Qualitative constraint enforcement in advanced policy specification

Alessandra Mileo[1] and Torsten Schaub[2⋆]

[1] Dipartimento di Informatica e Comunicazione, Universitá degli Studi di Milano-Bicocca — Lab. Nomadis, via Bicocca degli Arcimboldi 8, I–20126 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, where a large variety of events, conditions and actions are to be executed and monitored. The development and analysis of a collection of such policies can be rather complex, 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 (ASP;[2]).

A first extension of PDL lead to the description of PPDL language [4, 3]. In PPDL a *policy* is a set of event-condition-action rules describing how events observed in a system, trigger actions to be executed, and a *consistency monitors* is a set of rules of the form:

$$\textbf{never } a_1 \times \ldots \times a_n \textbf{ if } C. \tag{1}$$

meaning that actions named $a_1, \ldots, a_n$ cannot be jointly executed. In case 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 if all of a_1, ..., a_{n-1} must be performed, then a_n must be blocked.

⋆ Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

A rule as in (1) is mapped into ASP through LPOD [5] encoding as follows:

$$\begin{array}{ll} block(a_1) \times \ldots \times block(a_n) & \leftarrow exec(a_1), \ldots, exec(a_n), C. \\ accept(A) & \leftarrow not\ block(A). \end{array} \quad (2)$$

where $block(a_i)$ indicates conflicting actions that have to be blocked, $exec(a_i)$ refers to actions triggered by the policy application, and $accept(a_i)$ tells us which actions can be executed without any constraint violation.

As illustrated in [4], the introduction of user-preferences in PPDL *monitor* rules enables users to tell the system in which order to enforce constraints on the execution of actions triggered by the policy.

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 to be expressed on sets of entities having certain characteristics or being hierarchically organized.

As an example, consider the context of resource management: reliability of a resource could be influenced by its use and (dynamically determined) performance. Preference relation on actions involving resources has to be dynamic too.[3] Besides the dynamic nature of our logic-based approach, preferences on sets are much more intuitive than static classification of objects. 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 is related to expressing a further ordering among strategies (represented by monitor rules) for conflicts resolution.

We address these issues in the policy enforcement context by extending LPOD to allow for ordered disjunctions of cardinality constraints and we call this extension S-LPOD. 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

$$c_1 \times \ldots \times c_l \leftarrow a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n. \quad (3)$$

[3] See Section 5 for further details related to this context.

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 $l = 1$, then we have a normal rule. If $l = 0$, then (3) is an integrity constraint. (cf. [2]) For a rule r as in (3), let $head(r) = \{c_1, \ldots, c_l\}$ be the *head* of r and $body(r) = \{a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n\}$ be the *body* of r; and let $body^+(r) = \{a_1, \ldots, a_m\}$ and $body^-(r) = \{b_1, \ldots, b_n\}$.

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,
- ...,
- 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 LPOD program is given in terms of a preference criterion over answer sets. The formal definition of answer sets in LPOD 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

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

Then, P' is some *split program* of an LPOD 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 LPOD 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 $body^+(r) \not\subseteq S$ or $body^-(r) \cap S \neq \emptyset$, and otherwise,
- to degree $d = \min\{k \mid c_k \in S\}$.

The degree of rule r in answer set S is denoted by $deg_S(r)$. 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 LPOD 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 $deg_{S_1}(r) < deg_{S_2}(r)$ and for no $r' \in P$ we have $deg_{S_1}(r') > deg_{S_2}(r')$. 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 LPOD 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:

$$l\ \{a_1, \ldots, a_m\}\ u \tag{4}$$

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 $lit(C)$ 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

$$lb(C) \leq |lit(C) \cap S| \leq ub(C)\ .$$

Whenever bound l or u is missing, it is taken to be 0 or $|lit(C)|$, respectively. In what follows, we restrict ourselves to cardinality constraints, C, such that $0 \leq lb(C) \leq ub(C) \leq |lit(C)|$. 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.

In policy enforcement contexts [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 $c_1 \times \ldots \times c_j$), in particular when objects on which actions have to be executed (e.g. devices, users, etc.) are 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*

$$C_1 \times \ldots \times C_l \leftarrow A_1, \ldots, A_m, not\ B_1, \ldots, not\ B_n \tag{5}$$

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 $sel(C, S) = |S \cap lit(C)|$. 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:

[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.

– if $lb(C_1) \leq sel(C_1, S) \leq ub(C_1)$, then choose $sel(C_1, S)$ elements of $lit(C_1)$, otherwise
– if $lb(C_2) \leq sel(C_2, S) \leq ub(C_2)$, then choose $sel(C_2, S)$ elements of $lit(C_2)$, otherwise
– ...
– if $lb(C_l) \leq sel(C_l, S) \leq ub(C_l)$, then choose $sel(C_l, S)$ elements of $lit(C_l)$,
– 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:

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

We obtain 4 split programs:

$$\begin{array}{ll} P_1' : 1\{a, b\}1. & P_2' : 1\{a, b\}1. \\ \quad 1\{b, c, d\}. & \quad 1\{a, f\} \leftarrow not\ 1\{b, c, d\}. \\ P_3' : \{c, d, e\} \leftarrow not\ 1\{a, b\}1. & P_4' : \{c, d, e\} \leftarrow not\ 1\{a, b\}1. \\ \quad 1\{b, c, d\}. & \quad 1\{a, f\} \leftarrow not\ 1\{b, c, d\}. \end{array}$$

We obtain the following answer sets for program P[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 of 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,*
– *to degree $d = \min\{k \mid lb(C_k) \leq sel(C_k, S) \leq ub(C_k)\}$.*

As above, we denote the degree of rule r in answer set S as $deg_S(r)$. As with standard LPOD, our extended definition assures that if an answer set S of an S-LPOD program exists, then S satisfies all rules of P to some degree.

As well, we can use the degree of satisfaction to induce different preference criteria on the answer sets of an S-LPOD program. In particular, the criterion of *Pareto-preference* given above carries over from LPOD to S-LPOD.

[5] Each of the answer set reported is an answer set of at least one of the split programs. This is a necessary condition to be answer set of the original program [5].

Example 2. Consider again the program P given in Example 1. All Pareto-preferred answer sets satisfy both rules of program P with degree 1:

$$\{a, c\},\ \{a, d\},\ \{b\},\ \{b, c\},\ \{b, d\},\ \{a, c, d\},\ \{b, c, d\}.$$

Thus, we can have more than one preferred answer set and each of them is also an answer set of some split program of the original program.

Finally, let us show that S-LPOD is *conservative* insofar as it corresponds to LPOD whenever we have no cardinality constraints. To see this, consider rule (5), where each cardinality constraint is of the form $1\{l\}$ for some literal $l \in \mathcal{L}$:

$$1\{c_1\} \times \ldots \times 1\{c_l\} \leftarrow 1\{a_1\}, \ldots, 1\{a_m\}, not\ 1\{b_1\}, \ldots, not\ 1\{b_n\}. \quad (6)$$

A set of literals S satisfies such a rule r

- to degree 1, if $1\{a_i\}$ is not satisfied by S for some $0 \leq i \leq m$ or $1\{b_j\}$ is satisfied by S for some $0 \leq j \leq n$, and otherwise,
- to degree $d = \min\{k \mid lb(c_k) \leq sel(c_k, S) \leq ub(c_k)\}$.

In this special case, we have $sel(l, S) = |S \cap lit(l)| = |S \cap \{l\}|$, and $lb(l) = 1$ and $ub(l) = |lit(l)|$. While the first condition is equivalent to $body^+(r) \not\subseteq S$ or $body^-(r) \cap S \neq \emptyset$, the latter gives $d = \min\{k \mid 1 \leq |S \cap \{l\}| \leq |lit(l)|\}$. In order to respect the bounds $1 \leq |S \cap \{l\}| \leq 1$, we must have $l \in S$, so that the degree of satisfaction of rule r is $d = \min\{k \mid l \in S\}$, which is what we have in the definition of degree of satisfaction for LPODs.

4 From S-LPOD to SR-LPOD

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. In fact, 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 $(P, <)$, 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 $S_1 >_{pr} S_2$, if

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

We found out that this definition is applicable to SR-LPOD[6] 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. $\forall r \in P,\ deg_{S_1}(r) \geq deg_{S_2}(r)$
2. $\exists r' \in P :\ deg_{S_1}(r') > deg_{S_2}(r')$ and $\forall r'' > r',\ deg_{S_1}(r'') \geq deg_{S_2}(r'')$.

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, $deg_{S_1}(r') > deg_{S_2}(r')$ 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:

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

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-LPOD 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:

$$AS_1 = \{\{a,c\},\ \{a,d\},\ \{b\},\ \{b,c\},\ \{b,d\},\ \{a,c,d\},\ \{b,c,d\}\}$$
$$AS_2 = \{\{c\},\ \{d\},\ \{c,e\},\ \{c,d\},\ \{d,e\},\ \{c,d,e\}\}$$
$$AS_3 = \{\{a\},\ \{a,f\}\} \qquad AS_4 = \{\{f\},\ \{e,f\}\}$$

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

[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.

One may argue that these new meta-preferences among S-LPOD 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 three constraints are added to program P:

$$r_{c1}: \quad \leftarrow a, c. \qquad r_{c2}: \quad \leftarrow b. \qquad r_{c3}: \quad \leftarrow d.$$

As a result, all answer sets in AS_1 are eliminated and some in AS_2 are the preferred ones; preference ordering among them has been refined by the new preference relation among S-LPOD rules, so that the solution is reduced to answer sets $\{c\},\ \{c, e\}$ as the preferred ones.

The following example illustrates how Pareto-preference including preferences among rules can be meaningful even with simple LPOD programs.

Example 4. Let program P_{pref} consist of the LPOD rules:

$$\begin{array}{llll} r_1: & a \times c. & r_{p_1}: r_3 > r_1. & r_{c1}: \leftarrow a, d. \\ r_2: & b \times d. & r_{p_2}: r_4 > r_1. & \\ r_3: & b \times a. & r_{p_3}: r_3 > r_2. & \\ r_4: & d \times c. & r_{p_4}: r_4 > r_2. & \end{array}$$

where rules r_{pi} represent preference relations among rules r_k of P_{pref}.

We can compute 16 split programs[7] obtaining from them the following answer sets for the original program P_{pref}:

$$\begin{array}{ll} S_1 = \{a, b, c\} & deg_{S_1}(r_1) = deg_{S_1}(r_2) = deg_{S_1}(r_3) = 1,\ deg_{S_1}(r_4) = 2 \\ S_2 = \{b, c, d\} & deg_{S_2}(r_2) = deg_{S_2}(r_3) = deg_{S_2}(r_4) = 1,\ deg_{S_2}(r_1) = 2 \\ S_3 = \{b, c\} & deg_{S_3}(r_1) = deg_{S_3}(r_4) = 2,\ deg_{S_3}(r_2) = deg_{S_3}(r_3) = 1 \end{array}$$

According to the Pareto-preference ordering, under LPOD semantics, S_1 and S_2 are the preferred answer sets for P_{pref}; moreover, $S_1 >_p S_3$ and $S_2 >_p S_3$. The extended notion of preference relation on LPOD rules (expressed in rules $r_{p_i},\ i = 1..4$), gives us a more fine-grained ordering on answer sets S_1 and S_2 that were incomparable under the LPOD semantics, in that $S_2 >_{pr} S_1$.

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

5 Application to policy enforcement

As illustrated in [4], PPDL is a rather simple, easy-to-grasp language allowing to define policies and consistency mechanisms in a transparent and easy way by keeping the so-called *business logic* outside the specific system representation. PPDL specifications are directly mapped into ASP and can thus be computed very efficiently by invoking performant ASP solvers [6].

[7] Note that only the coherent ones are used for computing solutions.

Although the encoding of PPDL into LPOD proposed in [4] is intuitive and computationally easy to be automatized, 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 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 non-determinism 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. As an example, consider again a Resource Manager. We may want to tell that a clerk should be prevented from accessing critical resources, but resources availability is to be granted to managers[8]. 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 Section 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 $\langle A, < \rangle$ be a partially ordered set of actions. We define a level mapping ℓ as follows.

- $\ell(a) = 1$ iff $\nexists a', a' < a$.
- $\ell(a) = i + 1$ iff $\max\{\ell(a') : \ a' < a\} = i$.

The level function partitions A into disjoint sets of actions: $A = A_1 \cup \ldots \cup A_r$, 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:

$$r \ : \ \textbf{never} \ l_1[A_1]u_1 \times \ldots \ \times \ l_r[A_r]u_r \ \textbf{if} \ C. \tag{7}$$

where each A_i represents a set of atoms $\{a_1^i, a_2^i, \ldots, a_m^i\}$, 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 $C(A_i) = l_i\{a_1^i, a_2^i, \ldots, a_m^i\}u_i$ 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, $|X_i| = |D_i| - l_i + 1$.

[8] A complete example in this context will be detailed later on in this section.

Equation (7) tells us that when *all* cardinality constraints $C(A_i),\ i = 1..r$ are satisfied[9], 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, 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-LPOD 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-LPOD by using cardinality constraints for each set of equally preferred literals. Thus, according to the original PPDL encoding [4], given that

$$A_1 = \{a_1^1, a_2^1, \ldots, a_g^1\} \qquad A_2 = \{a_1^2, a_2^2, \ldots, a_h^2\} \qquad \ldots \qquad A_r = \{a_1^r, a_2^r, \ldots, a_m^r\}$$

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

$$\begin{aligned}
&l_1\{block(a_1^1),\ block(a_2^1),\ \ldots,\ block(a_g^1)\}l_1 \times \\
&l_2\{block(a_1^2),\ block(a_2^2),\ \ldots,\ block(a_h^2)\}l_2 \times \\
&\ldots \times \\
&l_r\{block(a_1^r),\ block(a_2^r),\ \ldots,\ block(a_m^r)\}l_r \ \leftarrow\ l_1\{exec(a_1^1), \ldots,\ exec(a_g^1)\}l_1, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad l_2\{exec(a_1^2), \ldots,\ exec(a_h^2)\}l_2, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \ldots \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad l_r\{exec(a_1^r), \ldots,\ exec(a_m^r)\}l_r,\ C. \\
&\ldots \\
&u_1\{block(a_1^1),\ block(a_2^1),\ \ldots,\ block(a_g^1)\}u_1 \times \\
&u_2\{block(a_1^2),\ block(a_2^2),\ \ldots,\ block(a_h^2)\}u_1 \times \\
&\ldots \times \\
&u_r\{block(a_1^r),\ block(a_2^r),\ \ldots,\ block(a_m^r)\}u_r \ \leftarrow\ u_1\{exec(a_1^1), \ldots,\ exec(a_g^1)\}u_1, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad u_2\{exec(a_1^2), \ldots,\ exec(a_h^2)\}u_2, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \ldots \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad u_r\{exec(a_1^r), \ldots,\ exec(a_m^r)\}u_r,\ C. \\
&accept(A) \ \leftarrow\ exec(A),\ not\ block(A). \\
&\leftarrow block(A),\ not\ exec(A).
\end{aligned}$$

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, further reduced when we introduce a preferential ordering of the form $r_i > r_j$ 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.

[9] 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.

Example 5. 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 critical (actions corresponding to the assignment of a and b should be preferentially blocked in case conflicts arise), user $u2$ is to be preferentially served over $u1$.

The corresponding monitor rules look like:

$$r_1 \ : \textbf{never}\ 1[ass(u1, R)]1 \times 1[ass(u2, R)]1.$$
$$r_2 \ : \textbf{never}\ 1[ass(U, a), ass(U, b)]2 \times 1[ass(U, c), ass(U, d)]2.$$

where U and R are grounded on the set of users and resources, respectively.

Suppose that the policy application yields user $u1$ to obtain resources b, c, d, and user $u2$ to obtain resources a, b, c[10] and $u1$ needs at least one resource among b and c.

The resulting SR-LPOD program $P_{sr-lpod}$ is as follows:

$$r_1^1 : 1\{block(u1, b)\}1 \times 1\{block(u2, b)\}1.$$
$$r_1^2 : 1\{block(u1, c)\}1 \times 1\{block(u2, c)\}1.$$
$$r_2^1 : 1\{block(u1, a), block(u1, b)\}1 \times 2\{block(u1, c), block(u1, d)\}2.$$
$$r_2^2 : 2\{block(u2, a), block(u2, b)\}2 \times 1\{block(u2, c), block(u2, d)\}1.$$
$$\leftarrow block(u1, b), block(u1, c).$$
$$\leftarrow block(U, R),\ not\ exec(U, R),\ res(R),\ usr(U).$$
$$accept(U, R) \leftarrow not\ block(U, R),\ res(R),\ usr(U).$$

We obtain three answer sets of $P_{sr-lpod}$:

$$S_1 = \{block(u1, b), block(u2, c)\}$$
$$S_2 = \{block(u1, c), block(u1, d), block(u2, b), block(u2, c)\}$$
$$S_3 = \{block(u1, b), block(u2, a), block(u2, b), block(u2, c)\}$$

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

Suppose we now 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^i is preferred to each of the rules r_1^j. According to relation $>_{rp}$, we now have $S_3 >_{rp} S_1$[11] and $S_3 >_{rp} S_2$, 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 one 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.

[10] For simplicity, we focus on the consistency monitor specification omitting policy rules. This does not change the way priorities are computed, cause not triggered rules have degree equal to 1 by definition.

[11] 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.

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 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: 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

1. Chomicki, J., Lobo, J., Naqvi, S.: Conflict resolution using logic programming. IEEE Transactions on Knowledge and Data Engineering **15**(1) (2003) 244–249
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
3. Mileo, A.: Preference Specification and Enforcement in Declarative Policies. PhD thesis, Universitá degli Studi di Milano (2006)
4. Bertino, E., Mileo, A., Provetti, A.: PDL with preferences. Proc. of *POLICY 2005*.
5. Brewka, G.: Logic programming with ordered disjunction. Proc. of *AAAI02*. Extended version presented at NMR02 (2002)
6. Brewka, G., Niemelä, I., Syrjänen, T.: Implementing ordered disjunction using answer set solvers for normal programs. Proc. of *JELIA02* (2002) 444–455
7. Sakama, C., Inoue, K.: An alternative approach to the semantics of disjunctive logic programs and deductive databases. J. Autom. Reasoning **13**(1) (1994) 145–172
8. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. **138**(1-2) (2002) 181–234
9. Liu, L., Truszczynski, M.: Properties of programs with monotone and convex constraints. Proc. of *AAAI05*. 701–706