Extending Ordered Disjunctions for Policy Enforcement: Preliminary report

Alessandra Mileo¹ and Torsten Schaub²

¹ Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, via Comelico 39, I–20135 Milano, mileo@dico.unimi.it
² Institut für Informatik, Universität Potsdam, Postfach 90 03 27, D–14439 Potsdam, torsten@cs.uni-potsdam.de

Abstract. We consider advanced policy description specifications in the context of Answer Set Programming (ASP). Motivated by our application scenario, we further extend an existing policy description language, so that it allows for expressing preferences among sets of objects. This is done by extending the concept of ordered disjunctions to cardinality constraints. We demonstrate that this extension is obtained by combining existing ASP techniques and show how it allows for handling advanced policy description specifications.

1 Introduction

The specification of policies and their enforcement plays a key role in advanced system environments. Such environments are characterized by complex policies, taking into account a large variety of events, conditions and actions to be executed and monitored. The development and analysis of a collection of such policies can thus constitute a rather complex task, in particular, in view of their overall consistency. To this end, a high-level policy description language called PDL has been developed by Chomicki, Lobo and Naqvi [1] in the context of Network management, through a mapping into Answer Set Programming [2]. The logical approach to policy specification and enforcement proposed in [1] has also been proved to be successfully applicable to Network and Resource Management as well as to the specification and enforcement of Authorization Constraints in Workflows [3].

In PDL, a policy is a set of event-condition-action rules describing how events observed in a system, trigger actions to be executed. A consistency mechanism called monitor is composed of a set of rules describing sets of actions that cannot be executed simultaneously to prevent illegal, hazardous or physically impossible situations.

In a first extension, the PDL language has been augmented by preferences among literals representing actions to be executed [4]. In the resulting extended

---

* Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.
language, called PPDL [3], the high-level policy specification is mapped into Logic Programs with Ordered Disjunction (LPOD) [5].

We briefly recall the idea of Consistency Monitors in PPDL, defined as a set of rules of the form:

\[
\text{never } a_1 \times \cdots \times a_n \text{ if } C.
\]

meaning that actions named \(a_1, \ldots, a_n\) cannot be jointly executed, and in case of constraint violation, expressed by the fact that condition \(C\) holds and actions \(a_1, \ldots, a_n\) have all been triggered by the policy application, \(a_1\) should be preferably blocked, if this is not possible (i.e. \(a_1\) must be performed), \(a_2\) should be blocked, then \(a_3, \ldots, \) if all of \(a_1, \ldots, a_{n-1}\) must be performed, then \(a_n\) must be blocked.

A rule as given in (1) is translated into an ASP program through LPOD encoding as follows:

\[
\begin{align*}
\text{block}(a_1) \times \cdots \times \text{block}(a_n) & \leftarrow \text{exec}(a_1), \ldots, \text{exec}(a_n), C. \\
\text{accept}(A) & \leftarrow \text{not block}(A).
\end{align*}
\]

where \(a_1, \ldots, a_n\) are action names; \(\text{block}(a_i)\) indicates conflicting actions that has to be filtered (i.e. blocked), \(\text{exec}(a_i)\) refers to actions triggered by the application of ECA rules in the policy, and \(\text{accept}(a_i)\) tells us which actions can finally be executed without any constraint violation.

The basic intuition underlying this approach is that the connective “\(\times\)”, called ordered disjunction, is used in rule heads to express a preferential order on atoms to be selected in answer sets of the logic program. As illustrated in [4], the introduction of user-preferences in the specification of PPDL monitor rules enables users to tell the system how to enforce constraints on the execution of actions triggered by the policy. This is done by allowing the user to specify an ordering relation on conflicting actions that have been triggered by the policy; as shown by the LPOD encoding of Rule (1), such a relation is directly reflected by the way actions are to be preferentially blocked in case conflicts arise. Action execution in PPDL can also be controlled by forcing an action to be executed (introducing hard constraints) or by defining criteria to automatically adjust the initially specified ordering relation on objects [3], but we do not treat these aspects here.

From the viewpoint of policy enforcement, it is often the case that an ordering relation among users, resources and more generally, among objects on which actions have to be executed, is not to be expressed on single entities, but on sets of entities having certain characteristics, or being hierarchically organized. This suggests to group entities into clusters (or hierarchies of clusters), in order to express preferences on sets of equally preferred objects. Similar groups (or clusters) can be statically specified by a complex classification of objects (e.g. users in an organization chart, or resources in a reliability chart) determined a-priori. However, given that parameters according to which classification criteria are determined could change as long as actions are performed on objects, it would be better to group entities dynamically. As an example, consider the context of resource management, where reliability of a resource could be
influenced by its use and related performance. Performance measures are dynamically determined, so that preference relation on actions involving resources should be dynamically computed too. Dynamic orderings can be obtained by using logical rules to express preferences on actions performed on sets of entities, eventually influenced by some boolean conditions on entities themselves as well as on parameters’ values. Besides the dynamic nature of our approach, another advantage of this formulation is that preferences on sets are much more intuitive than static classification of objects, as the latter has to be reproduced in rules as in 1 expressing a total preference relation on \( a_1, \ldots, a_n \). Formal aspects related to the specification of preferential monitors in PPDL have been fully addressed in [4,3] by appeal to LPOD programs.

Another interesting aspect we consider is related to the exclusive application of one among the possible rules in the monitor, to solve the same conflict. Each of the LPOD rules expressing the preferential blocking of actions as described above, represents a different criterion, or rather a different strategy (in terms of which actions to block) that can be applied to resolve conflicts on actions execution. Sometimes we may want to express a further ordering among those strategies (i.e. among LPOD rules) saying that one of them is preferred over another. This is the reason why we considered as a further step the combination of LPOD with a relation on LPOD rules as proposed in [6], so as to obtain a more fine-grained ordering among answer sets wrt traditional LPOD.

We address these issues in the policy enforcement context by developing an extension of LPOD that allows for ordered disjunctions of cardinality constraints. We call this extension S-LPOD in that it allows to specify LPOD rules with ordered disjunction on sets of literals. Moreover, we consider the preference relation on LPOD rules introduced by Brewka et al. [6], by discussing some of its properties, and we apply it to S-LPOD rules, resulting in so-called SR-LPOD programs.

Given our application-oriented motivation, we tried to keep our formal development as conservative as possible in relying on existing approaches whenever feasible. Fortunately, this is achievable in straightforward way due to the compositional nature of many ASP extensions.

2 Background

To begin, we recall the basic definitions of Logic Program with Ordered Disjunction (LPOD), as given in [5] and [6]. For basic definitions in Answer Set Programming, we refer the reader to [2].

Given an alphabet \( \mathcal{P} \) of propositional symbols, an LPOD-program is a finite set of LPOD-rules of the form

\[
\begin{align*}
  c_1 \times \cdots \times c_l \leftarrow a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n.
\end{align*}
\]

where each \( a_i, b_j, c_k \) is a literal, that is, an atom \( p \in \mathcal{P} \) or its negation \( \neg p \) for \( 0 \leq i \leq m \), \( 0 \leq j \leq n \), and \( 0 \leq k \leq l \). If \( m = n = 0 \), then (3) is a fact. If

\footnote{See Section 5 for further details related to this context.}
For a rule $r$ as in (3), let $\text{head}(r) = \{c_1, \ldots, c_l\}$ be the head of $r$ and $\text{body}(r) = \{a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n\}$ be the body of $r$; and let $\text{body}^+(r) = \{a_1, \ldots, a_m\}$ and $\text{body}^-(r) = \{b_1, \ldots, b_n\}$.

The “non-standard” part of such a rule is the ordered disjunction $c_1 \times \ldots \times c_l$ constituting its head. Given that the body literals are satisfied, its intuitive reading is:

- if possible $c_1$, but if $c_1$ is impossible, then $c_2$,
- $\ldots$,
- if all of $c_1, \ldots, c_{l-1}$ are impossible, then $c_l$.

Each $c_k$ stands for a choice of rule (3). Note that the “$\times$” connective is allowed to appear in the head of rules only; it is used to define a preference relation that allows to select some of the answer sets of a program by using ranking of literals in the head of the rules, on the basis of a given strategy.

To this end, the semantics of an LPPO program is given in terms of a preference criterion over answer sets. The formal definition of answer sets in LPPO is based on the concept of split programs [7]: Given a rule $r$ as in (3), we define for $1 \leq k \leq l$ the $k$-th option of $r$ as the rule

$$r_k = c_k \leftarrow \text{body}(r), \text{not } c_1, \text{not } c_2, \ldots, \text{not } c_{k-1}.$$

Then, $P'$ is some split program of an LPPO program $P$, if it is obtained from $P$ by replacing each rule in $P$ by one of its options. With this concept, Brewka defines in [5] an answer set of an LPPO program $P$ as a consistent answer set of some split program $P'$ of $P$.

For defining preferred answer sets, Brewka [5] introduces the notion of degree of satisfaction: An answer set $S$ satisfies a rule as in (3)

- to degree 1, if $\text{body}^+(r) \not\subseteq S$ or $\text{body}^-(r) \cap S \neq \emptyset$,
- and otherwise,
- to degree $d = \min\{k \mid c_k \in S\}$.

The degree of rule $r$ in answer set $S$ is denoted by $\text{deg}_S(r)$. Intuitively, the degrees can thus be considered as penalties: the higher the degree, the less we are satisfied about the choice. Brewka shows in [5] that every answer set satisfies all program rules to some degree.

Degrees can be used in various ways for defining a preference relation over answer sets. As an example, we give the definition for the well-known Pareto criterion: An answer set $S_1$ of an LPPO program $P$ is Pareto-preferred to another one $S_2$ ($S_1 >_P S_2$) if there is a rule $r \in P$ such that $\text{deg}_{S_1}(r) < \text{deg}_{S_2}(r)$ and for no $r' \in P$ we have $\text{deg}_{S_1}(r') > \text{deg}_{S_2}(r')$. Then, an answer set $S$ of $P$ is Pareto-preferred among all answer sets, if there is no answer set $S'$ of $P$ that is Pareto-preferred to $S$.

For extending the expressive power of LPPO programs in view of our application, we take advantage of the concept of cardinality constraint [8,9]. Syntactically, a cardinality constraint is a complex literal of the form:

$$l \{a_1, \ldots, a_m\} u$$ (4)
where $l$ and $u$ are two integers giving a lower and upper bound, respectively, on the number of satisfied literals within the constraint\(^4\). For a cardinality constraint $C$ as in (4), we let $\text{lit}(C)$ denote its set of literals $\{a_1, \ldots, a_m\}$ and let $lb(C) = l$ and $ub(C) = u$. $C$ is satisfied by a set of literals $S$, if

$$lb(C) \leq \left| \text{lit}(C) \cap S \right| \leq ub(C).$$

Whenever bound $l$ or $u$ is missing, it is taken to be 0 or $|\text{lit}(C)|$, respectively. In what follows, we restrict ourselves to cardinality constraints, $C$, such that $0 \leq lb(C) \leq ub(C) \leq |\text{lit}(C)|$. For defining answer sets of programs with cardinality constraints, we follow the approach taken in [9].

### 3 From LPOD to S-LPOD

In what follows, we present a straightforward extension of LPOD that allows us to express preferences on sets of atoms, while focusing on its application to policy enforcement.

When considering the problem of policy enforcement studied in [4], it is rather unintuitive that the syntax of rules of the form in (3) requires us to impose a total preference ordering over actions (as with $c_1 \times \cdots \times c_j$); this is much more evident whenever objects on which actions have to be executed (e.g., devices, users, etc.) are somehow classified on the basis of some given parameters. In similar cases, such total ordering may be unrealistic or even unacceptable.

We thus need to introduce a syntactic variation to the rules of (3) in order to accommodate partial preference orderings among actions, according to the classification of objects involved.

**Definition 1.** An S-LPOD program consists of S-LPOD rules of the form

$$C_1 \times \cdots \times C_l \leftarrow A_1, \ldots, A_m, \text{not } B_1, \ldots, \text{not } B_n$$

(5)

where each $A_i, B_j, C_k$ is a cardinality constraint for $0 \leq i \leq m$, $0 \leq j \leq n$, and $0 \leq k \leq l$. A single literal $l$ can be represented by the cardinality constraint $1\{l\}$, as we illustrate below.

For a set of literals $S$ and a cardinality constraint $C$, define the number of literal of $C$ that are in $S$ as $sel(C, S) = |S \cap \text{lit}(C)|$. Then, given a set of literals $S$, the intuitive reading of the rule head of an S-LPOD rule as in (5) can be given as follows:

- if $lb(C_1) \leq sel(C_1, S) \leq ub(C_1)$, then choose $sel(C_1, S)$ elements of $\text{lit}(C_1)$,
- otherwise

\(^4\) The interested reader may note that we confine ourselves to positive literals within cardinality constraints. As detailed below, this is motivated by our application. Also, it simplifies the semantical account of cardinality constraints.
if \( lb(C_2) \leq sel(C_2, S) \leq ub(C_2) \), then choose \( sel(C_2, S) \) elements of \( \text{lit}(C_2) \),
otherwise

\[ \ldots \]

if \( lb(C_1) \leq sel(C_1, S) \leq ub(C_1) \), then choose \( sel(C_1, S) \) elements of \( \text{lit}(C_1) \),
otherwise an incoherent situation is obtained.

The number of elements selected from the chosen cardinality constraint is determined by \( S \). It is nonetheless non-deterministic insofar that different choices of \( S \) yield different selections.

The definition of an option as well as that of a split program carry over from LPOD programs to S-LPOD programs. Answer sets of (split) programs with cardinality constraints are defined as in [9]. Let us illustrate this by building split programs of a S-LPOD along those definitions.

**Example 1.** Let program \( P \) consist of the rules:

\[
r_1 : 1\{a,b\} \times \{c,d,e\}. \\
r_2 : 1\{b,c,d\} \times 1\{a,f\}. 
\]

We obtain 4 split programs:

\[
P_1' : 1\{a,b\}1. \\
1\{b,c,d\}. \\
P_2' : \{c,d,e\} \leftarrow \text{not } 1\{a,b\}1. \\
1\{b,c,d\}. \\
P_3' : \{a,f\} \leftarrow \text{not } 1\{b,c,d\}. \\
P_4' : \{c,d,e\} \leftarrow \text{not } 1\{a,b\}1. \\
1\{a,f\} \leftarrow \text{not } 1\{b,c,d\}.
\]

We obtain the following answer sets\(^5\):

\[
\{a\}, \{b\}, \{c\}, \{d\}, \{f\}, \\
\{a,c\}, \{a,d\}, \{a,f\}, \\
\{b,c\}, \{b,d\}, \{c,e\}, \\
\{c,d\}, \{d,e\}, \{e,f\}, \\
\{a,c,d\}, \{b,c,d\}, \{c,d,e\}
\]

Hence, as with standard LPOD programs, an answer set of an S-LPOD program is simply an answer set of one of its split programs.

To complete the semantics of S-LPOD programs, we first have to account for the definition the degree of satisfaction:

**Definition 2.** A set of literals \( S \) satisfies a rule as in (5)

- to degree 1, if \( A_i \) is not satisfied by \( S \) for some \( 0 \leq i \leq m \) or \( B_j \) is satisfied by \( S \) for some \( 0 \leq j \leq n \), and otherwise,

\(^5\) Each of the answer set reported is an answer set of at least one of the split programs. This is a condition to be answer set of the original program, as shown in [5].
We have seen in Example 1 that S-LPOD programs may yield many answer sets, among which one may still find a substantial number of preferred answer sets. This is even more severe in practice. On the other hand, in practice, it is also very natural to impose additional preferences among S-LPOD rules.

As before, it turns out that ASP-techniques can be composed in a quite straightforward way in order to obtain an extension encompassing the desired features. To this end, we take advantage of ordered logic program, being a pair \((P, <)\), where \(P\) is a logic program and \(< \subseteq P \times P\) is a strict partial order.
Given, $r_1, r_2 \in P$, the relation $r_1 < r_2$ expresses that $r_2$ has higher priority than $r_1$. Then, an SR-LPOD program is an ordered logic program $(P, \prec)$, where $P$ is an S-LPOD program. As before, the formation of preferred answer sets can be made precise in different ways. Among them, we follow the proposal in [6] by using the extended definition of the Pareto-preference criteria proposed in [6, Definition 9]: An answer set $S_1$ of an LPOD program $P$ is Pareto-preferred to another one $S_2$ wrt program $P$, written as $S_1 >_{rp} S_2$, if:

1. there is a rule $r \in P$ such that $\text{deg}_{S_1}(r) < \text{deg}_{S_2}(r)$ and
2. for each $r' \in P$ such that $\text{deg}_{S_1}(r') > \text{deg}_{S_2}(r')$, there is some rule $r''$ such that $r' < r''$ and $\text{deg}_{S_1}(r'') < \text{deg}_{S_2}(r'')$.

We found out that this definition is applicable to SR-LPOD and it allows us to obtain a more fine-grained ordering on answer sets as with S-LPOD program, in that it may introduce additional preferences among answer sets that were considered incomparable or equally preferred according to the original definition of Pareto-preference criteria given in [5], even when preferences on sets of objects are expressed.

We also show that the ordering relation on answer sets of an (S-)LPOD program $P$ is preserved if we add to $P$ preferences on its (S-)LPOD rules. In fact, the following proposition holds:

**Proposition 1.** Let $S_1$ and $S_2$ be answer sets of an (S-)LPOD program $P$. Then $S_1 >_p S_2$ implies $S_1 >_{rp} S_2$.

**Proof.** Let us suppose that $S_1 >_{rp} S_2$ does not hold, and show that $S_1 >_p S_2$ does not hold too. $S_1 >_{rp} S_2$ does not hold if one of the properties in its definition do not hold, i.e.

1. $\forall r \in P, \text{deg}_{S_1}(r) \geq \text{deg}_{S_2}(r)$
2. $\exists r' \in P$ such that $\text{deg}_{S_1}(r') > \text{deg}_{S_2}(r')$ and $\forall r'' > r', \text{deg}_{S_1}(r'') \geq \text{deg}_{S_2}(r'')$.

In the first case, we can immediately conclude that $S_1 >_p S_2$ does not hold by the first part of the definition of preference relation $>_p$.

In the second case, we have that whenever such $r'$ exists, $\text{deg}_{S_1}(r') > \text{deg}_{S_2}(r')$ holds, and thus $S_1 >_p S_2$ by the second part of the definition of preference relation $>_p$.

**Example 3.** Let us consider the S-LPOD program $P$ in Example 1. The ordering relation among the answer sets of $P$ can be represented by considering the following three sets:

$$AS_1 = \{\{a, c\}, \{a, d\}, \{b\}, \{b, c\}, \{b, d\}, \{a, c, d\}, \{b, c, d\}\}$$
$$AS_2 = \{\{a\}, \{c\}, \{d\}, \{a, f\}, \{c, e\}, \{c, d\}, \{d, e\}, \{c, d, e\}\}$$
$$AS_3 = \{\{f\}, \{c, f\}\}$$

---

6 The interested reader may note that we only consider static preferences among S-LPOD rules, i.e. meta-preference statements of the form $r_1 < r_2$ with empty body.
According to the ordering relation derived from the original definition of Pareto-preference criteria in Section 2, we have \( S_i > S_j > S_k \) for \( S_i \in AS_1, S_j \in AS_2, S_k \in AS_3 \). Two answer sets in the same partition are considered incomparable or equally preferred.

If we add the meta-preference on S-LPO rules of \( P \) expressed by \( r_1 < r_2 \), a more fine-grained ordering is achieved and we can identify one partition more, thus specializing the preference relation among previously incomparable answer sets, as follows:

\[
\begin{align*}
AS_1 &= \{\{a, c\}, \{a, d\}, \{b\}, \{b, c\}, \{b, d\}, \{a, c, d\}, \{b, c, d\}\} \\
AS_2 &= \{\{c\}, \{d\}, \{c, e\}, \{c, d\}, \{d, e\}, \{c, d, e\}\} \\
AS_3 &= \{\{a\}, \{a, f\}\} \\
AS_4 &= \{\{f\}, \{e, f\}\}
\end{align*}
\]

where \( S_i > S_j > S_k > S_l \) for \( S_i \in AS_1, S_j \in AS_2, S_k \in AS_3, S_l \in AS_4 \).

One may argue that these new meta-preferences among S-LPO rules do not significantly change the solution, since the Pareto-preferred answer sets of \( P \) are the same. But suppose that there are integrity constraints preventing us from considering any of the most preferred answer sets as a solution, e.g., the following constraints are added to program \( P \):

\[
\begin{align*}
r_{c1} &\leftarrow a, c. \\
r_{c2} &\leftarrow b. \\
r_{c3} &\leftarrow d.
\end{align*}
\]

As a result, we have to look at answer sets in \( AS_2 \) as the preferred ones; preference ordering among them has been refined by the new preference relation among S-LPO rules, so that the solution is reduced to answer sets \( \{c\}, \{c, e\} \) as the preferred ones.

The following complete example illustrates again how the refined definition of Pareto-preference including preferences among rules is meaningful even with simple LPO programs.

**Example 4.** Let program \( P_{pref} \) consist of the LPO rules:

\[
\begin{align*}
r_1 &: a \times c. \\
r_3 &: b \times d. \\
r_4 &: b \times a. \\
r_3 &: d \times c. \\
r_{p1} &: r_3 > r_1. \\
r_{p2} &: r_4 > r_1. \\
r_{p3} &: r_3 > r_2. \\
r_{p4} &: r_4 > r_2. \\
r_{c1} &: \leftarrow a, d.
\end{align*}
\]

where rules \( r_{pi} \) represent preference relations among rules \( r_k \) of \( P_{pref} \).
We obtain 16 split programs, among which we report only the coherent ones:

\[ P'_1 : c \leftarrow \text{not } a. \quad P'_2 : a. \]
\[ b. \quad b. \]
\[ d. \quad c \leftarrow \text{not } d. \]
\[ P'_3 : c \leftarrow \text{not } a. \quad P'_4 : c \leftarrow \text{not } a. \]
\[ b. \quad b. \]
\[ a \leftarrow \text{not } b. \quad c \leftarrow \text{not } d. \]
\[ d. \]
\[ P'_5 : c \leftarrow \text{not } d. \quad P'_6 : c \leftarrow \text{not } a. \]
\[ b. \quad d \leftarrow \text{not } b. \]
\[ a \leftarrow \text{not } b. \quad b. \]
\[ c \leftarrow \text{not } d. \quad c \leftarrow \text{not } d. \]
\[ P'_7 : \]
\[ d \leftarrow \text{not } b. \]
\[ b. \]
\[ c \leftarrow \text{not } d. \]
\[ c \leftarrow \text{not } d. \]

We obtain the following answer sets for the original program \( P_{\text{pref}} \) with associated degrees of satisfaction:

\[ S_1 = \{ a, b, c \} \quad \text{deg}_{S_1}(r_1) = \text{deg}_{S_1}(r_2) = \text{deg}_{S_1}(r_3) = 1, \quad \text{deg}_{S_1}(r_4) = 2 \]
\[ S_2 = \{ b, c, d \} \quad \text{deg}_{S_2}(r_2) = \text{deg}_{S_2}(r_3) = \text{deg}_{S_2}(r_4) = 1, \quad \text{deg}_{S_1}(r_1) = 2 \]
\[ S_3 = \{ b, c \} \quad \text{deg}_{S_3}(r_1) = \text{deg}_{S_3}(r_4) = 2, \quad \text{deg}_{S_3}(r_2) = \text{deg}_{S_3}(r_3) = 1 \]

According to the Pareto-preference ordering, under LPOD semantics, \( S_1 \) and \( S_2 \) are the preferred answer sets for \( P_{\text{pref}} \); moreover, we have that \( S_1 >_{\text{pr}} S_3 \) and \( S_2 >_{\text{pr}} S_3 \).

The extended notion of preference relation that considers preferences among LPOD rules (rules \( r_{pi} \), \( i = 1..4 \)), gives us a more fine-grained ordering on answer sets \( S_1 \) and \( S_2 \) that were incomparable under the LPOD semantics, in that \( S_2 >_{\text{pr}} S_1 \).

As a consequence, only \( S_2 \) results being the Pareto-preferred answer set of \( P_{\text{pref}} \) according to \( >_{\text{pr}} \) ordering relation.

5 Application to policy enforcement

As illustrated in [4], the logical approach to policy enforcement in PPDL allows users to specify complex policies in an intuitive way, and independently from the details of the particular device executing it. In fact, PPDL is a rather simple, easy-to-grasp language which originates from the well-known Event-Condition-Action paradigm of active databases and allows to define their policies and consistency mechanisms in a transparent and easy way. PPDL specifications are directly mapped into ASP and can thus be computed very efficiently by invoking performant ASP solvers [6]. On the other hand, such a high-level specification of policies allows to keep the so-called business logic outside the specific system representation so that policies can be easily inspected and changed any time.
Roughly speaking, a Consistency Monitor is a mechanism to specify hazardous, insecure or physically impossible situations that should be avoided. When applying the policy yields a set of actions that actually violates one of the rules as in (1), the consistency monitor enforcement allows to filter actions that follow from the policy application, canceling some of them. Thus, the ordering relation on actions specified in the PPDL monitor rules, enables the administrator to specify how conflicts are to be solved, i.e., to control the—otherwise non-deterministic—conflict resolution process and the order according to which actions have to be canceled.

Although the encoding of PPDL into LPOD proposed in [4] is intuitive and computationally easy to be automatically performed, it requires us to impose a total preference ordering over actions to be blocked in a single constraint specification. Such a total ordering can be unrealistic or even unacceptable in applications, as it would force us to specify all possible combinations of a totally ordered list of actions. We could need to group objects and consequently actions performed on those objects, according to some common properties, thus adding a level of nondeterminism to the choice of which actions to block in order to solve a conflict but keeping the PPDL specification intuitive and the mapping into ASP computationally simple.

Example 5. Consider again a Resource Manager dealing with critical resources and a wide number of users. It would be desirable to tell that a certain category of users should be prevented from using a set of critical resources if they are clerks, but if they are managers, resources availability is to be granted.

The above mentioned scenario suggests that we need to introduce a syntactic variation to policy and monitor specification, in order to accommodate partial preference orderings among sets of actions.

It is worth mentioning the fact that, in our policy specification, we allow only positive atoms to appear in the constraints, as each literal represents an action and we do not consider the case in which a set of events causes an action not to be executed. Of course this is a possibility and things could be generalized, but we don’t deal with this case here.

According to the LPOD extensions we investigated in sections 3 and 4, we now extend the language of PPDL, mentioned in Section 1, into SR-PPDL, by providing a more general definition of a monitor expressing preferences on sets of actions that have to be blocked to solve conflicts arisen from policy enforcement.

Let $(A, <)$ be a partially ordered set of actions. We define a level mapping $\ell$ as follows.

\begin{itemize}
  \item $\ell(a) = 1$ iff $\exists a', a' < a$.
  \item $\ell(a) = i + 1$ iff $\max\{\ell(a') : a' < a\} = i$.
\end{itemize}

\footnote{A more complete example in this application context will be detailed later on in this section.}
The level function partitions $A$ into disjoint sets of actions: $A = A_1 \cup \cdots \cup A_r$, where each $A_i$ contains actions with the same preference level $i$ and $A_g \cap A_l = \emptyset$ for all $g \neq l$

The preference relation defined by $\langle A, < \rangle$ can be expressed by the extended syntax of SR-PPDL monitor rule (extending the one proposed in [4]) as follows:

$$r : \text{never } l_1[A_1]u_1 \times \cdots \times l_r[A_r]u_r \text{ if } C. \quad (7)$$

where each $A_i$ represents a set of atoms $\{a_i^1, a_i^2, \ldots, a_i^m\}$, $C$ is a Boolean condition and each element $l_i[A_i]u_i$ represents a cardinality constraint of the form in Equation (4).

Given that $D_i$ is the set of actions in $A_i$ triggered by the policy application, the cardinality constraint $C(A_i) = l_i\{a_i^1, a_i^2, \ldots, a_i^m\}u_i$ is satisfied if $l_i \leq |D_i| \leq u_i$. For each constraint $C(A_i)$ that is satisfied, we define the set of actions to be blocked $X_i$ as the minimum subset of $D_i$ for which $|D_i - X_i| \leq l_i - 1$. As a consequence, we have that, for each of these $X_i$, $|X_i| = |D_i| - l_i + 1$.

Equation (7) tells us that when all cardinality constraints $C(A_i)$, $i = 1..r$ are satisfied, then actions in $D_1$, actions in $D_2$, ..., actions in $D_r$ cannot be executed together and, in case of constraint violation, $|X_1|$ actions in $D_1$ should be preferably blocked; if it is not possible, block $|X_2|$ actions in $D_2$; ..., if all of the actions in $D_j$, $j = 1..r - 1$ must be performed, then block $|X_r|$ actions in $D_r$.

In this way, the total ordering among conflicting actions to be blocked can be released by admitting that actions at a certain level $i$ in the head of an SR-LPO rule can be non-deterministically chosen from a set $D_i \subseteq A_i$ of equally preferred actions triggered by the policy, given that $C(A_i)$ is satisfied and all other actions in $D_j$ with $l_j \leq |D_j| \leq u_j$ and level $j < i$, must be executed.

To express such non-determinism, we translate the SR-PPDL rule in Equation (7) into SR-LPO by using cardinality constraints for each set of equally preferred literals.

Thus, according to the original PPDL encoding in [4], given that

$$
A_1 = \{a_1^1, a_1^2, \ldots, a_1^g\} \\
A_2 = \{a_2^1, a_2^2, \ldots, a_2^h\} \\
\vdots \\
A_r = \{a_r^1, a_r^2, \ldots, a_r^m\}
$$

each rule of the form in Equation (7) will result into $\prod_{i=1}^r (u_i - l_i + 1)$ SR-LPO rules representing all possible combination of sets of elements in the head of the SR-LPO rules:

* Otherwise, if at least one of the $C(A_i)$ is not satisfied, there is no conflict and rule $r$ of the form in Equation 7 is not triggered.
The last constraint has been introduced into the mapping from SR-PPDL to SR-LPOD in order to assure that, in non-determinism induced by cardinality constraints on sets, actions blocked are among those triggered by the policy.

Corresponding split programs are built in the same way as illustrated in Section 2. This may generate a lot of possibilities that can be further reduced when we introduce a preferential ordering of the form \( r_i \succ r_j \) where \( r_i \) and \( r_j \) are SR-PPDL rules of the form in Equation (7). Combination of our LPOD extensions into the high-level policy language is illustrated in the following example.

**Example 6.** Let us consider the problem of allocation of resources \( a, b, c \) and \( d \) among two users, \( u1 \) and \( u2 \). Resources \( a \) and \( b \) are much more critical to the system than resources \( c \) and \( d \) (\( a \) and \( b \) should be preferentially not assigned to users if this is not necessary, i.e. actions corresponding to the assignment of resources \( a \) and \( b \) should be preferentially blocked in case conflicts arise), while user \( u2 \) is to be preferentially served over \( u1 \).

To reproduce the situation described above, we need two preferential monitor rules that look like:

\[
\begin{align*}
    r_1 & : \text{never } 1[\text{ass}(u1, R)]1 \times 1[\text{ass}(u2, R)]1. \\
    r_2 & : \text{never } 1[\text{ass}(U, a), \text{ass}(U, b)]2 \times 1[\text{ass}(U, c), \text{ass}(U, d)]2.
\end{align*}
\]

where \( U \) and \( R \) are variable to be grounded on the set of users and resources respectively.
Moreover, user $u1$ cannot be prevented from using both resources $b$ and $c$, but he needs at least one of them. For simplicity, we focus on the consistency monitor specification omitting policy rules. Suppose that the policy application yields user $u1$ to obtain resources $b, c, d$, and user $u2$ to obtain resources $a, b, c$, so that in the translation to SR-LPOD program we simply omit predicates $\text{exec}(u1, b), \text{exec}(u1, c), \text{exec}(u1, d), \text{exec}(u2, a), \text{exec}(u2, b), \text{exec}(u2, c)$ considering them true, and we report only triggered rules. Thus, we obtain the following SR-LPOD program $P_{sr-lpod}$:

$$r_1^1 : 1 \{ \text{block}(u1, b) \} 1 \times 1 \{ \text{block}(u2, b) \} 1.$$

$$r_1^2 : 1 \{ \text{block}(u1, c) \} 1 \times 1 \{ \text{block}(u2, c) \} 1.$$

$$r_2^1 : 1 \{ \text{block}(u1, a), \text{block}(u1, b) \} 1 \times 2 \{ \text{block}(u1, c), \text{block}(u1, d) \} 2.$$

$$r_2^2 : 2 \{ \text{block}(u2, a), \text{block}(u2, b) \} 2 \times 1 \{ \text{block}(u2, c), \text{block}(u2, d) \} 1.$$

$$\leftarrow \text{block}(u1, b), \text{block}(u1, c).$$

$$\leftarrow \text{block}(U, R), \text{not exec}(U, R), \text{res}(R), \text{usr}(U).$$

$$\text{accept}(U, R) \leftarrow \text{not block}(U, R), \text{res}(R), \text{usr}(U).$$

We obtain three answer sets:

$$S_1 = \{ \text{block}(u1, b), \text{block}(u2, c) \}$$

$$S_2 = \{ \text{block}(u1, c), \text{block}(u1, d), \text{block}(u2, b), \text{block}(u2, c) \}$$

$$S_3 = \{ \text{block}(u1, b), \text{block}(u2, a), \text{block}(u2, b), \text{block}(u2, c) \}$$

with $S_3 >_p S_1$. Thus, $S_2$ and $S_3$ are the preferred answer sets for program $P_{sr-lpod}$ in terms of blocked assignments.

If we add a rule preference $r_2 > r_1$ on rules of the monitor. This means that, in their grounded instances, each of the rules $r_2^1$ is preferred to each of the rules $r_1^1$. According to relation $>_r$, we now have $S_3 >_r S_1$ and $S_3 >_r S_2$, thus obtaining only $S_3$ as the preferred answer set.

To accomplish specific systems requirements, additional constraints could be added, such as that each user has to be assigned to at least a resource, or that a resource cannot be assigned to two different users, thus restricting the set of admissible solutions.

It’s easy to imagine that when a wide number of combinations are possible according to how resources/users are grouped into sets, introducing a further level of preferences on rules that determine (S-)LPOD preferences, can results in more accurate solutions by reducing the set of Pareto-preferred solutions.

---

9. This does not change the way priorities are computed, because by definition of degree of satisfaction of a rule wrt a set of literals, not triggered rules have degree equal to 1.

10. Note that the Pareto-preference relation $>_p$ is preserved.
6 Conclusion

We have considered advanced policy description specifications based on advanced semantics of Answer Set Programming. Our analysis is aimed at providing a tool to enforce complex policy consistency mechanisms, enriched with qualitative preferential information by using the high-level policy description language PPDL investigated in [4]. To this end, we extended the logical formalism by allowing ordered disjunctions over cardinality constraints and we used a rule-based Pareto-preference criterion for distinguishing preferred answer sets. Next step is to extend the PPDL language syntax in this direction, by mapping extended monitor constructs of the resulting SR-PPDL into SR-LPOD.

Although our technical contribution is a straightforward one, it can be regarded as a proof-of-concept in how far existing ASP extensions can be combined in order to tackle practical applications. We believe in the potential of high-level specification languages to control and monitor complex systems efficiently. In fact, the proposed extension to qualitative preference handling and specification from policy and monitor enforcement perspectives could enable us to find new contexts of application in other fields of AI.

Future work will address implementation issues. To be more precise, we want to adapt the compilation technique proposed in [6] to our approach. Also, it is worthwhile to check under which restrictions a specification can be compiled in a normal logic program (without any need for genuine disjunctions).

References