Augmenting Weight Constraints with Complex Preferences

Stefania Costantini
Università di L’Aquila, Italy
Stefania.Costantini@univaq.it

Andrea Formisano
Università di Perugia, Italy
formis@dmi.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. We present an approach to introduce preferences in the weight constraint construct, which is a very useful programming construct widely adopted in Answer Set Programming (ASP). We show the usefulness of the proposed extension, and we outline how to accordingly extend the ASP semantics.

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 other’s 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, (Hallden 1957) initiated a line of research that was subsequently systematized in (von Wright 1963) which is usually taken to be the seminal work in preference logic. This line of research continues nowadays: the works of (van Benthem, Girard, and Roy 2009) and (Liu 2009), for instance, develop new modal preference logics that improve over (Hallden 1957) in several directions. For other studies on preference the reader may refer, among many, to (Hansson 2001; Lichtenstein and Slovic 2006; Brafman and Tennenholtz 1997) and to the references therein.

Many forms of commonsense reasoning rely more or less explicitly upon some form of preferences. In contrast to expert knowledge, that is usually explicit, most commonsense knowledge is implicit and one of the issues in knowledge representation is making this knowledge explicit. Being able to express/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 (Boutilier et al. 2004; Chen and Pu 2004; Hansson 2001; Brafman and Tennenholtz 1997; Lang, Torre, and Weydert 2003; Bienvenu, Lang, and Wilson 2010), to mention a few). Explicit preferences for actions in rules have been studied in AI. As a notable example we mention the SOAR architecture (The Soar Group 2007).

Preferences handling in computational logic has been extensively studied too. The reader may refer, e.g., to (Delgrande et al. 2004; Brewka, Niemelä, and Truszczyński 2010) for recent overviews and discussion of many existing approaches to preferences. Many of these approaches have been defined in the context of ASP (Gelfond and Lifschitz 1988), a relatively recent paradigm of logic programming that has been successfully applied to many forms of knowledge representation and commonsense reasoning (cf., among others, (Baral 2003) and the references therein).

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 for graphs, then the answer sets represents the possible solutions to the problem. In knowledge representation, an ASP program can be used to formalize 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 (Capotorti and Formisano 2008). 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 (Brewka 2004) concerns scheduling problems to be solved according to both preferences and...
priorities. In (Brewka, Niemelä, and Truszczynski 2010) 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 we take in this paper, where of course the two points of view can be seen as complementary rather than exclusive.

In recent work (Costantini, Formisano, and Petturiti 2010) we have proposed RASP, an extension to ASP where complex forms of preferences can be flexibly expressed. In that work we have considered plain ASP. In this paper, we intend to introduce preferences into ASP extended with weight constraints (Simons, Niemelä, and Soininen 2002).

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. In this paper, we propose to enrich weight constraints by means of RASP-like preferences. For lack of space we explicitly 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 (cf. (Delgrande et al. 2004)), are based on establishing priorities/preferences among rules or preferences among atoms which are anyway globally valid in given program. A weight constraint represents a local context where RASP-like preferences find a natural application.

This paper is an excerpt of (Costantini and Formisano 2011), to which the reader is referred for the aspects not treated here because of lack of space.

**Cardinality Constraints in ASP**

Cardinality constraints are a special case of weight constraints (both are discussed in (Simons, Niemelä, and Soininen 2002), where their semantics is also presented). Though the computational complexity of ASP with weight constraints remains the same (deciding whether a program involving ground weight constraints has an answer set is NP-complete, and computing an answer set is FNP-complete), the modeling power of the extended language is higher.

A **Weight Constraint** is of the form:

\[
\begin{align*}
& l \leq \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \\
& \quad \not a_{n+1} = w_{a_{n+1}}, \ldots, \not a_{n+m} = w_{a_{n+m}} \} \leq u
\end{align*}
\]

where the \(a_i\)'s 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 \(\not a_j\) is \(w_{a_j}\). The numbers \(l\) and \(u\) give, respectively, the lower and upper bounds of the constraint. The weights and bounds are real numbers. The intended meaning is that an answer set \(I\) satisfies a weight constraint if \(I\) includes a subset of the atoms occurring in the constraint so that the corresponding sum of weights belongs to \([l, u]\), where the weight of a negative literal \(\not a_j\) is counted only if \(a_j\) is not in \(I\).

Plain literals can be seen as a special case of weight constraint, thus a rule has the following form, where the \(C_i\)'s are weight constraints:

\[
C_0 \leftarrow C_1, \ldots, C_n.
\]

A **Cardinality Constraint** is a weight constraint having all weights equal to one. In such case, the following shorthand form is provided:

\[
l \{a_1, \ldots, a_n, \not a_{n+1}, \ldots, \not a_{n+m} \} u
\]

**Conditional literals** are introduced to compactly write down weight constraints. They have the form \(l : d\) where \(l\) is a literal and the conditional part \(d\) is a domain predicate. A conditional literal \(l : d\) corresponds to the sequence of all the ground instances of \(l\) 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. As an example, assume you wish to state that a meal is composed of at least two and at most three courses. This may be expressed by the cardinality constraint

\[
2\{\text{menu}(X, C) : \text{course}(C)\} \leq \text{meal}(X).
\]

Assume now that the background knowledge base is

\[
\text{course}() = \text{course}() \cup \text{course}() \cup \text{meal}().
\]

The above constraint should be seen as a shorthand for

\[
2\{\text{menu}(X, \text{pasta}), \text{menu}(X, \text{meat}), \\
\text{menu}(X, \text{cake}), \text{menu}(X, \text{fruit})\} \leq \text{meal}(X).
\]

where every possible value for \(C\) is listed (thus the “domain predicate” \(\text{course}\) becomes irrelevant). 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 \(\text{meal}()\) and \(\text{meal}()\) in the knowledge base shown earlier):

\[
2\{\text{menu}(\text{lunch}, \text{pasta}), \text{menu}(\text{lunch}, \text{meat}), \\
\text{menu}(\text{lunch}, \text{cake}), \text{menu}(\text{lunch}, \text{fruit})\}.
\]

Notice that, because of weight constraints, there exist different answer sets such that one is subset of another (which is not possible in traditional answer sets semantics).

\footnote{The set of given program defining domain predicates consists of domain rules, syntactically restricted so as to admit a unique answer set (see (Baral 2003)).}
Preferences in RASP

RASP (Costantini, Formisano, and Petturiti 2010) is an extension to the ASP framework that allows the specification of various kinds of non-trivial preferences. RASP preferences follow the quite intuitive principles first formalized in (von Wright 1963), and illustrated at length, e.g., in (van Benthem, Girard, and Roy 2009). The first two principles state that any preference relation is asymmetric and transitive. An advancement of our approach over others is that preferences 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 $\phi$ to $\psi$ means that a state of affairs where $\phi \land \neg \psi$ holds is preferred to a state of affairs where $\psi \land \neg \phi$ holds. The fourth principle states that if I prefer $\psi$ to $\phi$ and $\phi$ to $\zeta$, then I will prefer $\psi$ to $\phi \land \zeta$. 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.

As an example, consider a RASP program specifying 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 (von Wright 1963), 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 > wholomilk 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 | less_caloric \}, the less caloric one among chocolate, nuts, coconut. These are instead instances of “extrinsic preferences”, i.e., preferences which come with some kind of “reason”, or “justification”. 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. In full RASP, quantities for ingredients and products are allowed to be specified, however, in this paper we neglect these aspects of RASP related to resources in order to concentrate on preferences.

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

A conditional p-list (cp-list, for short) is a writing of the form $(r \text{ pref when } L_1, \ldots, L_n)$, where $r = s_1 \succ \cdots \succ s_k$ is a p-list and $L_1, \ldots, L_n$ are literals.

Intuitively, a cp-list specifies that if all $L_1, \ldots, L_n$ are satisfied, then the choice among the $s_i$s occurring in $r$ is ruled by the preference expressed through $r$. Otherwise, no preference is expressed.

There might be cases in which useful preferences are not expressible as a linear order. 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 in expressing (collections of alternative) p-lists.

Definition 2 Let $q_1, \ldots, q_k$ be atoms and pred be a binary predicate. A p-set is of the form \{ $q_1, \ldots, q_k \mid \text{pred}$ \}.

The predicate pred is supposed to be defined elsewhere in the program where the p-set occurs, so as to induce a binary relation $R$ on $X = \{ q_1, \ldots, q_k \}$. $R$ does not need to be a partial order, e.g., it may imply cycles. The elements of the same cycle in $R$ are considered equally preferable. Moreover, there might exist elements of $X$ that are incomparable. $R$ is seen as a representation of a collection of p-lists, one for each possible total order on $X$ compatible with $R$.

Compound preferences among sets of objects are allowed in RASP. For lack of space we do not report on this here.

Preferences in cardinality constraints

It might be useful to enhance the modeling power of cardinality constraints by exploiting RASP-like preferences. In the example below, we consider menus and courses to state, through a p-list, that we prefer pasta over meat and we are indifferent about the third course.

$2(\text{menu}(X, C) \land \text{course}(C) \mid \text{menu}(X, \text{pasta}) \succ \text{menu}(X, \text{meat})) \leftarrow \text{meal}(X)$

where constants occurring in the p-list are among the possible values of variable $C$. (Note that we do not need to express preferences over all possible values of $C$.) Preference is “soft” in the sense that pasta will be chosen if available, otherwise meat will be selected, again if available. We may employ p-sets to state that we prefer the less caloric courses.

$2(\text{menu}(X, C) \land \text{course}(C) \mid \text{less_caloric}(C)) \leftarrow \text{meal}(X)$

Notice that less_caloric is, according to Definition 2, a binary predicate. The notation less_caloric(C) means that the comparison is on couples of distinct instances of variable C. 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. 3, to be seen, specifies, multiple preference are allowed in a constraint. 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. We call this new kind of cardinality constraints p-constraints. P-constraints may occur in the heads of rules. The general form of non-ground p-constraints is:

Definition 3 A p-constraint is a writing of the form:

$l(a_1, \ldots, a_n, \text{not } a_{n+1}, \ldots, \text{not } a_{m+n} : D \mid C_p)u$

where the $a_i$s 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.

Notice that the purpose of the set of atoms $D$ consists in defining the domains of the variables occurring in the $a_i$s, as happens for standard weight constraints (Simons, Niemelä,
and Soininen 2002). 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 $C_i$s are weight constraints. As a special case, each of the $C_i$s can be a plain literal. A preference-program (or simply program) $P$ is a set of program rules.

**Semantics**

An extension to the semantics of weight constraints as specified in (Simons, Niemelä, and Soininen 2002), so as to accommodate p-constraints can be introduced. In doing this one proceeds by adapting to weight constraints the approach developed for RASP in (Costantini, Formisano, and Petturiti 2010). For the lack of space we do not report on this issue. The interested reader might refer to (Costantini and Formisano 2011). We can state the following:

**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.

The preference degrees of the p-lists of a program induce a preference order on its answer sets. Accordingly, if one would like to choose the “best preferred” answer sets, a preference criterion should be exploited to compare answer sets. In a sense, any criterion should aggregate/combine all “local” partial orders to obtain a global one. Fundamental techniques for combining preferences can be found for instance in (Andréka, Ryan, and Schobbens 2002; Nitzan 2009; Lang 2007; Pini et al. 2009), and in the references therein. The complexity of finding the best preferred answer sets increases according to the selected criterion (Costantini, Formisano, and Petturiti 2010).

**Concluding Remarks**

In this paper we have presented an approach to express preferences in ASP cardinality constraints. Work is under way for implementing the proposed extension. 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 and quantitative reasoning in p-constraints and in their future evolutions. This kind of extended formalism can find applications in the realm in bio-informatics, where reactions involve quantities, weights and byproducts, and may happen according to complex preferences.

**References**


