Preferences and Priorities in ASP

Stefania Costantini[1] and Andrea Formisano[2]

[1] Università di L'Aquila
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica (DISIM)
Via Vetoio Loc. Coppito, I-67010 L'Aquila, Italy
stefcost@di.univaq.it
[2] Università di Perugia
Dipartimento di Matematica e Informatica
via Vanvitelli, 1, I-06123 Perugia, Italy
formis@dmi.unipg.it

Abstract. In this paper we extend our previous work concerning preferences in Answer Set Programming (ASP), and integrate this work with $\mathcal{PDL}$, a well-known approach to answer set optimization. We thus obtain a powerful language for expressing preferences in ASP.

Key words: Answer set programming, preferences, priorities.

1 Introduction

Logic programming under the answer set semantics (Answer Set Programming, for short ASP) is nowadays a well-established programming paradigm, with applications in many areas, including problem solving, configuration, information integration, security analysis, agent systems, semantic web, and planning (see among many [2, 1, 19, 24, 16] and the references therein).

As ASP has proved useful in formalizing many forms of commonsense reasoning, and as in turn many forms of commonsense reasoning rely more or less explicitly upon some form of preferences, preferential reasoning has also been studied in relation to ASP (cf., e.g., [14, 6, 3, 5, 4, 8, 10]).

In this paper we consider the work of [4] about answer set optimization, that proposes the language $\mathcal{PDL}$ for describing complex preferences among answer sets, so as to rank answer sets according to several criteria, and find the optimal ones. We compare the basic building blocks of $\mathcal{PDL}$, *preference rules*, with RASP rules that we have proposed in [8, 10]. First we show that our rules fit into the $\mathcal{PDL}$ framework to which they bring some useful extension. Then, elaborating on our past work and on principles discussed in [25, 20], we further improve the forms of preferences that can be expressed. The reader may refer to e.g., [14, 6, 4, 8, 10] and to the references therein for recent overviews and discussion of the many existing approaches to preferences in ASP and more generally in Computational Logic. In [4, 8, 10] the relationship of the approaches treated here with related work is also widely discussed.

The paper is organized as follows. In Section 2 we briefly introduce Answer Set Programming and $\mathcal{PDL}$, and in Section 3 we illustrate RASP. In Section 4 we propose how the two approaches could be usefully integrated, and in Section 5 we propose extensions to the language for expressing preferences. Finally, in Section 6 we conclude.

2 Background on ASP and $\mathcal{PDL}$

Answer Set Programming is the well-established logic programming paradigm adopting Datalog programs with default negation under the *answer set semantics*, briefly discussed below.

A program Π in this setting (cf. [17, 18]) is a collection of *rules* of the form

$$H \leftarrow \ L_1, \ldots, L_m, \ not \ L_{m+1}, \ldots, not \ L_{m+n}.$$

where H is an atom, $m \geq 0$ and $n \geq 0$, not is *default negation* and each L_i is either an atom or the *classical negation* (also called *strong negation*) of an atom (as defined in [18]). Atoms and their classical negation and literals $not \ L_i$ will be called in the following "ASP literals", to make a distinction w.r.t. the extensions that will be proposed.

The left-hand side and the right-hand side of the clause are called *head* and *body*, respectively. A rule with empty body is called a *fact*. A rule with empty head is a *constraint*, where a constraint of the form

$$\leftarrow L_1, ..., L_n.$$

states that ASP literals $L_1, \ldots, L_n$ cannot be simultaneously true.

Various extensions to the basic paradigm exist, that we do not consider here as they are not essential in the present context.

In the rest of the paper, whenever it is clear from the context, by "a (logic) program Π" we mean an answer set program (ASP program) Π, and we will implicitly refer to the "ground" version of Π. The ground version of Π is obtained by replacing in all possible ways the variables occurring in Π with the constants occurring in Π itself, and is thus composed of ground atoms, i.e., atoms which contain no variables.

The answer sets semantics [17, 18] considers logic programs as sets of inference rules (more precisely, default inference rules). Alternatively, one can see a program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program.

Consider the simple program $\{q \leftarrow not\, p.\ \ p \leftarrow not\, q.\}$. For instance, the first rule is read as "assuming that p is false, we can *conclude* that q is true." This program has two answer sets. In the first one, q is true while p is false; in the second one, p is true while q is false.

Given a subset M of $\mathbf{B}_\Pi$ that does not contain both A and $\neg A$ for any atom A, M is an answer set of Π if it coincides with the least model of the reduct P^M of P with respect to M. This reduct is obtained by deleting from Π all rules containing a condition $not\, a$, for some a in M, and by deleting all negative conditions from the other rules. Answer sets are minimal supported models. Referring to the original terminology of [17], answer sets are sometimes called *stable models*. As usual, a set of atoms S satisfies a rule if S satisfies its head H, i.e., $H \in S$, whenever it satisfies the body, i.e., all positive conditions are in S and none of the negative conditions is in S.

Unlike other semantics, a program may have several answer sets, or may have no answer set, because conclusions are included in an answer set only if they can be justified. The following program has no answer set: $\{a \leftarrow not\, b.\ b \leftarrow not\, c.\ c \leftarrow not\, a.\}$.

The reason is that in every minimal model of this program there is a true atom that depends (in the program) on the negation of another true atom, which is strictly forbidden in this semantics, where every answer set can be considered as a self-consistent and self-supporting set of consequences of given program. A program with no answer sets is called *inconsistent*. In the ASP paradigm, each answer set is seen as a solution of given problem, encoded as an ASP program. To find these solutions, an ASP-solver is used. Several solvers have became available, see [23], and can be freely downloaded by potential users.

The expressive power of ASP and its computational complexity have been extensively investigated. The interested reader can refer, for instance, to [12]. The reader can also see [2] and [15], among others, for a presentation of ASP as a tool for declarative problem-solving. For the applications of ASP, the reader can refer for instance to [2, 1, 19, 24, 16].

$\mathcal{PDL}$ is a language aimed at describing complex preferences among answer sets. The approach has been proposed in [4] and further generalized in [7]. The basic ingredients of $\mathcal{PDL}$, as introduced in [4, Def. 5], are the so-called *preference rules*. A preference rule has the form

$$h_1 > \cdots > h_k \leftarrow \ L_1, \ldots, L_m, \ not \ L_{m+1}, \ldots, not \ L_{m+n}. \tag{1}$$

where each h_i is of the form C_i:p_i and C_i is a *Boolean combination*, that is, a formula built of atoms by means of disjunction, conjunction, strong and default negation, while p_i is an integer number, intended to be the "weight" or the "penalty" associated to C_i, i.e., a measure of the degree of (dis)satisfaction in the choice of that option. A preference rule thus establishes a ranking of the answer sets, or, more precisely, a preorder between answer sets that satisfy the body of the rule.

Preference rules constitute the sub-language $\mathcal{PDL}^p$ of $\mathcal{PDL}$ made of the simplest $\mathcal{PDL}$ expressions. In turn, complex $\mathcal{PDL}$ expressions can be built out of $\mathcal{PDL}^p$ expressions by means of operators that specify how to impose and order upon answer sets of given programs based upon the preorders expressed as base step by the preference rules (then, nested $\mathcal{PDL}$ expressions are also allowed). Among the $\mathcal{PDL}$ operators there are $Pareto$ ordering, $lexicographic$ ordering and others. More specifically, we recall the following definition from [4]:

Definition 1. *[Def. 7 of [4]] $\mathcal{PDL}^p$ and $\mathcal{PDL}$ expressions are inductively defined as follows:*

1. *if r is a preference rule, then $r \in \mathcal{PDL}^p$,*
2. *if $e_1, \ldots, e_k$ are expressions in $\mathcal{PDL}^p$, then $(psum\ e_1, \ldots, e_k) \in \mathcal{PDL}^p$,*
3. *if $r \in \mathcal{PDL}^p$ then $r \in \mathcal{PDL}$,*
4. *if $e_1, \ldots, e_k$ are expressions in $\mathcal{PDL}^p$, then $(incr\ e_1, \ldots, e_k)$, $(rincr\ e_1, \ldots, e_k)$, $(card\ e_1, \ldots, e_k)$, and $(rcard\ e_1, \ldots, e_k)$ are in $\mathcal{PDL}$,*
5. *if $e_1, \ldots, e_k$ are in $\mathcal{PDL}$, then $(pareto\ e_1, \ldots, e_k)$ and $(lex\ e_1, \ldots, e_k)$ are in $\mathcal{PDL}$.*

As said before, $\mathcal{PDL}$ expressions allow one to specify a rank among answer sets according to several criteria (e.g., Pareto ordering). Optimal answer sets w.r.t. such a ranking, are said to be *solutions* of the *answer set optimization problem* as defined below.

Definition 2. *[Def. 4 of [4]] An answer set optimization problem (AOP) is a pair* $O = (P, prex)$ *where* P *is a program and* $prex$ *a* $\mathcal{PDL}$ *expression.*

A solution of O *is an answer set of* P *which is optimal according to the preorder represented by* $prex$*.*

Hence, a solution of an answer set optimization problem is a non-dominated answer set, that is, an answer set such that no strictly better answer set exists. (The reader is referred to [4] for a more detailed treatment of $\mathcal{PDL}$.)

The approach to "qualitative preferences" based on $\mathcal{PDL}$ is particularly interesting because several existing preference handling methods turn out to be special cases of it, where it is now well-known that many AI problems have natural formulations as optimization problems. An advantage of the approach is that $\mathcal{PDL}$ expressions can be compiled into logic programs that can be used as tester programs in a methodology for finding optimal answer sets. Moreover, as a further demonstration of its applicability, the approach has been generalized from ASP to many other optimization contexts in [7].

3 Previous Work on Preferences Revisited

In our work on RASP (ASP with Resources and Preferences) [9, 8, 11] we have proposed an approach to expressing preferences in ASP programs, and then [10] also in Weight Constraints (which are a well-known useful construct of ASP [21, 22]).

In particular, we allow any occurrence of a positive literal in ASP rules to be replaced by a *p-list*, defined below. p-lists make sense in the head of a rule, to state what one *prefers* to derive (equivalently, which atom one prefers to be included in an answer set). However, p-lists make sense in the body of rules as well, to specify which conditions one would prefer to exploit for reaching a conclusion. As seen below, preferences in the body are particularly interesting when they are conditional, that is, they should be applied only when some precondition is satisfied.

Definition 3. *Let* $s_1, \ldots, s_k$ $(k > 0)$ *be either distinct constants or distinct atoms. Then,*

- *a* basic preference-list *(*p-list*, for short) is an expression of the form* $s_1 > \cdots > s_k$*. Each component* s_i $(i \leq k)$ *has* degree of preference i *in the p-list.*
- *a* conditional p-list *(*cp-list*, for short) is an expression of the following form:*

$$(r \ \ pref\_when \ \ L_1, \ldots, L_n),$$

where $r = s_1 > \cdots > s_k$ *is a p-list and* $L_1, \ldots, L_n$ *are ASP literals.*

Intuitively, a cp-list $(r \ \ pref\_when \ \ L_1, \ldots, L_n)$ specifies that if all $L_1, \ldots, L_n$ are satisfied, then the choice among the s_is occurring in r is ruled by the preference expressed through r. Otherwise, if any of the L_i is not satisfied, then no preference is expressed.

In general, there might be cases in which useful (conditional) preferences are not expressible as a linear order on a set of alternatives. This may originate from lack of adequate knowledge or expertise in modeling a piece of knowledge, or from incapability to

completely describe total comparative relations, in presence of uncertainty. Moreover, preferences might depend on specific contextual conditions that are not foreseeable in advance.

P-sets are a generalization of p-lists that allows one to use any binary relation (not necessarily a partial order) in expressing (collections of alternative) p-lists.

Definition 4. *Let $q_1, \ldots, q_k$ be $k > 0$ atoms and let $pred$ be a binary predicate. A* p-set *is of the form*

$$\{q_1, \ldots, q_k \mid pred\}.$$

Predicate $pred$ is supposed to be defined elsewhere in the program where the p-set occurs. Considering any given model of the program, such a binary predicate is intended to (possibly) have pairs of the form $\langle q_i, q_j \rangle$ as elements of its extension. This allows the programmer to describe preferences among the q_is, that are computed contextually to any specific answer set. The intuitive semantics of a p-set can be grasped by considering a particular extension for the predicate $pred$ (namely, a set of pairs $\langle a, b \rangle$ assumed to be true in a certain situation, that is, in a certain model of the program). Let X be the set of atoms $\{q_1, \ldots, q_k\}$. Consider the binary relation $R \subseteq X \times X$ obtained by restricting to X the extension of $pred$. R is interpreted as a preference relation over X: namely, for any $q_i, q_j \in X$ the fact that $\langle q_i, q_j \rangle \in R$ models a preference of q_i over q_j.

In general, there might exist many total orders on the set $\{q_1, \ldots, q_k\}$ that are compatible (i.e., do not contradict) such a generic relation R. Any of these total orders describes one different, alternative, p-list. Hence, simple p-lists, as introduced in Definition 3, are particular cases of p-sets, obtained when R describes a total order.

As mentioned, R does not need to be a partial order, e.g., for instance, it may imply cycles. In such cases, those resources that belong to the same cycle in R are considered equally preferable. On the other hand, R might be a partial relation. So, there might exist elements of X that are incomparable. In this paper, for the sake of simplicity we assume that R defines a partial order.

Compound preferences are allowed in RASP. For lack of space in this paper we do not consider compound preferences and thus we do not report the full definition, however p-lists (and cp-lists) are generalized so that preferences can be expressed among sets.

A RASP rule (for short r-rule) has the following form, whereas a RASP program is a set of RASP rules:

Definition 5. *Let an* a-literal *be either an atom or a p-list, a cp-list, or a p-set, and let an* r-literal *be either an ASP literal or an a-literal. An* r-rule γ *has the form*

$$H \leftarrow B_1, \ldots, B_m. \tag{2}$$

where $B_1, \ldots, B_m$ are r-literals, H is either an atom or an a-literal (a generalization where H is a set of a-literals is easily feasible, though not treated here).

Let a *proper RASP rule* be a RASP rule involving a-literals (otherwise a RASP rule is identical to an ASP rule).

Definition 6. *Let Π^{Pref} be the subprogram of RASP program Π composed of all the proper RASP rules occurring in Π.*

The following example, related to preparation of desserts, should demonstrate the expressivity of RASP. In preparing the dessert, one might employ, e.g., either skim-milk or whole milk. The *cp-list*

$$skimmilk > wholemilk\ pref\_when\ diet$$

states that, if on a diet, the former is preferred. Then, to spice the dessert, one would choose, by the *p-set*

$$\{chocolate, nuts, coconut \mid less\_caloric\},$$

the least caloric among chocolate, nuts, and coconut.

Previous examples show instances of "extrinsic preferences", i.e., preferences which come with some kind of "reason", or "justification" [20]. Notice that, in RASP, extrinsic preferences may change even non-monotonically as the knowledge base evolves in time, as the justification can be any conjunction of literals.

Below is a RASP program fragment representing dessert preparation with the preferences outlined above. Notice that the preference among the ingredients chocolate, nuts, and coconut, occurring in the p-set has to be determined by referring to the extension that the predicate $less\_caloric$ has in the particular model at hand. The first rule states that if one has eggs and sugar, milk (preferably skim milk if one is on a diet) and one of chocolate, nuts and coconut (choosing the less caloric) one can prepare either ice-cream or zabaglione, but preferably one would prepare ice-cream (consider that some constraint stated elsewhere in the program might forbid any of the two).

$$\begin{aligned}
&icecream > zabaglione \leftarrow egg,\ sugar,\\
&\qquad\qquad\qquad\qquad (skimmilk > wholemilk\ pref_when\ diet),\\
&\qquad\qquad\qquad\qquad \{chocolate, nuts, coconut \mid less_caloric\}.\\
&less_caloric(X,Y) \leftarrow calory(X,A), calory(Y,B), A < B.\\
&calory(nuts, 2).\\
&calory(coconut, 3).\\
&calory(X,Y) \leftarrow ...
\end{aligned}$$

In full RASP, resource management is provided, so that one can, in the above example, associate *quantities* to the various produced and consumed items. We do not consider this feature here, so in the rest of the paper we will implicitly consider RASP programs without quantities. In summary, in the rest of the paper we consider RASP programs without quantities and without compound preferences.

We may notice that RASP rules can be seen (within the above restrictions) as a particular case (on the one hand) and a generalization (on the other hand) of preference rules as introduced in Definition 1.

RASP rules are a special case of preference rules in that arbitrary formulas are not allowed in the elements of p-lists, and the (implicit) weight is always 1. They are a generalization, in that various forms of conditional preferences are allowed, also in the body of rules. We showed in [8] how RASP rules can be restated in terms of rules with preferences expressed only in the head (we considered precisely the LPOD approach [3,

5]). The re-writing however was not trivial and involved transforming a single RASP rule into a set of rules. In the following sections we intend first to show how RASP fits into the $\mathcal{PDL}$ framework. Then, elaborating on our past work and on principles discussed in [25, 26, 13, 20], we further improve the forms of preferences expressible in preference rules.

4 $\mathcal{PDL}$ and RASP: Integration

We may notice that a $\mathcal{PDL}$ program can be added on top of an ASP program, and as said above it produces an ordering of the answer sets in terms of given preferences. The advantage is that the same ASP program may be transformed, by adding different $\mathcal{PDL}$ parts, into different optimization problems. The disadvantage is that a programmer cannot use complex preferences *within* the ASP program. RASP is thus complementary to $\mathcal{PDL}$: in fact, in an integration part of the preferences (in the form of RASP rules) might be expressed within the program, and part in the external (one may say "upper") $\mathcal{PDL}$ layer.

The integration with RASP would then bring to $\mathcal{PDL}$ the advantage of more general and flexible preference rules definable directly in the ASP program. However, RASP would profit from being integrated into $\mathcal{PDL}$ as the method of eliciting partial orders from preference rules and combining them (based upon other $\mathcal{PDL}$ rules) allows for further significant extensions, like the one that we propose below. In the integration, semantics of RASP programs can be defined differently from [9, 8, 11]. In fact, a RASP program Π can be transformed into an ASP program Π' where each r-rule (as defined in Definition 2) is replaced by a set of rules as defined below.

Definition 7. *An ASP rule γ' derived from RASP rule γ*

$$H \leftarrow B_1, \ldots, B_m.$$

is of the form

$$A \leftarrow L_1, \ldots, L_m.$$

where: $A = H$ if H is a ASP atom, and for $i \leq m$ $L_i = B_i$ if B_i is a ASP literal; whenever H or, respectively, B_i (for $i \leq m$) is a p-list $r = s_1 > \cdots > s_k$ or a cp-list $(r \ \ pref\_when \ \ L_1, \ldots, L_n)$, then $A = s_j$, or, respectively, $L_i = s_j$, $j \leq k$; if B_i is a p-set $\{q_1, \ldots, q_k \mid pred\}$, then $L_i = q_r$, $r \leq k$.

Hence, derived rules are obtained by non-deterministically selecting one of the options among the a-literals. By relying on the usual notion of satisfaction of (ground) ASP rules, we can state the following notion of satisfaction for derived rules.

Definition 8. *Given a set of atoms S and a RASP rule γ, S satisfies γ if and only if S satisfies some γ' derived from γ. In this case we write $S \models \gamma$.*

The next definition will be useful in what follows.

Definition 9. *Given RASP program Π, the program Π' composed of all the derived rules obtainable from rules of Π, is called the* ASP program derived from Π.

We now extend $\mathcal{PDL}$ by including proper RASP rules in $\mathcal{PDL}$ expressions. (Recall that $\mathcal{PDL}$ expressions are built out of basic $\mathcal{PDL}^p$ expressions by means of preference operators.) The following definition introduce a further case to extend Definition 1:

Definition 10 (Extension of Def. 1). *$\mathcal{PDL}^p$ and $\mathcal{PDL}$ expressions are inductively defined as in Definition 1, by considering also the following case:*

1′. *If r is a proper RASP rule then $r \in \mathcal{PDL}^p$.*

The new semantics for RASP programs, also corresponding to an extension to $\mathcal{PDL}$ that includes proper RASP rules as preference rules, can be defined as follows (cf., Definition 2).

Definition 11. *Given RASP program Π, an extended answer set optimization problem (EAOP) is a pair $O^E = (\Pi', prex)$ where Π' is the ASP program derived from Π and $prex$ is a $\mathcal{PDL}$ expression involving among the preference rules all rules of Π^{Pref}.*

What remains to be done is to extend the semantics of $\mathcal{PDL}$ expressions defined in [4, pp. 4-5] so as to accommodate RASP rules. This semantics introduces a preorder on answer sets (i.e., a reflexive and transitive relation) that specifies which answer set is preferred over which other. Below we enrich the definition of [4]. Notice that given expression $prex$, $pen(S, prex)$ is the penalty associated to the expression $prex$ w.r.t. answer set S, and $Ord(prex)$ is the preorder associated with $prex$. The objective of the definition is to state in which cases we have $(S_1, S_2) \in Ord(prex)$ based on $pen(S_1, prex)$ and $pen(S_2, prex)$. By abuse of notation, we consider a-literals (which are not in themselves $\mathcal{PDL}^p$ expressions) as a base case of $Ord(prex)$.

Definition 12. *Given RASP program Π and EAOP $(\Pi', prex)$, and given the answer sets of Π' S, S_1, S_2 (clearly, if Π' has none or just one answer set the EAOP is trivially solved) then, in addition to what defined in [4], we define the following.*

- *If $prex$ is a p-list of the form $s_1 > \cdots > s_k$ then $pen(S, prex) = j$ where $j = min\{i | S \models s_i\}$. $pen(S, prex) = \infty$ otherwise.*
- *If $prex$ is a cp-list of the form $s_1 > \cdots > s_k$ $pref\_when$ $L_1, \ldots, L_n$ then $pen(S, prex) = j$ where $j = min\{i | S \models s_i\}$ if $S \models L_1, \ldots, L_n$. $pen(S, prex) = \infty$ otherwise.*
- *If $prex$ is a p-set of the form $\{q_1, \ldots, q_k \mid pred\}$ then $pen(S, prex) = j$ where $j = min\{i | \forall j \leq k, j \neq i, S \models pred(q_i, q_j)\}$. $pen(S, prex) = \infty$ otherwise.*
- *Given a RASP rule r, we have $(S_1, S_2) \in Ord(r)$ iff for each a-literal s occurring in r, $pen(S_1, s) \leq pen(S_2, s)$.*

Notice that in the above definition we apply a lexicographic ordering within the preorders induced by the a-literals occurring in a RASP rule. Other choices would be of course possible. We might state more generally that $(S_1, S_2) \in Ord(r)$ iff $(S_1, S_2) \in Ord(Op\ a_1, \ldots, a_n)$ where Op is one of the ordering operators provided by $\mathcal{PDL}$, applied to the a-literals $a_1, \ldots, a_n$ occurring in r (i.e., to the preorders induced by them).

It is easy to extend complexity results reported in [4] to our extended setting. I.e., finding solutions to AOP's is a difficult problem but our extensions, though providing features that may help programmers to declaratively encode the situation at hand, do not add further complexity.

Theorem 1. *Let $O = (\Pi', prex)$ be an extended answer set optimization problem and S an answer set of Π'. Deciding whether S is a solution of O is coNP-complete.*

Theorem 2. *Let $O = (\Pi', prex)$ be an extended answer set optimization problem and A be either an atom or the "classical negation" of an atom. Deciding whether there is a solution S of O such that $A \in S$ is Σ_2^P-complete.*

5 Extensions

As discussed in [25, 26, 13, 20], one may prefer one option over others for reasons of various kinds: from general principles to specific facts. Several reasons or criteria may justify one single preference, where however criteria are often ordered according to some kind of priority. By adapting the example proposed in [20], if one wants to buy a house (s)he may consider several aspects, say the price, the neighborhood where the house is situated, and the quality of the building. These aspects may be ordered according to their importance, i.e., they are given a certain priority. In [20], a method is proposed to elicit a strict order among options from a given *priority sequence* of the form, say,

$$C_1(X) \gg C_2(X) \gg \ldots \gg C_n(X)$$

i.e., in the above example,

$$good\_neighborhood(X) \gg reasonable\_price(X) \gg good\_quality(X).$$

A way of combining preferences is also proposed in [20, Def. 3.2]. This method is reported below, and adopted in the present setting.[3] Given a priority sequence of length n, strict preference of x over y between two objects x and y, $Pref(x, y)$, is defined, according to [20], as follows:

$$Pref_1(x, y) ::= C_1(x) \wedge \neg C_1(y)$$
$$Pref_{k+1}(x, y) ::= Pref_k(x, y) \vee (Eq_k(x, y) \wedge C_{k+1}(x) \wedge \neg C_{k+1}(y)), k < n$$
$$Pref(x, y) ::= Pref_n(x, y)$$

where the auxiliary binary predicate $Eq_k(x, y)$ is the conjunction of equivalences

$$(C_1(x) \leftrightarrow C_1(y)) \wedge \cdots \wedge (C_k(x) \leftrightarrow C_k(y)).$$

In simple words, x is preferred over y whenever there is at least one of the C_is that x enjoys while y does not. Consequently, if both x and y enjoy the C_is in the same way, none of them is preferred over the other.

Referring to the example, one may find it very important to live in a good neighborhood, of some importance to keep the cost reasonable and less important (but relevant anyway) to buy a house in a building of good quality. For instance, if we have two houses h_1 and h_2 which are both in a good neighborhood but none of them has a reasonable price, then the one of good quality will be preferred. If both are of good quality,

[3] Refining both the form and the meaning of priority sequences in the direction of more flexibility can be a subject of future work.

they will be equally preferred. If both are in a good neighborhood, and the first one has reasonable price while the other one does not, then the former will be preferred, whatever the quality of both. RASP, in the context of the integration with $\mathcal{PDL}$, might be extended so as to allow rules such as the following:

$$buy(X) \leftarrow house(X), \\ good\_neighborhood \gg reasonable\_price \gg good\_quality.$$

In this paper we limit the conclusion of such a rule to be an atom (as defined in ASP), while in perspective it might be an a-literal. The priority sequence is defined over predicates, each of which is applicable on the variables occurring in the head (thus, if the predicate in the head is n-ary, so are the predicates in each priority list). For the sake of simplicity, below we restrict ourselves to unary predicates. The atom in the head, in the ground version of the rule, becomes a disjunction, one for each option that can be possibly chosen. In the example, if $house(X)$ returns, e.g., h_1, h_2 and h_3, the ground version of the rule will be (where, as customary in ASP, ';' indicates disjunction):

$$buy(h_1)\,;\, buy(h_2)\,;\, buy(h_3) \leftarrow \\ good\_neighborhood \gg reasonable\_price \gg good\_quality.$$

We extend the definition of RASP rules as follows. As before, in the definitions we refer to ground atoms and rules.

Definition 13. *A priority list is an expression of the form*

$$p_1 \gg \ldots \gg p_n$$

where each of the p_i's is a predicate.

Definition 14. *A RASP priority rule ρ is of the form:*

$$H \leftarrow B_1, \ldots, B_m.$$

where $B_1, \ldots, B_m$ are either r-literals or priority lists, and H is a disjunction of atoms of the form $h(c_1); \ldots; h(c_r)$, $r \geq 0$, where the c_is are constants.

As far as non-ground rules are considered, a ground priority rule can be obtained from a non-ground rule of the form $h(X) \leftarrow d(X), B_1, \ldots, B_m$ where $d(X)$ provides the values for instantiating X (d might be for instance a domain predicate.[4])

We define a notion of *degree of satisfaction* of a priority list in the context of a set of atoms by rephrasing the above-mentioned definition introduced in [20].

Definition 15. *A priority list $l = p_1 \gg \ldots \gg p_n$ is satisfied with degree k w.r.t. a disjunction of atoms $d = h(c_1); \ldots; h(c_r)$ and a set of atoms S (we write $sat(S, l, d) = k$) iff $S \models h(c_i)$, $i \leq r$, where $S \models p_1(c_i) \wedge \ldots \wedge S \models p_k(c_i)$ and $\nexists c_j$, $j \neq i$ such that $S \models h(c_j)$ and $S \models p_1(c_j) \wedge \ldots \wedge S \models p_t(c_j)$ with $t > k$.*

[4] The subset of given program defining *domain predicates* consists of *domain rules*, syntactically restricted so as to admit a unique answer set that should be relatively efficiently computable. All the other rules in the program are required by most answer set solvers to be *domain-restricted* in the sense that every variable in a rule must appear in a domain predicate which occurs positively in the body of the rule.

To accommodate priority lists, we further extend Definition 12 as follows.

Definition 16.

– *Given a RASP priority rule r with head d, we have $(S_1, S_2) \in Ord(r)$ iff for each a-literal s occurring in r, $pen(S_1, s) \leq pen(S_1, s)$ and for each priority list l occurring in r, $sat(S_1, l, d) \geq sat(S_2, l, d)$.*

6 Concluding Remarks

In this paper we coped with expressing preferences in Answer Set programs. In particular, we have proposed an integration of RASP (that has been defined in previous work) and $\mathcal{PDL}$, previously proposed by G. Brewka. We argued that both approaches may profit from the integration in that RASP fits into the more general semantic $\mathcal{PDL}$ framework, and RASP enriches $\mathcal{PDL}$ with more expressive preference rules. To show this, we have further extended the formalism by means of priority rules that express a kind of preference that could not be directly expressed before, without affecting complexity. Precisely, deciding whether an answer set S is a solution of an answer set optimization problem is coNP-complete, and deciding whether there is a solution implying given literal l is $\Sigma_P^2 - complete$. Immediate future work concerns the implementation of the integrated approach. Other future work can be related to further enriching the language for expressing preferences.

References

[1] C. Anger, T. Schaub, and M. Truszczyński. ASPARAGUS – The Dagstuhl Initiative. *ALP Newsletter*, 17(3), 2004. See `http://asparagus.cs.uni-potsdam.de`.

[2] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[3] G. Brewka. Logic programming with ordered disjunction. In *Proc. of AAAI-02, Edmonton, Canada*, 2002.

[4] G. Brewka. Complex preferences for answer set optimization. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the Ninth International Conference (KR2004), June 2-5, 2004*, pages 213–223, 2004.

[5] G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):335–357, 2004.

[6] G. Brewka, I. Niemelä, and M. Truszczyński. Preferences and nonmonotonic reasoning. *AI Magazine*, 29(4), 2008.

[7] G. Brewka, M. Truszczynski, and S. Woltran. Representing preferences among sets. In M. Fox and D. Poole, editors, *Proc. of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, pages 273–278. AAAI Press, 2010.

[8] S. Costantini and A. Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *J. of Algorithms in Cognition, Informatics and Logic*, 64(1):3–15, 2009.

[9] S. Costantini and A. Formisano. Answer set programming with resources. *J. of Logic and Computation*, 20(2):533–571, 2010.

[10] S. Costantini and A. Formisano. Weight constraints with preferences in ASP. In *Proc. of LPNMR'11*, number 6645 in LNCS, pages 229–235. Springer, 2011.

[11] S. Costantini, A. Formisano, and D. Petturiti. Extending and implementing RASP. *Fundamenta Informaticae*, 105(1-2):1–33, 2010.

[12] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[13] D. de Jongh and F. Liu. Preference, priorities and belief. In T. Grüne-Yanoff and S. O. Hansson, editors, *Preference Change*, Theory and Decision Library, pages 85–108. 2009.

[14] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(12):308–334, 2004.

[15] A. Dovier, A. Formisano, and E. Pontelli. An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. *J. of Experimental and Theoretical Artificial Intelligence*, 21(2):79–121, 2009.

[16] M. Gelfond. Answer sets. In *Handbook of Knowledge Representation*, chapter 7. Elsevier, 2007.

[17] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of 5th ILPS conference*, pages 1070–1080, 1988.

[18] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[19] N. Leone. Logic programming and nonmonotonic reasoning: From theory to systems and applications. In C. Baral, G. Brewka, and J. Schlipf, editors, *Proc. of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *LNCS*, page 1. Springer, 2007.

[20] F. Liu. Von Wright "The Logic of Preference" revisited. *Synthese*, 175(1):69–88, 2009.

[21] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proc. of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, number 1730 in LNCS, pages 317–331. Springer, 1999.

[22] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[23] Solvers URLs. Clasp: `http://potassco.sourceforge.net`;
Cmodels: `http://www.cs.utexas.edu/users/tag/cmodels`;
DLV: `http://www.dbai.tuwien.ac.at/proj/dlv`;
Smodels: `http://www.tcs.hut.fi/Software/smodels`.

[24] M. Truszczyński. Logic programming for knowledge representation. In V. Dahl and I. Niemelä, editors, *Proc. of the 23rd International Conference on Logic Programming (ICLP'07)*, volume 4670 of *LNCS*, pages 76–88. Springer, 2007.

[25] J. van Benthem, P. Girard, and O. Roy. Everything else being equal: A modal logic for ceteris paribus preferences. *J. Philos. Logic*, 38:83–125, 2009.

[26] J. van Benthem and F. Liu. Dynamic logic of preference upgrade. *J. of Applied Non-Classical Logics*, 17(2):157–182, 2007.