$\mathcal{P}$ policies and their penalties and streamline the use of our ASP-based framework, we developed the first Python-based translator for $\mathcal{AOPL}$-$\mathcal{P}$. This translator takes as input a text file containing all relevant $\mathcal{AOPL}$-$\mathcal{P}$ policy rules for a given domain, including associated penalties, and generates the corresponding ASP encodings for all policy rules and penalties as described in Sections 4.1 and 4.2. This automated translation produces ASP rules defining predicates *rule*, *type*, *head*, *mbr*, *prefer*, and *penalty* and corresponds to the policy-specific component $\mathcal{E}(\mathcal{P})$ of the program $rei_lp(\mathcal{P})$ for a policy $\mathcal{P}$.

For example, consider the $\mathcal{AOPL}$-$\mathcal{P}$ representation of policy rule 1 from Figure 2 shown in (6) and its associated penalties seen in (9), copied below for clarity:

$$r1(L_1, L_2, S, S_1): \quad \textbf{normally} \quad \neg permitted(drive(L_1, L_2, S)) \\ \textbf{if} \quad speed_limit(L_1, L_2, S_1),\ S > S_1 + 5,\ S_1 < 55$$

$$penalty(r1(L_1, L_2, S, S_1), 1) \quad \textbf{if} \quad S - S_1 < 10 \\ penalty(r1(L_1, L_2, S, S_1), 2) \quad \textbf{if} \quad S - S_1 >= 10,\ S - S1 < 20 \\ penalty(r1(L_1, L_2, S, S_1), 3) \quad \textbf{if} \quad S - S_1 >= 20$$

According to policy rule 10 in Figure 2, $r1$ overrides $r7$, which corresponds to the $\mathcal{AOPL}$-$\mathcal{P}$ statement:

$$prefer(r1(L_1, L_2, S, S_1), r7(L_1, L_2, S))$$

The expected and generated output of the translator for rule $r1$, as well as its associated penalties and *prefer* statement, is shown in Figure 3.

The rule $r1$ highlights several challenges for the translator in generating safe rules for a solver like Clingo. First, to ensure the safety of the policy rule definition (see lines 1–3 in the listing in Figure 3), the translator must not only identify the action being permitted or prohibited – in this case, $drive(L_1, L_2, S)$ – but also account for any other variables in the rule label not directly associated with the action. These variables must be linked to an appropriate fluent or static predicate and added to the body of the ASP rule. In this case, S_1 is unrelated to the $drive(L_1, L_2, S)$ action mentioned in the head of the

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

$\mathcal{AOPL}$-$\mathcal{P}$ rule (i.e., before the keyword **if**), and instead represents the speed limit on the road segment from L_1 to L_2. The translator identifies $speed_limit(L_1, L_2, S_1)$ in the body of the $\mathcal{AOPL}$-$\mathcal{P}$ rule as the relevant static and includes $holds(speed_limit(L_1, L_2, S_1))$ in the body of the corresponding ASP rule (see line 3 in the listing). A second issue is that the translator must determine whether a policy rule is defeasible or strict based on the presence or absence of the keyword **normally**, adding a corresponding rule to the translation that specifies its type as either $type(r, defeasible)$ or $type(r, strict)$ (see line 4 in the listing). Finally, the body of the policy rule contains comparison operators, which must be translated into predicates such as *gt* or *lt* in the reified logic programming encoding. These predicates ensure that comparisons can be properly represented as terms within the *mbr* predicate (see lines 9–10 in the listing).

Our Python translator relies on the following functions to achieve this:

- $parse_aopl_rule()$: decomposes an $\mathcal{AOPL}$-$\mathcal{P}$ policy into its fundamental components; extracts rule labels; separates the rule into head (authorization or obligation for an action) and body (conditions); and analyzes variables for classification and further processing.
- $classify_literals()$: processes the body of the rule to format it for ASP; filters out numerical conditions (e.g., inequalities); wraps state-related conditions in *holds* predicates; and classifies literals based on whether they pertain to actions or states.
- $translate_aopl_rule_to_asp()$: converts parsed $\mathcal{AOPL}$-$\mathcal{P}$ rules into ASP syntax; constructs the rule *head* using *permitted* and *obl* constructs and assembles the body *mbr* by incorporating conditions and constraints.
- $aopl_to_asp()$: processes different types of $\mathcal{AOPL}$-$\mathcal{P}$ rules, determining whether they are strict or defeasible; translates penalty clauses and preference statements into the corresponding ASP representation.
- $process_aopl_file()$: manages file input and output for the translator; reads $\mathcal{AOPL}$-$\mathcal{P}$ policies from a text input file, processes them using $aopl_to_asp()$, and writes the translated ASP rules to an output text file.

The code for the translator is available at https://github.com/vineelsai313/Penalties.

4.4 Reasoning about penalties in planning

In this section we present an ASP reasoning mechanism that considers penalties in planning. We have shown in Section 4.1 how the program $rei_lp(\mathcal{P})$ can be used to determine which policies are applicable (i.e., are active and hence should be checked for compliance with) at each step in a trajectory. The policy rules applicable at each step may be strict or defeasible. An agent can choose to be noncompliant with either type; the distinction simply indicates whether a rule allows exceptions to its applicability.

We start by devising ASP rules that can flag what penalty is incurred by an agent at a time step, and for being noncompliant with which policy rule. We do so by introducing a predicate $add_penalty(r, p, i)$, which says that a penalty of p points should be incurred for noncompliance with an applicable policy rule r at time step i. The definition of this predicate covers three cases for noncompliance. The first one is when an action that is

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

not permitted to be executed is included in the plan:

$$add_penalty(R,P,I) \leftarrow rule(R),\ holds(R,I),\ head(R,neg(permitted(E))),\\ occurs(E,I),\ penalty(R,P)$$

where R is a rule, I is a time step, E is an elementary agent action, and P is a penalty.

The second case is when an agent is obligated to execute an action but the action is not included in the plan:

$$add_penalty(R,P,I) \leftarrow rule(R),\ holds(R,I),\ head(R,obl(E)),\\ action(E), \text{not } occurs(E,I),\ penalty(R,P)$$

The final case is when an agent is obligated to not execute an action, but the action is included in the plan:

$$add_penalty(R,P,I) \leftarrow rule(R),\ holds(R,I),\ head(R,obl(neg(E))),\\ occurs(E,I),\ penalty(R,P)$$

The overall penalty for a plan can be captured by the predicate *cumulative_penalty* defined using the $\#sum$ aggregate:

$$cumulative_penalty(N) \leftarrow \#sum\{P,R,I : add_penalty(R,P,I)\} = N$$

Introducing this predicate is optional as this information can be retrieved from the solver's output, but we include it here to make the overall penalty of plans clearer in later examples.

Example 2.
Consider the Traffic Norms domain from Section 3 and an agent that starts its journey at location 6 with the objective of reaching destination location 10. A critical point of this route is that there is a "Do not enter" sign situated at location 8, and a school bus is stopped between 14 and 13 at time step 3. Let's analyze a possible plan and the associated penalties:

occurs(drive(6,8,45),0)	add_penalty(r3(6,8,45),3,0)
occurs(drive(8,7,85),1)	add_penalty(r1(6,8,45,15),3,0)
occurs(drive(7,9,45),2)	add_penalty(r1(8,7,85,45),3,1)
occurs(drive(9,10,85),3)	add_penalty(r1(7,9,45,15),3,2)
cumulative_penalty(15)	add_penalty(r1(9,10,85,25),3,3)

Recall that the third parameter of a drive action is the speed in mph. Looking at the incurred penalties, note that there are two penalties of 3 points added at time step 0, one for breaking rule 3 by disobeying the "Do not enter" sign at location 8, and another one for rule 1 for speeding between locations 6 and 8 with a speed of 45 mph while the speed limit on that section is set to 15 mph.

4.5 Adding other metrics: Time

One of our goals is to introduce a distinction between different Risky behavior plans (those that allow noncompliant actions). So far we considered cumulative penalty as one of the metrics. Another metric, which corresponds to reasoning about emergency

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

situations, is time (i.e., duration). This is in fact a more accurate metric than the one used in previous work for emergency situations – plan length – and more generalizable to other scenarios, as compared to other metrics such as distance. A new predicate is added to our ASP framework, $add_time(t, i)$, which means add t time units to the overall plan execution time, to account for the duration of the action executed at time step i. Rules that define this predicate look like:

$$add_time(t, I) \leftarrow occurs(e, I)$$

where t is a number representing the time units and e is an elementary action.

To test our framework, we assigned a certain number of time units for each action, as follows: 5-time units if driving between two connected locations at a speed > 55 mph; 10-time units if driving at a speed between 36 mph and 55 mph; 15-time units if driving at a speed that is ≤ 35 mph; and 2-time units if stopped. Here is an example in ASP:

$$add_time(5, I) \leftarrow occurs(drive(L_1, L_2, S), I),\ \ S > 55$$

Overall time to execute a plan can be calculated similarly to cumulative penalty, by introducing a *cumulative_time* predicate:

$$cumulative_time(N) \leftarrow \#sum\{T, I : add_time(T, I)\} = N$$

Example 3.
Consider the scenario in Example 2. The following add_time and cumulative_penalty atoms would characterize the plan in that example:

add_time(10,0)	add_time(10,2)	cumulative_time(30)
add_time(5,1)	add_time(5,3)	

4.6 Behavior modes revisited

Different behavior modes can be expressed as priorities between the metrics discussed above, cumulative penalty and cumulative time, as well as other potential ones. As a first distinction, we introduce the **emergency** and **nonemergency** behavior modes. The former prioritizes time over penalties, while the latter does the opposite. The behavior mode is set by the agent's controller by adding one of the two facts below:

$$emergency \quad \textbf{or} \quad non_emergency$$

Priorities for time versus penalty for each of these modes are set via the rules:

$$\begin{aligned}
&time_priority(2) \leftarrow emergency\\
&penalty_priority(1) \leftarrow emergency\\
&penalty_priority(2) \leftarrow non_emergency\\
&time_priority(1) \leftarrow non_emergency\\
&:\sim add_penalty(R, P, I), penalty_priority(Y).\ [P@Y, R, I]\\
&:\sim add_time(T, I), time_priority(Y).\ [T@Y, I]
\end{aligned}$$

This encoding relies on the fact that the larger the number following the "@" symbol in a soft constraint, the higher the priority of complying with that constraint.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Example 4.
For the scenario in Example 2, an agent in the ***emergency*** *mode would choose the plan:*

$$occurs(drive(6,8,45),0) \quad occurs(drive(8,7,85),1) \quad occurs(drive(7,9,45),2)$$
$$occurs(drive(9,10,85),3)$$

with cumulative penalty 15 and cumulative time 30. An agent in the ***nonemergency*** *mode would choose the plan:*

$$occurs(drive(6,11,15),0) \quad occurs(drive(11,12,65),1) \quad occurs(drive(12,14,15),2)$$
$$occurs(stop(14),3) \quad occurs(drive(14,13,25),4) \quad occurs(drive(13,10,15),5)$$

with cumulative penalty 0 and cumulative time 67. This agent would stop at location 14 at time step 3 as there is a stopped school bus between 13 and 14.

If the $\mathcal{AOPL}$-$\mathcal{P}$ policy is both consistent and unambiguous, then the ASP program for each of the behavior modes **emergency** vs **nonemergency** is guaranteed to compute the optimal plan with respect to that behavior mode.

4.7 Discussion

As mentioned in Section 3, penalties are set initially on a scale from 1 to 3 depending on the severity of the infraction. However, not harming humans should be of utmost importance. Therefore, in the Traffic Norms domain, an agent should not violate policy rules that involve pedestrians and school buses. One way to achieve this is to assign a high penalty for noncomplying with those policy rules (e.g., a penalty of 50 points, but this number can be changed according to the domain), and flag this value via a predicate *high_penalty*. To prevent the harm of human life, we can then add a constraint:

$$\leftarrow \ add_penalty(R,H,I),\ high_penalty(H)$$

This ensures that the agent always complies with applicable policy rules whenever noncompliance could result in human harm.

To account for situations like the ones in the thought experiment called "The Trolley Problem" developed by philosopher Philippa Foot and adapted by Judith Jarvis Thomson, such a constraint may be excluded by the controllers of the agent or policy makers, and replaced with a directive to minimize the execution of actions that result in human harm (i.e., minimize actions that incur a high penalty, represented as a soft constraint):

$:\sim add_penalty(R,H,I),\ high_penalty(H).\ [H@3,R,I]$

Variations of the rules above can be devised for different scenarios.

Furthermore, other behavior modes can be specified, for instance one in which the agent looks for plans with an upper bound for the overall time and (otherwise) minimum penalty (which corresponds to situations in real life where someone needs to arrive to the destination within a time limit, but tries to do so with minimum penalty):

$$\leftarrow \ culumative_time(N),\ max_time(M),\ N > M$$
$$:\sim add_penalty(R,P,I), penalty_priority(Y).\ [P@Y,R,I]$$

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

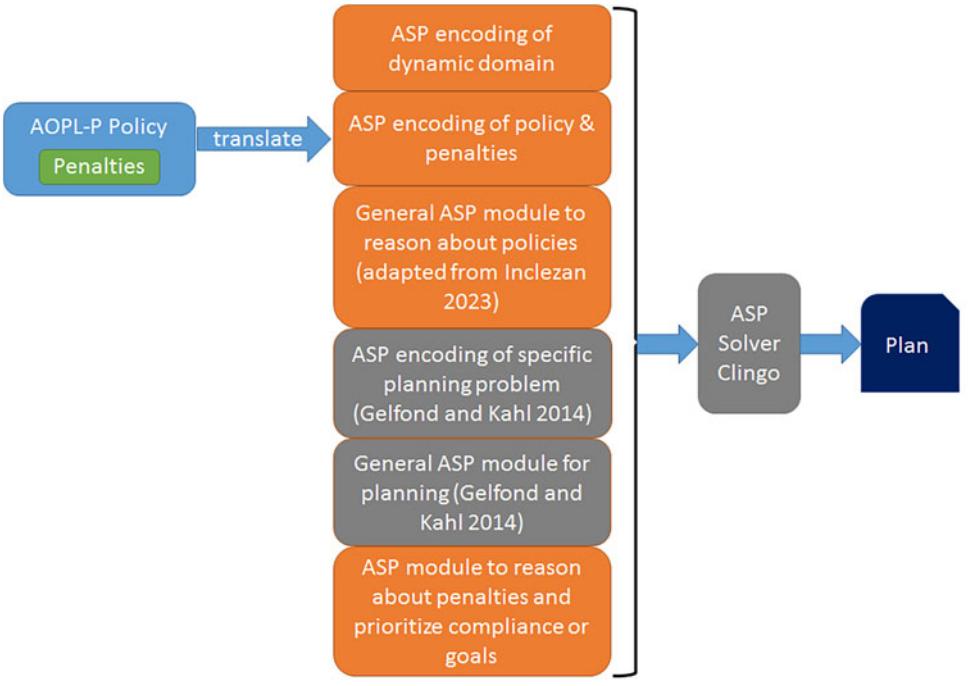


Fig 4. High-level framework view.

4.8 High-level view of the framework

Key components of our ASP framework are depicted in Figure 4. Grey elements indicate modules that are adopted from work by others, and non-gray components are either our own or adapted from others' work with substantial modifications. The main components of the framework are:

1. The ASP encoding of the **dynamic domain**, written according to established ASP methodologies (e.g., (Gelfond and Kahl 2014)).
2. The ASP encoding of the **policy and its associated penalties**. Policies for the dynamic domain are initially specified in $\mathcal{AOPL}$-$\mathcal{P}$, together with their penalties. Then $\mathcal{AOPL}$-$\mathcal{P}$ statements are automatically translated into ASP using the translator described in Section 4.3. This corresponds to the $\mathcal{E}(\mathcal{P})$ component of the program $rei_lp(\mathcal{P})$ for a policy $\mathcal{P}$.
3. A **general ASP module for reasoning about policies.** This module determines which policies are applicable at different time steps in the execution of a plan. It corresponds to the policy-independent component $\mathcal{R}$ of the program $rei_lp(\mathcal{P})$, substantially adapted from work by Inclezan (2023).
4. An ASP encoding of a **specific planning problem**, specifying the initial state, goal state, any observations along the way, and whether the situation is an emergency or nonemergency, following established methods for ASP planning (Gelfond and Kahl 2014).

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Table 1. *Performance results: room domain*

	Our Proposed Framework				The HI Framework			
	Non-Emergency		Emergency		Normal Mode		Risky Mode	
Scenario #	T (s)	L	T (s)	L	T (s)	L	T (s)	L
1	0.366	3	0.267	3	2.989	3	3.078	3
2	0.254	5	0.225	4	3.267	5	3.086	4
3	0.227	6	0.202	5	2.831	6	2.780	5
4	0.286	10	0.203	3	3.325	10	3.360	3
5	0.239	7	0.411	2	3.014	7	2.267	2
6	0.285	3	0.277	3	2.717	3	2.969	3
7	0.253	5	0.328	3	3.158	5	2.555	3
8	0.297	4	0.421	4	2.559	4	3.180	4
9	0.325	3	0.247	3	2.897	3	2.319	3
10	0.178	0	0.150	0	3.116	0	3.034	0
11	0.230	4	0.238	3	3.399	4	2.766	3
12	0.274	3	0.227	3	3.074	3	3.050	3
13	0.270	6	0.389	6	2.959	6	2.636	6
14	0.281	4	0.239	4	2.616	4	2.836	4

T (s) – average time in seconds over 10 runs; L – plan length.

5. A **general ASP module for planning** developed according to established ASP planning methodologies, to generate plans (Gelfond and Kahl 2014; Son *et al.* 2023).
6. An ASP module to **rank plans and select the optimal plan**. This module captures the decision making process of a policy-aware agent in emergency versus nonemergency scenarios, by prioritizing actions based on a combination of penalty severity and time-efficient goal achievement.

In our experimental setup, these ASP components are combined and provided to an ASP solver to compute the optimal plan.

5 Experimental Results

Our framework is meant to refine agent behavior specification and simulation – as compared to previous work by Harders and Inclezan (2023), which we call here the *HI framework* – for situations when noncompliant actions need to be included in a plan due to a high-stakes goal. By introducing penalties and additional metrics such as plan execution time, we improve on the quality of plans generated by the HI framework, since minimizing plan length does not necessarily result in minimal execution time. We also account for minimizing harm to humans, which the HI framework does not address. We report the relative time performance and plan quality of the two frameworks in Table 1 for the *Rooms Domain* (Harders and Inclezan 2023), and in Table 2 for the

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Table 2. *Performance results: traffic norms domain*

	Our Proposed Framework				The HI Framework			
	Non-Emergency		Emergency		Normal Mode		Risky Mode	
Scenario #	T (s)	L	T (s)	L	T (s)	L	T (s)	L
1	3.853	6	4.108	4	1.840	6	1.559	4
2	4.494	2	3.678	2	1.731	2	1.134	2
3	3.902	4	3.944	4	0.738	4	0.739	4
4	4.252	5	4.682	5	0.722	5	0.598	3
5	3.644	3	3.599	3	0.715	3	1.808	2
6	4.660	4	3.661	4	1.631	4	0.773	3
7	4.852	3	4.837	3	0.592	3	1.001	3
8	3.624	4	5.005	4	0.759	4	0.603	3
9	3.634	3	5.510	3	0.887	3	0.681	3
10	5.283	4	3.731	3	0.649	4	0.618	3
11	18.627	6	12.558	6	0.763	6	1.198	5
12	17.492	6	15.853	6	0.869	6	0.602	5
13	17.085	6	18.223	6	0.686	9	0.940	6

T (s) – average time in seconds over 10 runs; L – plan length.

newly introduced *Traffic Norms Domain* described in Section 3. All the tests were performed on a computer with a 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz CPU and RAM 16 GB. We used the Clingo solver, as in the HI Framework by Harders and Inclezan (to facilitate comparisons), and leveraged some of its specific constructs (e.g., *#minimize* in place of soft constraints). The code used in these experiments can be found at https://github.com/vineelsai313/Penalties.

The Rooms Domain consists of nine rooms labeled from $r0$ to $r8$, as shown in Figure 5 for a sample scenario. Rooms may be connected by doors, some of which may be one-way doors. Doors can be locked and unlocked by the agent using either a key specific to a door or a badge that can open any door. The agent starts in one of the rooms and aims to reach a specified destination. Additionally, the agent has knowledge of extreme conditions, such as active fires or contamination in certain rooms, and may possess special protective equipment.

This domain is governed by both authorization and obligation policies, which include both strict and defeasible rules. Some of these policies are:

1. The agent is obligated to use the key before using the badge if it possesses both.
2. The agent is not permitted to use its badge more than three times.
3. The agent is not permitted to open a one-way door from the wrong side.
4. Normally, the agent is obligated not to enter a room where there is an active fire.
5. However, if the agent has special protective equipment, it is allowed to enter a room with an active fire.
6. Normally, the agent is not permitted to enter a contaminated room.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

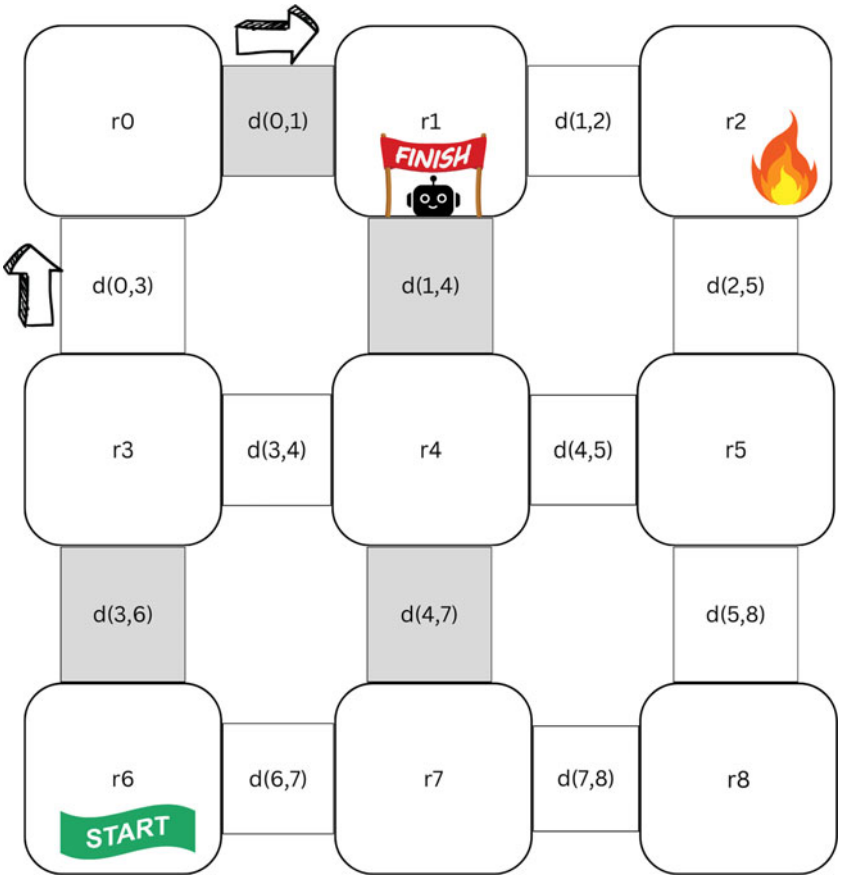


Fig 5. Rooms domain – scenario #3. The agent starts in room $r6$ and needs to get to room $r1$. Arrows indicate uni-directional doors. White doors are unlocked, while grey doors are locked. There is an active fire in room $r2$.

We further refined these policies by introducing penalty points ranging from 1 to 3, assigned based on the severity of the action.

We present the results for the Rooms Domain in Table 1. Our nonemergency mode corresponds to the Normal mode in the HI framework, which prioritizes plan length while disallowing noncompliant actions (see Section 2.2). Meanwhile, our emergency mode aligns with the Risky mode, which disregards policies entirely – note the similar plan lengths between these modes. As the table indicates, our penalization framework shows significant improvements in time efficiency, with the experiments for our framework completing in under 0.5 s, while the HI framework takes 2–4 s to complete each experiment on the same machine. We believe this is because the HI framework employs more metrics that are more complex (e.g., percentage of strongly-compliant elementary actions in a plan) than the ones in our framework.

The performance results for the Traffic Norms Domain in Table 2 show higher runtimes. This is because generating plans in this domain does not involve only selecting the path that the agent must follow, but also choosing a driving speed for each road segment,

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

out of a set of available speeds. We noticed increased runtimes for higher cardinalities of the set of possible speeds to select from (Scenarios #1–10 had six driving speed values to choose from; Scenarios #11–13 had eleven driving speed values). Other causes of an increased runtime are the additional obstacles that the agent must navigate in each scenario compared to earlier ones. The results in Table 2 indicate that the HI framework is more efficient than ours on the Traffic Norms Domain, but this is in the detriment of plan quality. Due to the nature of the Traffic Norms Domain in which selecting the appropriate speed is important, especially when penalties are considered, our framework goes through substantially more optimization cycles (3–16 vs 2–6 for the HI framework). This is a main cause for the performance differences for this domain. The HI framework simply finds a plan that it deems optimal sooner, because it does not work as hard on refining driving speeds.

In terms of plan quality, the plans produced by the HI Framework in the Risky mode are sometimes shorter than ours in the emergency mode (see Scenarios #4–6, 8, and 11–12). This is because our agent stops in situations when humans are involved (pedestrians or stopped school buses as described in Section 4.7) even in the emergency mode, while the HI Risky agent does not and may lead to harming humans. In most occasions, however, the plans produced by the two frameworks do not differ with respect to the path chosen by the agent but they may differ in the driving speed that is selected. For instance, a Normal agent's plan under the HI framework may include multiple actions of driving substantially under the speed limit, for example at 5 mph on road segments where the speed limit is 25 mph. In real life, such situations are undesirable as they may prompt other drivers to unsafe road behavior. In contrast, an agent in our nonemergency mode would choose driving speeds that follow closely the speed limit because total time to destination is considered in the selection of an optimal plan, though with less of a priority in the nonemergency mode, and different driving speeds imply different time needed to complete the action. In the Risky mode, the HI framework may select speeds that are either very low as before, or very high compared to the speed limit, because it deems all noncompliant actions equal. In our framework, the higher the driving speed is over the speed limit, the higher the penalty that is paid. Moreover, if an action breaks two policy rules, under the HI framework this would be counted as one noncompliant action, while in our framework, the agent pays the corresponding penalties for each of the two infractions.

Example 5

(Sample Plan Comparison: Our Framework vs. the HI Framework). To highlight the difference between the plans generated by our framework and those from the HI framework, consider Scenario #5 from the Traffic Norms Domain – specifically in emergency mode, which corresponds to the Risky behavior mode in the HI framework. In this scenario, the agent starts at location 6 at time step 0 and aims to reach location 4. The speed limit between locations 6 and 5 is 25 mph, and between 5 and 4 it is 15 mph, with similar speed limits on other segments. Additionally, pedestrians are crossing at location 5 at time step 1. As shown in Table 2*, our framework produces a plan of length 3, whereas the HI framework generates a shorter plan of length 2. The plan generated by our framework*

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Table 3. *Impact of the number of distinct speed values on performance in the traffic domain*

# of Speeds	6 Speed Values		7 Speed Values		8 Speed Values	
Scenario #	Non-Em. T (s)	Em. T (s)	Non-Em. T (s)	Em. T (s)	Non-Em. T (s)	Em. T (s)
1	3.853	4.108	5.626	5.907	9.986	10.024
2	4.494	3.678	3.653	3.989	8.936	8.731
3	3.902	3.944	4.457	6.853	7.839	8.867
4	4.252	4.682	6.282	8.656	8.264	12.325
5	3.644	3.599	4.845	5.436	3.911	4.903
6	4.660	3.661	5.901	3.723	11.372	4.813
7	4.852	4.837	4.392	5.426	7.463	10.180
8	3.624	5.005	5.459	4.817	9.948	12.963
9	3.634	5.510	4.882	4.798	11.160	13.605
10	5.283	3.731	5.142	5.450	10.753	11.884

T (s) – average time in seconds over 10 runs; Non-Em. – nonemergency scenario; Em. – emergency scenario.

is as follows:

$$occurs(drive(6,5,85),0) \quad occurs(stop(5),1) \quad occurs(drive(5,4,45),2)$$

The HI Framework generates the following plan:

$$occurs(drive(6,5,65),0) \quad occurs(drive(5,4,15),1)$$

Notably, even in emergency mode, our framework's plan includes stopping for crossing pedestrians at time step 1, reflecting its design to prevent harm to humans even under high-stakes conditions. In contrast, the agent guided by the HI framework proceeds without stopping at time step 1. Additionally, the HI framework does not optimize driving speeds to minimize total time, as it considers only plan length, regardless of the urgency or context of the situation.

In the Traffic Norms Domain, we examined how the number of distinct speed values affects performance within our framework. We ran scenarios 1–10 from Table 2 using the original six speed values, as well as configurations with seven and eight speed values. The results are presented in Table 3. As expected, the average computation time increases as the number of considered speed values grows. Some scenarios (e.g., scenario #5) are less affected, likely because the origin and destination are closer on the map, resulting in inherently shorter plans. The nonemergency mode is more sensitive to increased number of different speeds due to the stricter enforcement of policy rules in this mode, many of which directly involve speed constraints.

To enhance the efficiency of our framework and prepare it for real-world applications, we leveraged the description of dynamic domains, which specifies elementary actions that cannot physically be executed in certain states (e.g., the agent cannot stop at location l if it is not physically present there). For this, we extended the signature of $rei_lp(\mathcal{P})$ with two predicates: $action_in_rule(r,e)$, indicating that the elementary action referenced in

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Table 4. *Performance results: traffic norms domain – revisited*

	Our Improved Framework		Our Original Framework	
	Nonemergency	Emergency	Nonemergency	Emergency
Scenario #	T (s)	T (s)	T (s)	T (s)
1	2.842	2.930	3.853	4.108
2	2.655	2.777	4.494	3.678
3	3.447	2.695	3.902	3.944
4	2.777	2.760	4.252	4.682
5	2.917	2.897	3.644	3.599
6	2.725	3.077	4.660	3.661
7	2.728	2.989	4.852	4.837
8	2.711	3.172	3.624	5.005
9	2.649	3.042	3.634	5.510
10	2.786	3.106	5.283	3.731
11	5.494	5.955	18.627	12.558
12	4.839	4.733	17.492	15.853
13	4.622	4.528	17.085	18.223

T (s) – average time in seconds over 10 runs.

rule r is e; and $action_is_executable(r, i)$, indicating that the action in rule r is physically executable at time step i. These predicates were defined by the following new rules added to the component $\mathcal{R}$ of $rei_lp(\mathcal{P})$:

$$
\begin{aligned}
&action_in_rule(R, E) && \leftarrow head(R, permitted(E)).\\
&action_in_rule(R, E) && \leftarrow head(R, neg(permitted(E))).\\
&action_in_rule(R, E) && \leftarrow head(R, obl(E)), action(E).\\
&action_in_rule(R, E) && \leftarrow head(R, obl(neg(E))), action(E).\\
&action_in_rule(R, E) && \leftarrow head(R, neg(obl(E))), action(E).\\
&action_in_rule(R, E) && \leftarrow head(R, neg(obl(neg(E)))), action(E).\\
&action_is_executable(R, I) && \leftarrow action_in_rule(R, E), \text{not } \neg occurs(E, I), step(I).
\end{aligned}
$$

We also modified the second and third rules in $\mathcal{R}$, shown in (4), to require that an action be executable in order for a rule to be deemed applicable (i.e., to hold) at a given time step. The updated rules are encoded as follows:

$$
\begin{aligned}
&holds(R, I) \leftarrow type(R, strict), holds(b(R), I), action_is_executable(R, I),\\
&holds(R, I) \leftarrow type(R, defeasible), holds(b(R), I), action_is_executable(R, I),\\
&\qquad\qquad opp(R, O), \text{not } holds(O, I), \text{not } holds(ab(R), I)
\end{aligned}
$$

We report the performance results for this improved version of component $\mathcal{R}$ in Table 4, comparing them against our original framework shown in Table 2. As shown, performance improves, suggesting that with further enhancements the framework will be closer to practical real-world implementation.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

6 Related work

Penalties in the context of norm systems have been discussed by Balke *et al.* (2013a; 2013b). Shams *et al.* (2017) introduced an ASP framework for reasoning and planning with norms for autonomous agents. As in our work, their actions have an associated duration and can incur penalties. Moreover, their policies have an expiration deadline, which we do not consider in our work. In contrast, our framework leverages policy specification language $\mathcal{AOPL}$, which has built-in means for specifying situations when one policy overrides another one, expressed via defeasible policy rules and *prefer* predicates. Mechanisms for checking $\mathcal{AOPL}$ policies for unwanted features such as ambiguity have been developed and support the process of producing valid plans. Furthermore, our framework was tested on more complex scenarios than the ones presented by Shams et al., with policies consisting of up to 11 rules as seen in Figure 2.

Other existing work on norms, planning, and penalties tends to assume that pregenerated plans are available to the agent that then needs to choose an optimal plan based on a utility function (penalty or reward) (e.g., (Criado *et al*. 2010; Panagiotidi *et al.* 2012; Meneguzzi *et al.* 2015)). The BDI architecture (Rao and Georgeff 1991) is the underlying basis of a number of these frameworks (e.g., (Kollingbaum 2005)). Our framework stems from the AAA (Balduccini and Gelfond 2008) and AIA (Blount *et al.* 2014) architectures. It builds on initial explorations of policy-aware agents (Meyer and Inclezan 2021) and prior work on specifying agent behavior modes in relation to policies (Harders and Inclezan 2023; Inclezan *et al.* 2024).

While a majority of existing approaches focus on agents that are always compliant to norms (e.g., (Oren *et al.* 2011; Alechina *et al.* 2012)), our intention is to study agent behavior in situations when noncompliant actions must be executed to achieve an important goal (e.g., emergency situations). We believe this to be important to policy makers by simulating human behavior, as humans may act in ways that are noncompliant to cultural norms and even regulations.

7 Conclusions and future work

In this paper, we introduced a framework for modeling penalties for noncompliant behavior in autonomous agents operating under norm-governed environments. Building on the work of Harders and Inclezan (2023), we refined the distinction between plans containing noncompliant actions, enabling the selection of optimal plans in situations where such behavior is necessary. To this end, we extended $\mathcal{AOPL}$ with the ability to associate penalties with policies, capturing different levels of offense severity, and named the extended version $\mathcal{AOPL}$-$\mathcal{P}$. Additionally, we developed an automated translator from $\mathcal{AOPL}$-$\mathcal{P}$ into ASP to streamline the use of our framework. We evaluated our approach in two dynamic domains and found that it generates higher-quality plans while, most importantly, preventing harm to humans, an aspect not addressed in the previous framework.

We identify three key avenues for future work. First, we aim to collaborate with ethics experts to refine the penalty scheme, ensuring it better aligns with societal values and ethical considerations. While our current framework follows a utilitarian approach, alternative ethical perspectives, such as the ethics of care, deserve exploration. Second,

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

we plan to identify and define behavior modes most relevant to policymakers, enabling more realistic simulations of human attitudes toward policies and compliance for policy refinement. Finally, we seek to enhance the efficiency of our framework to improve scalability and performance.

References

Alechina, N., Dastani, M. and Logan, B. 2012. Programming norm-aware agents. In *Proceedings of AAMAS'12, Vol. 2*, Richland, SC, IFAAMAS, 1057–1064.

Alviano, M., Calimeri, F., Dodaro, C., Fuscá, D., Leone, N., Perri, S., Ricca, F., Veltri, P. and Zangari, J. 2017. The ASP system DLV2. In *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017*, F. Calimeri, G. Ianni and M. Truszczynski, Eds. Vol. 10377, Springer, Espoo, Finland, 215–221. July 3-6, 2017, Proceedings.

Balduccini, M. and Gelfond, M. 2008. The AAA architecture: An overview. In *Architectures for Intelligent Theory-Based Agents, Papers from the 2008 AAAI Spring Symposium, 2008*, AAAI Press, 1–6.

Balke, T., De Vos, M. and Padget, J. A. 2013a. Evaluating the cost of enforcement by agent-based simulation: A wireless mobile grid example. In *Proceedings of PRIMA'13*, Lecture Notes in Computer Science, Vol. 8291, Springer, 21–36.

Balke, T., De Vos, M. and Padget, J. A. 2013b. I-ABM: Combining institutional frameworks and agent-based modelling for the design of enforcement policies. *Artificial Intelligence and Law* 21, 4, 371–398.

Blount, J. L., Gelfond, M. and Balduccini, M. 2014. Towards a Theory of Intentional Agents. In *2014 AAAI Spring Symposium Series*, AAAI Press, 10–17.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2020. ASP-Core-2 input language format. *Theory and Practice of Logic Programming* 20, 2, 294–309.

Chintabathina, S., Gelfond, M. and Watson, R. 2005. Modeling hybrid domains using process description language. In *Proceedings of ASP '05: Answer Set Programming: Advances in Theory and Implementation*, CEUR-WS.org, 303–317.

Craven, R., Lobo, J., Ma, J., Russo, A., Lupu, E. and Bandara, A. 2009. Expressive policy analysis with enhanced system dynamicity. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, Association for Computing Machinery, New York, NY, USA, 239–250.

Criado, N., Argente, E. and Botti, V. 2010. A BDI architecture for normative decision making. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. AAMAS '10*, IFAAMAS, Richland, SC, 1383–1384.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19, 1, 27–82.

Gelfond, M. and Inclezan, D. 2013. Some properties of system descriptions of ALd. *Journal of Applied Non Classical Logics* 23, 1-2, 105–120.

Gelfond, M. and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3/4, 365–386.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Gelfond, M. and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17, 2-4, 301–321.

Gelfond, M. and Lifschitz, V. 1998. Action languages. *Electronic Transactions on Artificial Intelligence* 3, 16, 195–210.

Gelfond M. and Lobo J. 2008. Authorization and obligation policies in dynamic systems. In *Logic Programming*, M. Garcia de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 22–36.

Giunchiglia, E., Lee, J., Lifschitz, V., Mccain, N. and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153, 1-2, 105–140.

Harders, C. and Inclezan, D. 2023. Plan selection framework for policy-aware autonomous agents. In Proceedings of JELIA'23, S. A. Gaggl, M. V. Martinez and M. Ortiz, Eds. Lecture Notes in Computer Science, Vol. 14281, Springer, 638–646.

Inclezan, D. 2012. Modular action language ALM for dynamic domain representation. Ph.D. thesis, Texas Tech University, Lubbock, TX, USA.

Inclezan, D. 2023. An ASP framework for the refinement of authorization and obligation policies. *Theory and Practice of Logic Programming* 23, 4, 832–847.

Inclezan, D., Harders, C. and Tummala, V. S. K. 2024. A framework for defining behavior modes in policy-aware autonomous agents. In *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS 2024) co-located with the 40th International Conference on Logic Programming (ICLP 2024), Dallas, TX, USA, October 12th and 13th, 2024*, J. Arias, D. Azzolini, K. Basu, V. Dahl, M. Hecher, F. Pacenza, Z. G. Saribatur and S. C. Varanasi, Eds. CEUR Workshop Proceedings, Vol. 3799, CEUR-WS.org, Dallas, TX, USA.

Kollingbaum, M. 2005. Norm-governed practical reasonig agents. Ph.D. thesis, University of Aberdeen, UK.

Lifschitz, V. and Ren, W. 2006. A modular action description language. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, AAAI Press, 853–859.

Marek, V. W. and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczynski and D. S. Warren, Eds. Artificial Intelligence. Springer, 375–398.

Meneguzzi, F., Rodrigues, O., Oren, N., Vasconcelos, W. W. and Luck, M. 2015. BDI reasoning with normative considerations. *Engineering Applications of Artificial Intelligence* 43, 127–146.

Meyer, J. and Inclezan, D. 2021. APIA: An architecture for policy-aware intentional agents. In *Proceedings of ICLP'21 (Tech. Comm.). EPTCS*, Vol. 345, Open Publishing Association, 84–98.

Oren, N., Vasconcelos, W., Meneguzzi, F. and Luck, M. 2011. Acting on norm constrained plans. In *Computational Logic in Multi-Agent Systems*, J. Leite, P. Torroni, T. $\mathcal{R}$ Agotnes, G. Boella and L. van der Torre, Eds. Springer, 347–363.

Panagiotidi, S., Vázquez-Salceda, J. and Dignum, F. 2012. Reasoning over norm compliance via planning. In *Coordination, Organizations, Institutions, and Norms in Agent Systems (VIII) - 14th International Workshop, (COIN) 2012, Held Co-located with (AAMAS) 2012, Valencia, Spain*, H. Aldewereld, J. Simão Sichman, Eds. Lecture Notes in Computer Science, Springer, Vol. 7756, 35–52.

Rao, A. S. and Georgeff, M. P. 1991. Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Morgan Kaufmann, 473–484.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press

Shams, Z., De Vos, M., Padget, J. A. and Vasconcelos, W. W. 2017. Practical reasoning with norms for autonomous software agents. *Engineering Applications of Artificial Intelligence* 65, 388–399.

Son, T. C., Pontelli, E., Balduccini, M. and Schaub, T. 2023. Answer set planning: A survey. *Theory and Practice of Logic Programming* 23, 1, 226–298.

Tummala, V. and Inclezan, D. 2024. Policies, penalties, and autonomous agents. In *Logic Programming and Nonmonotonic Reasoning – 17th International Conference, LPNMR 2024, Dallas, TX, USA, October 11–14, 2024, Proceedings*, C. Dodaro, G. Gupta and M. V. Martinez, Eds. Lecture Notes in Computer Science, Vol. 15245, Springer, Dallas, TX, USA, 359–372.

https://doi.org/10.1017/S1471068425100380 Published online by Cambridge University Press