Augmenting Weight Constraints with Complex Preferences

Stefania Costantini[1] and Andrea Formisano[2]

[1] Università di L'Aquila, Italy stefania.costantini@univaq.it
[2] Università di Perugia, Italy formis@dipmat.unipg.it

Abstract. Preference-based reasoning is a form of commonsense reasoning that makes many problems easier to express and sometimes more likely to have a solution. In this paper, we present an approach to introducing preferences in the weight constraint construct, which is a very useful programming construct widely adopted in Answer Set Programming (ASP). We show, mainly by means of examples, the usefulness of the proposed extension, and we outline how to accordingly extend the ASP semantics.

1 Introduction

Preference is deeply related to a subject's personal view of the world, and it drives the actions that we take in it. Preference is subjective, but in practice we influence each others preferences all the time by making evaluative statements, uttering requests, commands, and statements of facts that have an impact either on the possibility of performing actions or on the objectives that one intends to pursue. Preference has been studied in many disciplines, especially in philosophy and social sciences, but also in psychology, economics (especially when the need arises of formalizing some form of rational mental process that human beings activate during decision making, possibly in presence of uncertain knowledge), and, last but not least, in logic.

The studies of the processes that support the construction or the elicitation of preferences have historically deep roots. In logic, [21] initiated a line of research that was subsequently systematized in [42] which is usually taken to be the seminal work in preference logic. This line of research continues nowadays: the works of [40] and [29], for instance, develop new modal preference logics that improve over [21] in several directions. For other studies about preference the reader may refer, e.g. among many, to [22, 27, 39, 7, 18] and to the references therein.

Many forms of commonsense reasoning rely more or less explicitly upon some form of preferences. In fact, in contrast to expert knowledge, which is usually explicit, most commonsense knowledge is implicit and one of the issues in knowledge representation is making this knowledge explicit. Thus, being able to express and use preferences in a formal system constitutes a significant step in this direction. Intuitively, it simulates a skill that every person takes for granted. From the point of view of knowledge representation, many problems are more naturally represented by flexible rather than by hard descriptions. Practically, many problems would not even be solvable if one would stick firmly on all requirements.

Not surprisingly, several formalisms and approaches to deal with preferences and uncertainty have been proposed in Artificial Intelligence (AI), such as CP-nets and preference logics (see [6, 12, 22, 7, 18, 25, 5], to mention a few). Explicit preferences for actions in rules have been studied in AI. As a notable example we mention the SOAR architecture [37].

Preferences handling in computational logic has been extensively studied too. The reader may refer, e.g., to [17, 10] for recent overviews and discussion of many existing approaches to preferences. Many of these approaches have been defined in the context of Answer Set Programming (ASP for short) [20, 28, 30], a relatively recent paradigm of logic programming that has been successfully applied to many forms of knowledge representation and commonsense reasoning (cf., among others, [4, 26, 38, 19] and the references therein).

In fact, ASP has the peculiarity that an ASP program may have none, one or several answer sets. These answer sets can be interpreted in various possible ways. If the program formalizes a search problem, e.g., a colorability problem or a path finding problem for graphs, then the answer sets represents the possible solutions to the problem, namely, in the examples, the possible colorings or the existing paths for given graph. In knowledge representation, an ASP program may represent a formal definition of the known features of a situation/world of interest. In this case, the answer sets represent the possible consistent states of this world, that can be several whenever the formalization involves some kind of uncertainty. Also, and ASP program can be seen as the formalization of the knowledge and beliefs of a rational agent about a situation/world, and the answer sets represent the possible belief states of such an agent, that can be several if either uncertainty or alternative possible choices are involved in the description [11]. Such an agent can exploit an ASP module for several purposes, such as answering questions, building plans, explaining observations, making choices, etc. For knowledge representation tasks, preferences may play a relevant role. For instance, the motivating example discussed in [8] concerns scheduling problems to be solved according to both preferences and priorities.

In [10] it is observed that "...commonsense reasoning [is] based on our inherent preference to assume that things, given what we know, are normal or as expected. This assumption allows us to form preferred belief sets, base our reasoning exclusively upon them, and ignore all other belief sets that are consistent with our incomplete knowledge but represent situations that are abnormal or rare". More generally, expressing preferences can be viewed as an indirect way of expressing preferences on belief sets in terms of the elements these belief sets contain. One may also want to represent conditional preferences so that beliefs can be accepted based on other beliefs already accepted or rejected. Preferences in ASP may be employed to select the "most preferred" answer sets, considering that this however implies an increase in computational complexity. Or also one may introduce programming constructs that influence the construction of answer sets by including the best preferred available conclusions. This is the stance that we take in this paper, where of course the two points of view can be seen as complementary rather than exclusive.

In recent work [15, 14, 16] we have proposed RASP, an extension to ASP where complex forms of preferences can be flexibly expressed. In that work we have consid-

ered plain ASP. In this paper, we intend to introduce preferences into ASP extended with weight constraints [31, 34]. A weight constraint allows one to include in the answer sets of given program an arbitrary (bounded, but variable) number of the literals indicated in the constraint itself, according to weights. Weight constraints have proved to be a very useful programming tool in many applications such as planning and configuration, and they are nowadays adopted (in some form) by most ASP inference engines (usually called "ASP solvers" [2]).

In this paper, we propose to enrich weight constraints by means of RASP-like preferences. For simplicity we consider the particular case of cardinality constraints, which are however very widely used, considering that the extension to general weight constraints is easily feasible. The advantage of introducing RASP-like preferences rather than considering one of the competing approaches is their *locality*. In fact, in RASP one may express preferences which are local to a single rule or even to a single literal. Contrasting preferences can be freely expressed in different contexts, as in our view preferences may vary with changing circumstances. Instead, most of the various forms of preferences that have been introduced in ASP (see [17] and [41] for a comparison of these approaches) are based on establishing priorities/preferences among rules or preferences among atoms which are anyway globally valid in given program (for a discussion and a comparison of RASP with different approaches to preferences in ASP and logic programming the reader may refer to [14]). A weight constraint represents a local context where RASP-like preferences find a natural application.

The rest of the paper is organized as follows. In Section 2 we briefly illustrate weight constraints by means of examples. In Section 3 we summarize the form of preferences as introduced in RASP and their motivations in relation to some relevant literature. In Section 4 we illustrate our proposal of application of these preferences, in ASP, within weight constraints. In Section 5 we show how the semantics for ASP with weight constraints proposed in [31, 34] can be extended so as to encompass preferences, without affecting complexity: this is in our view one of the advantages of our approach. Finally, we conclude in Section 6.

2 Cardinality Constraints in ASP

Cardinality constraints are a special case of weight constraints (both are discussed in [31, 34], where their semantics is also presented). Though the computational complexity of ASP with weight constraints remains the same, the modeling power of the extended language is higher, as proved by the wide application of this construct (see, e.g., [35]). A *Weight Constraint* is of the form:

$$l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathit{not}\ b_1 = w_{b_1}, \ldots, \mathit{not}\ b_m = w_{b_m}\} \leq u$$

where the a_is and b_is are atoms. Each literal in a constraint has an associated weight, i.e., the weight of each a_i is w_{a_i} and the weight of each *not* b_j is w_{b_j} The numbers l and u give, respectively, the lower and upper bounds of the constraint. The weights and bounds are real numbers. Intuitively, a weight constraint is satisfied by a set of atoms S if the sum of weights of those literals occurring in the constraint that are satisfied by S is between l and u. Either of the bounds can also be omitted, in which case it is taken

to be not relevant. The intended meaning is that an answer set is allowed to include a subset of the atoms occurring in the constraint so that the corresponding sum of weights results in the interval $[l, u]$, where the weight of a negative literal *not* b_j is counted only if b_j is not in the answer set.

Plain literals can be seen as a special case of weight constraint, thus a program rule (called weight constraint rule) will have the form

$$C_0 \leftarrow C_1, \ldots, C_n.$$

where the C_is are weight constraints. As mentioned, the extra-expressivity is obtained without increasing complexity of ASP. In fact, as proved in [31, 34], deciding whether a program composed of a set of ground weight constraint rules has an answer set is still NP-complete, and computing an answer set is still FNP-complete.

Weight constraints have become in time a very important and widely used programming tool in ASP, especially in the formulation where all weights are equal to one. Weight constraints in this special form are called *Cardinality Constraints*, for which the following shorthand form is provided:

$$l\,\{a_1, \ldots, a_n, \textit{not}\ b_1, \ldots, \textit{not}\ b_m\}\,u$$

In order to be able to compactly write down sets of literals for weight constraints, [31, 34] introduce a notion of a conditional literal of the form $l\ :\ d$ where l is a literal and the conditional part d is a domain predicate[3]. A conditional literal corresponds to the sequence of all the instances of the literal l obtained by making a substitution to $l\ :\ d$ such that for the resulting $l'\ :\ d'$, d' is in the unique answer set of the domain part of the program. This allows programmers to write, in domain-restricted programs, weight and cardinality constraints involving variables.

Below we illustrate this general form of cardinality constraints by means of an example (for a full and formal definition the reader should refer to the above-mentioned references). Assume that you wish to state that a meal is composed of at least two and at most three courses. This may be expressed by the following cardinality constraint.

$$2\{in\_menu(X, C) : course(C)\}3 \leftarrow meal(X).$$

Assume now that the background knowledge base is the following.

$$\begin{array}{ll} meal(lunch). & course(pasta). \\ meal(dinner). & course(meat). \\ & course(cake). \\ & course(fruit). \end{array}$$

The above constraint should be seen as a shorthand for what follows, where every possible value for variable C is listed (and thus the "domain predicate" $course$ becomes

[3] 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.

irrelevant), so as to make explicit which are the possible choices.

$$2\{in\_menu(X, pasta),\ in\_menu(X, meat),\\ in\_menu(X, cake),\ in\_menu(X, fruit)\}3 \leftarrow meal(X).$$

In turn, in the ground version of the program, which is the one that is processed by the solvers, we will have the two instances (simplified because of the truth of $meal(lunch)$ and $meal(dinner)$ in the knowledge base shown earlier):

$$2\{in\_menu(lunch, pasta), in\_menu(lunch, meat),\\ in\_menu(lunch, cake), in\_menu(lunch, fruit)\}3.$$

and

$$2\{in\_menu(dinner, pasta), in\_menu(dinner, meat),\\ in\_menu(dinner, cake), in\_menu(dinner, fruit)\}3.$$

Therefore, there will be different answer sets where one is the subset of another (which is not possible in traditional answer sets, which are selected among the models in classical sense) in that they include a different number of atoms.

Notice that the grounding of a given constraint may change according to the underlying domain. Assume in fact to have this knowledge base:

$meal(lunch).$	$course(pasta) \leftarrow$ *not* $coeliac.$
$meal(dinner).$	$course(meat).$
	$course(cake).$
$coeliac.$	$course(fruit).$

The fact $coeliac$ excludes $pasta$ form the domain, and thus the grounding becomes:

$$2\{in\_menu(lunch, meat), in\_menu(lunch, cake), in\_menu(lunch, fruit)\}3.$$
$$2\{in\_menu(dinner, meat), in\_menu(dinner, cake), in\_menu(dinner, fruit)\}3.$$

3 Preferences in RASP

RASP, defined in [15], is an extension to the ASP framework that allows for the specification of various kinds of non-trivial preferences. RASP preferences follow the quite intuitive principles first formalized in [42], and illustrated at length, e.g., in [40]. The first two principles state that any preference relation is asymmetric and transitive. For simplicity we stick to strict preferences, i.e., if in a certain context one prefers ϕ to ψ, then in the same context one cannot also prefer ψ to ϕ. An advancement of our approach over others is that preferences, as illustrated below, have a local flavor. I.e., a preference holds in the context of the rule where it is defined, where different (even contrasting) preferences can be expressed (and simultaneously hold) in different contexts. The third principle states that preferring ϕ to ψ means that a state of affairs where $\phi \wedge \neg\psi$ holds is preferred to a state of affairs where $\psi \wedge \neg\phi$ holds. The fourth principle states that if I prefer ψ to $(\phi \vee \zeta)$ then I will prefer ψ to ϕ and ψ to ζ. Finally, the last principle states that a change in the world might influence the preference order between two states of

affairs, but if all conditions stay constant in the world ("ceteris paribus"), then so does the preference order.

We propose an example in order to illustrate the approach, and then the formal definitions follow. The RASP program below defines a recipe for a dessert. The writing $icecream > zabaglione$ is called a *p-list* (preference list) and states that with the given ingredients one might obtain either ice-cream or zabaglione, but the former is preferred. This is, in the terminology of [42], an "intrinsic preference", i.e., a preference without a specific reason. In preparing the dessert, one might employ either skim-milk or whole milk. The *cp-list* $skimmilk > wholemilk\ pref\_when\ diet$ states that, if on a diet, the former is preferred. Finally, to spice the dessert, one would choose, by the *p-set* $\{chocolate, nuts, coconut \mid less\_caloric\}$, the less caloric among chocolate, nuts, and coconut. These are instead instances of "extrinsic preferences", i.e., preferences which come with some kind of "reason", or "justification". 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.

$$
\begin{array}{l}
icecream > zabaglione \leftarrow egg,\ sugar, \\
\qquad\qquad\qquad\qquad (skimmilk > wholemilk\ pref\_when\ diet), \\
\qquad\qquad\qquad\qquad \{chocolate, nuts, coconut \mid less\_caloric\}. \\
less\_caloric(X,Y) \leftarrow calory(X,A), calory(Y,B), A < B. \\
calory(nuts, 2). \\
calory(coconut, 3). \\
calory(X,Y) \leftarrow ...
\end{array}
$$

In full RASP, quantities for ingredients and products are allowed to be specified, and the peculiarity of RASP is that resources in the body of rules are actually *consumed* and thus are no longer available (unless there is a reminder) and instead resources in the head are *produced*. However, in this paper we neglect the aspects of RASP related to resources in order to concentrate on preferences. The constructs seen in the example are defined below.

Definition 1. *Let $s_1, \ldots, s_k$ be $k > 0$ either distinct constants or distinct atoms. Then a* basic preference-list *(*p-list*, for short) is a writing of the form $s_1 > \cdots > s_k$. Each component s_i has* degree of preference i *in the p-list.*

It is often the case that a preference should be applied only when some precondition is satisfied. To model situations of such kind, we introduce *conditional p-lists*.

Definition 2. *A* conditional p-list *(*cp-list*, for short) is a writing 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 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 3. *Let $q_1, \ldots, q_k$ be $k > 0$ atoms and let $pred$ be a binary program predicate. A* p-set *is of the form*

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

The program predicate $pred$ is supposed to be defined elsewhere in the program where the p-set occurs.

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, i.e., a certain model of the program). Let X be the set of atoms $\{q_1, \ldots, q_k\}$. Consider the binary relation $R \subseteq X^2$ 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 on q_j. The case of p-lists is a particular case 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.

Because of the presence of incomparable resources and equivalent resources, R can be seen as a representation of a collection of p-lists, one for each possible total order on X compatible with R. In particular, in case of equally preferable options, a nondeterministic choice is made. Instead, in case of incomparable options one among the possible total orderings of these options is arbitrarily selected.

Compound preferences are allowed in RASP. For lack of space we do not report the full definition, however p-lists (and cp-lists) are generalized so that preferences can be expressed among sets of objects, instead of simple options.

4 Preferences in cardinality constraints

It might be useful to enhance the modeling power of cardinality constraints by exploiting complex preferences such as those that can be expressed in RASP. In the example below, we reconsider menus and courses to state by means of a p-list that pasta is preferred over meat.

$$2\{in\_menu(X, C) : course(C) \mid in\_menu(X, pasta) > in\_menu(X, meat)\}3 \leftarrow meal(X).$$

where constants occurring in the p-list are among the possible values of variable C, i.e., are elements of the domain of $course$ (cf., the knowledge base shown in Section 2).

Notice that we do not need to express preferences over all possible values of C (even though this is possible). In the example, we state that we prefer to include pasta rather than meat, and we are indifferent about the third course. Preference is "soft" in the sense that pasta will be chosen if available, otherwise meat will be selected, again if available, otherwise some other course will be selected anyway.

Below we extend our example by employing the $pref\_when$ form of conditional preference, i.e., a cp-list. precisely, we state that in summer we prefer fruit over cake.

$$2\{in\_menu(X, C) : course(C) \mid in\_menu(X, fruit) > in\_menu(X, cake) \\ pref\_when\ summer\}3 \leftarrow meal(X).$$

Finally, we may employ p-sets to state that we prefer the less caloric courses.

$$2\{in\_menu(X, C) : course(C) \mid less\_caloric[X]\}3 \leftarrow meal(X).$$

Notice that $less\_caloric$ is, according to Definition 3, a binary predicate. The notation $less\_caloric[X]$ means that the comparison is on couples of distinct instances of variable X. This specification is necessary as different domain predicates defined over different variables may occur in constraints: this requires to indicate which variable must be considered for defining a p-set. Moreover, as Def. 4, to be seen, specifies, multiple preference are allowed in a constrait. Here, the p-set actually occurring (implicitly) in the constraint is

$$\{pasta, meat, fruit, cake : less\_caloric\}$$

i.e., the p-set is defined over the domain of the domain predicate $course$ (cf., Def. 3).

We call this new kind of cardinality constraints *p-constraints*. P-constraints may occur in the head of program rules.

As p-lists can be defined both on constants and atoms, we can extend our example as follows, where, for instance, Italian food is preferred to Chinese food. In general, for expressing preferences we may employ any variable occurring in the rule where the constraint appears.

$$2\{in\_menu(X, C) : course(C) : italian(C) > chinese(C)\}3 \leftarrow meal(X).$$

The general form of non-ground p-constraints is the following

Definition 4. *A p-constraint is of the form:*

$$l\{a_1, \ldots, a_n, \ldots, not\ b_1, \ldots, not\ b_m : D \mid C_p\}u$$

where the a_is and b_js are atoms, D is a set of atoms (concerning domain predicates), and C_p is a collection of preference specifications possibly including p-lists, cp-lists and binary predicates defining p-sets. Each two preference specifications in C_p are defined on distinct variables.

Notice that the purpose of the set of atoms D consists in defining the domains of the variables occurring in the a_is and b_js, as happens for the standard definition of weight constraints [31, 34].

In our extended ASP language, a program rule has the form:

$$C_0 \leftarrow C_1, \ldots, C_n.$$

where C_0 is a p-constraint and the C_is are (following [31, 34]) weight constraints. As a special case, each of the Cs can be a plain literal. Let a preference-program (for short p-program, or simply program) Π be a set of program rules. Notice that preferences may occur only in the heads of rules, i.e., preferences might influence what should be derived from a rule.

In future work, we intend to overcome the limitation of preference specifications in C_p to be disjoint. We intend to admit interacting preferences, and even preferences (or, better, priorities) among preferences. As a first step, we intend to allow p-lists and cp-lists to be defined over p-sets. For instance, referring to the above example of preferring less caloric courses, an extension would be to allow one to prefer less caloric courses in the first place, the best in quality in the second place, and the less expensive in the third place. I.e., we would allow expressions such as:

$$less\_caloric[X] > better\_quality[X] > less\_expensive[X]$$

5 Semantics

In this section, we introduce an extension to the semantics of weight constraints as specified in [34], so as to accommodate p-constraints. We implicitly consider the ground version of given program Π.

By adapting to weight constraints the approach developed for RASP in [14, 16], we introduce a function aimed at encoding the possible orderings on domain elements, according to the preferences expressed through p-constraints. The key point of the semantic modification that we propose is exactly to exploit such a function to reorder the atoms occurring in the p-constraints of Π. Then a (candidate) answer set is accepted only if it models the most preferred atoms.

We need the following preliminary definitions. Given a collection $\mathcal{S}$ of non-empty sets, a *choice function* $c(\cdot)$ for $\mathcal{S}$ is a function having $\mathcal{S}$ as domain and such that for each s in $\mathcal{S}$, $c(s)$ is an element of s. In other words, $c(\cdot)$ chooses exactly one element from each set in $\mathcal{S}$.

Let p be a binary predicate symbol p and I a set of ground atoms. Consider all the atoms of the form $p(a, b)$ in I. Let $I_{|\,p}$ denote the transitive closure of the set $\{p(a, b) \mid p(a, b) \in I\}$. Namely, $I_{|\,p}$ is the smallest set such that for all a, b, c it holds that $\big(p(a, b) \in I \vee (p(a, c) \in I_{|\,p} \wedge p(c, b) \in I_{|\,p})\big) \rightarrow p(a, b) \in I_{|\,p}$.

A given answer set might satisfy one or more of the atoms occurring in a p-list (resp., cp-list, p-set). Each atom q occurring in such a p-list has a degree of preference i associated with. We introduce a function ch in order to represent each pair $\langle q, i\rangle$, occurring in a p-list (resp., cp-list, p-set) of a p-constraint. In particular, for the case of p-lists we put

$$ch(q_1 > \cdots > q_k, I) = \big\{\{\langle q_1, 1\rangle, \ldots, \langle q_k, k\rangle\}\big\}.$$

Due to the presence of the preconditions, the representation of cp-lists is parameterized by a set I of p-atoms. Let $r \ = \ q_1 > \cdots > q_k \ pref\_when \ L_1, \ldots, L_n$. We put:

$$ch(r, I) = \begin{cases} ch(q_1 > \cdots > q_k, \emptyset) & \text{if } I \text{ satisfies } L_1, \ldots, L_n \\ \{\{\langle q_1, i_1\rangle, \ldots, \langle q_k, i_k\rangle\} \mid \{i_1, \ldots, i_k\} = \{1, \ldots, k\}\} & \text{otherwise} \end{cases}$$

A p-set $ps = \{q_1, \ldots, q_k \mid p\}$ represents a collection of alternative p-lists. Each of them is potentially exploitable in a given answer set. Given a set I of ground atoms, let us denote the set of p-lists represented by ps as follows:

$$PLists(ps, I) = \{ \ q_{i_1} > \cdots > q_{i_n} \mid \langle 1, \ldots, n\rangle \text{ is a maximal prefix of } \langle 1, \ldots, k\rangle \text{ such that } \forall j, h \, (j < h \rightarrow p(q_{i_h}, q_{i_j}) \notin I_{\mid p})\}$$

Then, we define $ch(ps, I) \ = \bigcup_{pl \in PLists(ps,I)} ch(pl, I)$. The definition of ch is then extended to rules by putting: $ch(\gamma, I) = \{ch(\ell, I) \ \mid \ \ell \text{ in the head of } \gamma\}$.

Finally, we associate to each rule γ, the set $\mathcal{R}(\gamma, I)$ of sets. Each $X \in \mathcal{R}(\gamma, I)$ is a collection of sets, each one having the form $\{\langle q_1, 1\rangle, \ldots, \langle q_k, k\rangle\}$, where q_i is an atom and i is a degree of preference. Given such an X, we say that each q such that $\langle q, 1\rangle \in x$ for some $x \in X$ is a *most preferred* element for X. Note that, each of the sets $\{\langle q_1, 1\rangle, \ldots, \langle q_k, k\rangle\}$ belonging to X encodes an ordering (i.e., a preference) on the atoms of one of the p_γ p-lists (resp., cp-lists, p-sets) occurring in γ. (Hence, for a fixed rule γ, each of the sets X in $\mathcal{R}(\gamma, I)$ has cardinality equals to p_γ.)

$$\mathcal{R}(\gamma, I) = \{\{c(s) \mid s \text{ in } ch(\gamma, I)\} \ \mid \ \text{for } c \text{ choice function for } ch(\gamma, I).\}$$

where c ranges on all possible choice functions for $ch(\gamma)$.

Then, a candidate answer set S is actually an answer set if it chooses from each ground p-constraint C of the form

$$l\{g_1, \ldots, g_n, \ldots, not\ h_1, \ldots, not\ h_m\}u$$

that occurs in the head of a rule γ, the most preferred elements (whatever their number is) according to some $X \in \mathcal{R}(\gamma, S)$. More formally:

Definition 5. *A set of atoms S is an answer set for program Π if the following conditions hold.*

- *S is an answer set of Π according to Definitions 2.7 and 2.8 in [34].*
- *For every p-constraint C, head of a rule $\gamma \in \Pi$, S includes all the most preferred elements of C, w.r.t. at least one of the $X \in \mathcal{R}(\gamma, S)$.*

Notice that we do not have to consider interaction among different rules: our preferences are in fact *local* to the p-constraint where they occur. Different p-constraints can be defined over different, even contrasting, preferences.

It is easy to get convinced that function ch can be computed in polynomial time. Therefore obtain the following results, that guarantee that the further modeling power that we offer implies no computational expense.

Theorem 1. *Deciding whether a ground preference-program P has answer sets is NP-complete.*

Theorem 2. *Deciding that a set of atoms is an answer set of a ground preference-program P is NP-complete.*

If one would now like to choose the "best preferred" answer sets, a *preference criterion* should be provided to compare answer sets. Such a criterion should impose an order on the collection of answer sets by reflecting the preference degrees in the p-lists. In a sense, any criterion should aggregate/combine all "local" partial orders to obtain a global one. Fundamental techniques for combining preferences (seen as generic binary relations) can be found for instance in [1] (see also, among many, [32, 23, 24, 33], and the references therein). Regarding combination of preferences in Logic Programming, criteria are also given, for instance, in [3, 9, 8, 36]. However, the complexity of finding the best preferred answer sets increases according to the selected criterion (see [14] for a discussion).

6 Concluding Remarks

In this paper we have presented an approach to expressing preferences in ASP cardinality constraints. Work is under way for implementing the proposed extension within our RASP implementation Raspberry (`http://www.dmi.unipg.it/formis/raspberry/`). The implementation will allow us to experiment the actual usefulness of the approach in practical problems.

Future work includes: the introduction of preferences among sets of options; the extension of preference treatment to the general form of weight constraints.

The main point however is the full integration of weight constraints and other forms of aggregates with RASP, i.e., the introduction of resource usage in p-constraints and in their future evolutions. In fact, as mentioned, weight constraints have proved useful in many applications, among which configuration. RASP is very promising in this direction, as one may specify not only the qualitative aspects of processes, but also the quantitative aspects that are in many cases of some or great importance. Preferences are often related to resources, as an agent may prefer to consume (or, more generally, to "invest") some resources rather than others, and may also have preferences about what one should try to obtain with the available resources. Consumption or production of some resource may have a relevance, that can be expressed by the weights in weight constraint. This kind of extended formalism can find suitable applications in the realm in bio-informatics, where reactions involve quantities, weights and byproducts, and may happen according to complex preferences. Exploring this field is an exciting perspective of our work.

Also, though a full modal logic is at present beyond our reach, in [13] we have introduced a (limited) form of reasoning about possibility and necessity by means of ASP modules. We mean to extend this approach to reasoning about preferred worlds.

References

[1] H. Andréka, M. Ryan, and P.-Y. Schobbens. Operators and laws for combining preference relations. *Journal of Logic and Computation*, 12(1):13–53, 2002.

[2] Web references for some ASP solvers. Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.

[3] M. Balduccini and V. S. Mellarkod. CR-Prolog$_2$ with ordered disjunction. In M. De Vos and A. Provetti, editors, *Proceedings of the ASP'03 Workshop*, volume 78 of *CEUR Workshop Proceedings*, 2003.

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

[5] M. Bienvenu, J. Lang, and N. Wilson. From preference logics to preference languages, and back. In F. Lin, U. Sattler, and M. Truszczyński, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010*. AAAI Press, 2010.

[6] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

[7] R. Brafman and M. Tennenholtz. Modeling agents as qualitative decision makers. *Artificial Intelligence*, 94(1-2):217–268, 1997.

[8] G. Brewka. Complex preferences for answer set optimization. In D. Dubois, C. A. Welty, and M.-A. Williams, editors, *Proceedings of 9th International Conference on the Principles of Knowledge Representation and Reasoning (KR'04)*, pages 213–223. AAAI Press, 2004.

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

[10] G. Brewka, I. Niemelä, and M. Truszczyński. Preferences and nonmonotonic reasoning. *AI Magazine*, 29(4), 2010.

[11] A. Capotorti and A. Formisano. Comparative uncertainty: theory and automation. *Mathematical Structures in Computer Science*, 18(1), 2008.

[12] L. Chen and P. Pu. Survey of preference elicitation methods. Technical report, EPFL Technical Report IC/2004, 2004.

[13] S. Costantini. Answer set modules for logical agents. In G. Gottlob, editor, *Datalog 2.0*, LNCS. Springer, 2011. Forthcoming.

[14] S. Costantini and A. Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic*, 64(1), 2009.

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

[16] S. Costantini, A. Formisano, and D. Petturiti. Extending and implementing RASP. *Fundamenta Informaticae*, 103, 2011.

[17] 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.

[18] D. Dubois, H. Fargier, H. Prade, and P. Perny. Qualitative decision theory: from Savage's axioms to nonmonotonic reasoning. *Journal of the ACM*, 49(4):455–495, 2002.

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

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

[21] S. Hallden. *On the logic of better*. No. 2 in Library of Theoria, Lund: Library of Theoria. 1957.

[22] S. O. Hansson. Preference logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 4. Kluwer, 2001.

[23] J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):37–71, 2004.

[24] J. Lang. Vote and aggregation in combinatorial domains with structured preferences. In M. M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1366–1371. 2007.

[25] J. Lang, L. v. d. Torre, and E. Weydert. Hidden uncertainty in the logical representation of desires. In *Proc. of Eighteenth Intl. Joint Conf. on Artificial Intelligence (IJCAI2003)*, pages 685–690, 2003.

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

[27] S. Lichtenstein and P. Slovic. *The construction of preference*. Cambridge University Press, 2006.

[28] V. Lifschitz. Answer set planning. In *Proc. of ICLP '99 Conf.*, pages 23–37. The MIT Press, 1999. Invited talk.

[29] F. Liu. Von wrights the logic of preference revisited. *Synthese*, 175(1):69–88, 2009.

[30] V. W. Marek and M. Truszczyński. Stable logic programming - an alternative logic programming paradigm. In *25 years of Logic Programming Paradigm*, pages 375–398. Springer, 1999.

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

[32] S. Nitzan. *Collective preference and choice*. Cambridge University Press, 2009.

[33] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Aggregating partially ordered preferences. *Journal of Logic and Computation*, 19(3):475–502, 2009.

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

[35] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming (ASP'01): Towards Efficient and Scalable Knowledge*. AAAI Press, Menlo Park, 2001. Technical report SS-01-01.

[36] T. Son and E. Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.

[37] The Soar Group. SOAR: a general cognitive architecture for developing systems that exhibit intelligent behavior. URL: http://sitemaker.umich.edu/soar/home, 2007. Documentation, download and publications.

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

[39] A. Tversky. Assessing uncertainty. *Journal of the Royal Statistical Society. Series B*, 36, 1974.

[40] 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.

[41] D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming*, 6(1-2):107–167, 2006.

[42] G. H. von Wright. *The logic of preference*. Edinburgh University Press, 1963.