Complex Preferences for Answer Set Optimization

Gerhard Brewka
University of Leipzig
Dept. of Computer Science
Augustusplatz 10-11
04109 Leipzig, Germany
brewka@informatik.uni-leipzig.de

Abstract

The main contribution of this paper is the definition of the preference description language PDL. This language allows us to combine qualitative and quantitative, penalty based preferences in a flexible way. This makes it possible to express complex preferences which are needed in many realistic optimization settings. We show that several preference handling methods described in the literature are special cases of our approach. We also demonstrate that PDL expressions can be compiled to logic programs which can be used as tester programs in a generate-and-improve method for finding optimal answer sets.

Introduction

Answer sets (Gelfond & Lifschitz 1991), originally invented to define the semantics of (extended) logic programs with default negation, have proven to be extremely useful for solving a large variety of AI problems. Two important developments were essential for this success:

1. the development of highly efficient answer-set provers, the most advanced among them being *Smodels* (Niemelä & Simons 1997) and *dlv* (Eiter *et al.* 1998),
2. a shift from a theorem proving to a constraint programming perspective (Niemelä 1999),(Marek & Truszczyński 1999).

It turned out that many problems, for instance in reasoning about actions, planning, diagnosis, belief revision and product configuration, have elegant formulations as logic programs so that models of programs, rather than proofs of queries, describe problem solutions (Lifschitz 2002; Soininen 2000; Baral 2003; Eiter *et al.* 1999). This view of logic programs as constraints on the sets of literals which may count as solutions has led to a new problem solving paradigm called answer set programming (ASP).

The predominant methodology in ASP is a generate-and-test method which proceeds as follows:

1. generate answer sets which represent potential solutions,
2. specify conditions which destroy those answer sets which do not correspond to actual solutions.

For instance, in graph colouring arbitrary assignments of colours to nodes constitute potential solutions. If we add the condition that an answer set is to be disregarded if it assigns the same colour to neighbouring nodes, then the remaining answer sets will be the solutions to our original graph colouring problem.

Answer set optimization goes one step further: from a constraint programming paradigm to a paradigm of qualitative optimization. In terms of the above mentioned methodology, a third step is added:

3. among the solutions generated by the program, pick one of the solutions with maximal quality.

The (relative) quality of an answer set is described using a preference ordering on answer sets. 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.

Many AI problems have natural formulations as optimization problems (see for instance (Brewka 2004a) for a discussion of abduction and diagnosis, inconsistency handling and solution coherence viewed as an optimization problem), and many problems which can be represented in a "hard" constraint programming paradigm have fruitful, more flexible refinements as optimization problems. Consider planning as a typical example. We know how to represent planning problems as ASP problems (Lifschitz 2002). Moving from a constraint programming to an optimization perspective allows us to specify criteria by which we can rank plans according to their quality. This allows us to select good plans (or to generate suboptimal plans if there is no way to satisfy all requirements).

An example for quantitative optimization is planning under action costs (Eiter *et al.* 2002a). Qualitative optimization techniques in the context of planning are investigated in (Son & Pontelli 2004).

Our interest in optimization based on qualitative preferences stems from the fact that for a variety of applications numerical information is hard to obtain (preference elicitation is rather difficult) - and often turns out to be unnecessary. On the other hand, if numerical information is available then it is often convenient to use it. For instance, it is sometimes rather straightforward to assign numerical penalties to suboptimal solutions, and our approach will allow for flexible combinations of qualitative and numerical, penalty based preference handling techniques.

Of course, the use of optimization techniques in answer set programing is not new. There is a large body of work on preferred answer sets, see for instance (Schaub & Wang 2001) and the references in that paper. Also some of the existing answer set solvers have (numerical) optimization facilities: *Smodels* with weight constraints has $maximize$ and $minimize$ statements operating on weights of atoms in answer sets (Simons, Niemelä, & Soininen 2002). An interesting application of these constructs to modeling auctions can be found in (Baral & Uyan 2001). *dlv* has weak constraints (Buccafurri, Leone, & Rullo 2000) of the form

$$\leftarrow body.\ [w{:}\ l]$$

where w is a numerical penalty and l is a priority level. For each priority level, the sum of the penalties of all violated constraints (i.e., constraints whose bodies are satisfied) is computed. The answer sets with minimal overall penalty in one level are compared based on the overall penalties of the next level, etc. Such constraints were used, for instance, to implement planning under action costs as described in (Eiter *et al.* 2002a) and for knowledge based information-site selection (Eiter *et al.* 2002b). For the use of logic programs with ordered disjunction (Brewka, Niemelä, & Syrjänen 2002) and ASO programs (Brewka, Niemelä, & Truszczyński 2003) for several generic optimization problems see (Brewka 2004a).

All these approaches are based on fixed built-in preference handling strategies. For realistic applications the availability of a variety of strategies is highly important, and in particular the possibility to combine different strategies in flexible ways. For this reason we develop in this paper the preference description language PDL. The language allows us to describe complex preferences among answer sets. Our approach shares a lot of motivation with ASO programs (Brewka, Niemelä, & Truszczyński 2003): we treat the logic program generating candidate solutions separately to make answer set selection independent form answer set generation, and the rules we use here to express preferences are the same as in (Brewka, Niemelä, & Truszczyński 2003), apart from allowing for the explicit specification of penalties. However, rather than using preference programs, that is sets of rules, to describe preferences, we use PDL expressions which give us much more flexibility.

The outline of the paper is as follows: we first give a short reminder on answer sets. We then discuss a course scheduling example motivating the need of complex preference combination methods. The subsequent section introduces syntax and semantics of PDL, the new preference description language. We also show how the preferences needed for the scheduling example can be expressed using PDL.

We then demonstrate how several preference handling methods described in the literature can be expressed using PDL, and we discuss complexity issues. Finally, we show how optimal answer sets can be computed on top of a standard answer set solver using a generate and improve strategy. For this purpose, arbitrary PDL expressions are compiled to logic programs. We conclude with a discussion of related work.

Answer sets: a short reminder

In this section we recall the definition of answer sets as introduced by (Gelfond & Lifschitz 1991). Readers familiar with answer sets can safely skip this section. We consider propositional extended logic programs with two kinds of negation, classical negation $\neg$ and default negation not. Intuitively, $\mathrm{not}\ a$ is true whenever there is no reason to believe a, whereas $\neg a$ requires a proof of the negated literal. An *extended logic program* (program, for short) P is a finite collection of rules r of the form

$$c \leftarrow a_1, \ldots, a_n, \mathrm{not}\ b_1, \ldots, \mathrm{not}\ b_m \qquad (1)$$

where the a_i, b_j and c are classical ground literals, i.e., either positive atoms or atoms preceded by the classical negation sign $\neg$. We denote by *head*(r) the head c of rule r and by *body*(r) the body $a_1, \ldots, a_n, \mathrm{not}\ b_1, \ldots, \mathrm{not}\ b_m$ of the rule. We will call $a_1, \ldots, a_n$ the *prerequisites* of the rule and use *pre*(r) to denote the set of prerequisites of r.

We say a rule r of the form (1) is *defeated by a literal* ℓ, if $\ell = b_i$ for some $i \in \{1, \ldots, m\}$, and we say it is *defeated by a set of literals* X, if X contains a literal that defeats r. Moreover, a rule r is applicable in X whenever it is not defeated by X and its prerequisites are in X. We denote this condition by $X \models$ *body*(r). A rule r is satisfied by X (denoted $X \models r$) if *head*$(r) \in X$ or if $X \not\models$ *body*(r). A set of literals X is *consistent* if, for all atoms a, $a \in X$ implies that $\neg a \notin X$.

An answer set of a program P is a set of literals S satisfying two conditions:

1. if $r \in P$ is applicable in S, then r is applied, that is, *head*$(r) \in S$, and
2. all literals in S have a non-circular derivation using only rules undefeated by S.

We can make this precise as follows:

Definition 1 *Let P be an extended logic program, and let X be a set of literals. The X-reduct of P, denoted P^X, is the collection of rules resulting from P by*

1. *deleting each rule which is defeated by X, and*
2. *deleting all default negated literals from the remaining rules.*

This construction is often called Gelfond-Lifschitz reduction, after its inventors.

Definition 2 *Let R be a collection of rules without default negation. Then, $Cn(R)$ denotes the smallest set S of literals such that*

1. *S is closed under R, i.e., for any rule $c \leftarrow a_1, \ldots, a_n$ in R, if $a_1, \ldots, a_n \in S$, then $c \in S$; and*
2. *S is logically closed, i.e., either S is consistent or $S = Lit(R)$, the set of all literals.*

Definition 3 *Let R be a collection of rules. Define an operator $\gamma_R(X)$ on sets of literals as follows:*

$$\gamma_R(X) = Cn(R^X)$$

Then, a set S of literals is an answer set of R iff $S = \gamma_R(S)$. The collection of answer sets of R is denoted by $AS(R)$.

As mentioned in the introduction, in many applications of answer set programming one can distinguish a generate and a test part of the program: the generate part produces candidate solutions, the test part destroys those answer sets which do not represent solutions. For the test part rules of the form $\leftarrow$ *body* with empty head are often used. Such a rule is an abbreviation for

$$new \leftarrow \text{not } new, body$$

where new is a new atom not appearing elsewhere in the program. The effect of the rule is that all answer sets satisfying body are eliminated. These rules are also called constraints.

Although answer set programs are basically propositional, it is common to use rule schemata containing variables. These schemata are representations of their ground instances, and current answer set solvers use intelligent ground instantiation techniques before the actual answer set computation takes place. We will also frequently use schemata with variables.

Here is a standard example, the graph colouring problem. Given a description of a graph in terms of atoms built from predicate symbols *node*$(\cdot)$ and *edge*$(\cdot,\cdot)$, the answer sets of this program contain colour assignments (r for red, b for blue, g for green) to nodes such that neighbouring nodes have different colours.[1]

$$\begin{aligned} col(X,r) &\leftarrow node(X), \text{not } col(X,b), \text{not } col(X,g)\\ col(X,b) &\leftarrow node(X), \text{not } col(X,r), \text{not } col(X,g)\\ col(X,g) &\leftarrow node(X), \text{not } col(X,b), \text{not } col(X,r)\\ &\leftarrow col(X,C), col(Y,C), edge(X,Y), X \neq Y \end{aligned}$$

A motivating example

In this section we want to illustrate the need for flexible preference strategies as provided by the language to be developed in this paper. We will consider a simple ASP-based scheduling system that assigns lecturers, time slots and rooms to university courses. To simplify our discussion we will assume that each lecturer has to teach exactly one course per semester.

The information needed to solve this problem includes the available lecturers $l_1, \ldots, l_n$, the available rooms $r_1, \ldots, r_m$, the time slots $s_1, \ldots, s_k$, and the courses $c_1, \ldots, c_j$. To represent this information in a logic program, we use atoms built from the unary predicates $lecturer$, $room$, $slot$ and $course$, respectively.

To solve this problem in the answer set programming paradigm it is convenient to use programs with cardinality constraints (Niemelä & Simons 2000; Simons, Niemelä, & Soininen 2002). Intuitively, a cardinality constraint of the form $l\{a_1, \ldots, a_r\}u$ is satisfied if at least l and at most u of the atoms a_i are satisfied, where l and u are integers. Similarly, $l\{a(x) : b(x)\}u$ is satisfied if at least l and at most u ground intances of $a(x)$ are satisfied, where x is replaced by a ground term g for which $b(g)$ holds. It was shown in (Simons, Niemelä, & Soininen 2002) that cardinality constraints do not increase complexity. For the purposes of this paper an intuitive understanding of cardinality constraints is sufficient. The reader is referred to the original papers for further details.

To make sure that each answer set contains an assignment of lecturers, rooms and time slots to courses, we can use the following rules:

$$\begin{aligned} &1\{teaches(L,C) : lecturer(L)\}1 \leftarrow course(C)\\ &1\{in(R,C) : room(R)\}1 \leftarrow course(C)\\ &1\{at(S,C) : slot(S)\}1 \leftarrow course(C) \end{aligned}$$

Solutions to the scheduling problem have to satisfy several hard constraints:

1. as discussed earlier, there is only one course per lecturer,
2. different courses cannot take place in the same room at the same time.

This can be expressed by the following constraints:

$$\begin{aligned} &\leftarrow teaches(L,C), teaches(L,C'), C \neq C'\\ &\leftarrow in(R,C), in(R,C'), at(S,C), at(S,C'), C \neq C' \end{aligned}$$

Each answer set now corresponds to a solution of our scheduling problem: the assignments of lecturers, rooms and time slots to, say, a course c are part of each answer set in the form of atoms $teaches(l,c)$, $in(r,c)$ and $at(s,c)$.

So far our logic program allows us to generate possible solutions of the scheduling problem. Of course, not all of these solutions are of the same quality since the personal preferences of lecturers are not yet taken into account. In the example, several kinds of preferences may exist:

1. Lecturers will have preferred courses which they like (and are able) to teach.
2. Some of the lecturers prefer to teach, say, in the morning, others may prefer afternoon or evening lectures.
3. Some lecturers may even have their preferred lecture rooms, maybe because they are close to their offices.
4. Finally, since in most realistic cases it is impossible to satisfy the personal preferences of each single lecturer, it is necessary to specify how conflicts are solved, in other words, which preferences are more important than others.

In case of the preferred courses one can ask each lecturer to rank courses, for instance using penalty values. A good solution with respect to these preferences then is one where the overall penalty is small. Preferences regarding time slots and rooms may be purely qualitative. Conflict solving in a university environment may be based on the rule that professors and their wishes are more important than assistants. In any case, we need flexible ways of expressing preferences, and it must be possible to combine them using various combination strategies to yield a single preference order on answer sets. The language PDL to be developed in the next section allows us to do this.

Preference description language

In this section we develop PDL, a language for representing preference information. This language will then be used to select maximally preferred answer sets of generating programs. The language generalizes the rule based preference programs of (Brewka, Niemelä, & Truszczyński 2003) in two respects:

[1] We follow the Prolog convention that terms starting with capital letters are variables.

- it allows us to combine qualitative and numerical, penalty based preference information within a single framework, and
- it allows us to use different preference combination strategies for different aspects of the answer sets.

Before introducing PDL we define what we mean by an answer set optimization problem.

Definition 4 *An answer set optimization problem (AOP) is a pair $O = (P, prex)$ where P is a logic program and $prex$ a PDL expression (to be defined below). A solution of O is an answer set of P which is optimal according to the preorder represented by $prex$.*

Note that the generating program P can be any kind of logic program (e.g. normal, extended, disjunctive etc.) as long as it generates answer sets, that is sets of literals. An expression of our preference description language PDL represents a preorder, that is a transitive and reflexive relation, on answer sets. A preorder $\geq$ induces an associated strict partial order $>$ via $S > S'$ iff $S \geq S'$ and not $S' \geq S$. An answer set S is optimal according to the preorder $\geq$ iff for each answer set S' such that $S' \geq S$ we also have $S \geq S'$. PDL expressions thus play a similar role in our framework as objective functions in numerical optimization.

The basic building blocks of PDL are rules which represent context dependent preferences. The rules are similar to the ones in (Brewka, Niemelä, & Truszczyński 2003) but allow us to specify numerical penalties for suboptimal options.

Definition 5 *Let A be set of atoms. A* preference rule *over A is of the form*

$$C_1\text{: } p_1 > \ldots > C_k\text{: } p_k \leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m$$

where the a_j and b_k are literals built from atoms in A, the C_i are boolean combinations over A, and the p_i are integers satisfying $p_i < p_j$ whenever $i < j$.

We use $C_1 > C_2 > \ldots > C_k \leftarrow body$ as abbreviation for $C_1\text{: } 0 > C_2\text{: } 1 > \ldots > C_k\text{: } k\text{-}1 \leftarrow body$. Rules of this kind were used in (Brewka, Niemelä, & Truszczyński 2003).

A *boolean combination* over A is a formula built of atoms in A by means of disjunction, conjunction, strong ($\neg$) and default (not) negation, with the restriction that strong negation is allowed to appear only in front of atoms, and default negation only in front of literals. For example, $a \wedge (b \vee \text{not } \neg c)$ is a boolean combination, whereas $\text{not } (a \vee b)$ is not. The restriction simplifies the treatment of boolean combinations later on.

Definition 6 *Let S be a set of literals, Satisfaction of a boolean combination C in S (denoted $S \models C$) is defined as:*

$S \models l$ *(l literal)*	*iff*	$l \in S$
$S \models \text{not } l$ *(l literal)*	*iff*	$l \notin S$
$S \models C_1 \vee C_2$	*iff*	$S \models C_1$ *or* $S \models C_2$
$S \models C_1 \wedge C_2$	*iff*	$S \models C_1$ *and* $S \models C_2$.

We can characterize the intuitive meaning of a preference rule of the form above as follows: given two answer sets S_1 and S_2 such that both satisfy the body of the rule and at least one of the options in the head, then S_1 is preferred to S_2 if, for some j, $S_1 \models C_j$ and $j < min\{i \mid S_2 \models C_i\}$. Moreover, as in (Brewka, Niemelä, & Truszczyński 2003) we consider answer sets for which the rule is irrelevant - because the body is not satisfied or because none of the alternatives in the head is satisfied - to be as good as the best answer sets. This is due to our penalty based view of rule preferences: if a rule is irrelevant to an answer set it does not seem appropriate to penalize the answer set at all, based on this rule. A preference rule thus represents a ranking of answer sets. Moreover, the penalty values associated with the options represent a numerical measure of our degree of dissatisfaction.

Preference rules are the basic building blocks of PDL. In addition, PDL allows us to specify combination strategies. Some of these strategies make use of the actual penalty values, for instance by using their sum. Others, like the Pareto strategy, are more qualitative in nature. For this reason not all combinations make sense and we restrict the syntax accordingly by distinguishing between a subset PDL^p of PDL, the penalty producing expressions.

Definition 7 *PDL^p and PDL expressions are inductively defined as follows:*

1. *if r is a preference rule then $r \in PDL^p$,*
2. *if $e_1, \ldots, e_k$ are in PDL^p then $(psum\ \ e_1 \ldots e_k) \in PDL^p$,*
3. *if $e \in PDL^p$ then $e \in PDL$,*
4. *if $e_1, \ldots, e_k$ are in PDL^p then $(inc\ \ e_1 \ldots e_k)$, $(rinc\ e_1 \ldots e_k)$, $(card\ e_1 \ldots e_k)$ and $(rcard\ e_1 \ldots e_k)$ are in PDL,*
5. *if $e_1, \ldots, e_k$ are in PDL then $(pareto\ e_1 \ldots e_k)$ and $(lex\ e_1 \ldots e_k)$ are in PDL.*

The semantics of a PDL expression is a preorder, that is, a reflexive and transitive relation, on answer sets. We first define penalties of answer sets, denoted $pen(S, prex)$, for the penalty generating preference expressions in PDL^p as follows:

1. If $prex$ is a rule of the form
$$C_1\text{: } p_1 > \ldots > C_k\text{: } p_k \leftarrow body$$
then:
$pen(S, prex) = p_j$, where $j = min\{i \mid S \models C_i\}$, if S satisfies $body$ and at least one C_i,
$pen(S, prex) = 0$ otherwise.
2. If $prex$ is a complex expression of the form
$$(psum\ e_1 \ldots e_k)$$
then $pen(S, prex) = \sum_{i=1}^{k} pen(S, e_i)$.

We use $Ord(prex)$ to denote the preorder associated with a PDL expression $prex$. For a rule r we have $(S_1, S_2) \in Ord(r)$ iff $pen(S_1, r) \leq pen(S_2, r)$.

For complex expressions the corresponding preorders are defined as follows:

Let $\geq_1, \ldots, \geq_k$ be the preorders represented by $e_1, \ldots, e_k$. Let $>_1, \ldots, >_k$ be the corresponding strict partial orders (defined as $S >_j S'$ iff $S \geq_j S'$ and not $S' \geq_j S$). Let $K = \{1, \ldots, k\}$. Furthermore, for the case where $e_1, \ldots, e_k$ are penalty producing, we define $P^n_S = \{j \in K \mid pen(S, e_j) = n\}$.

- $(S_1, S_2) \in Ord(pareto\ e_1 \ldots e_k)$ iff
  $S_1 \geq_j S_2$ for all $j \in K$.
- $(S_1, S_2) \in Ord(lex\ e_1 \ldots e_k)$ iff
  $S_1 \geq_j S_2$ for all $j \in K$ or
  $S_1 >_j S_2$ for some $j \in K$, and for all $i < j$: $S_1 \geq_i S_2$.
- $(S_1, S_2) \in Ord(inc\ e_1 \ldots e_k)$ iff
  $P^0_{S_1} \supseteq P^0_{S_2}$.
- $(S_1, S_2) \in Ord(rinc\ e_1 \ldots e_k)$ iff
  $pen(S_1, e_j) = pen(S_2, e_j)$ for all $j \in K$ or
  $P^p_{S_1} \supset P^p_{S_2}$, for some p, and for all $q < p$, $P^q_{S_1} = P^q_{S_2}$.
- $(S_1, S_2) \in Ord(card\ e_1 \ldots e_k)$ iff
  $|P^0_{S_1}| \geq |P^0_{S_2}|$.
- $(S_1, S_2) \in Ord(rcard\ e_1 \ldots e_k)$ iff
  $|P^p_{S_1}| = |P^p_{S_2}|$ for all p or
  $|P^p_{S_1}| > |P^p_{S_2}|$, for some p, and $|P^q_{S_1}| = |P^q_{S_1}|$ for all $q < p$.
- $(S_1, S_2) \in Ord(psum\ e_1 \ldots e_k)$ iff
  $\sum_{i=1}^k pen(S_1, e_i) \leq \sum_{i=1}^k pen(S_2, e_i)$.

$pareto$ is the standard Pareto ordering where S_1 is at least as good as S_2 if it is at least as good with respect to all constituent orderings $\geq_i$. S_1 is strictly preferred over S_2 if it is strictly better according to at least one of the constituent orderings and at least as good with respect to all other orderings.

lex is a lexicographic ordering which considers the constituent preorders $\geq_i$ in the order in which they appear in preference expressions. S_1 is as good as S_2 if it is as good with respect to all $\geq_i$, or if it is strictly better. S_1 is strictly better if, for some i, it is as good as S_2 with respect to $\geq_1, \ldots, \geq_{i-1}$, and strictly better with respect to $\geq_i$. An order $\geq_j$ is thus only used to distinguish between answer sets which are equally good with respect to all orderings appearing before $\geq_j$.

inc is an inclusion based strategy which prefers answer sets satisfying more (in the sense of set inclusion) orderings as well as possible, that is with lowest penalty 0.

$rinc$ (ranked inc) is as inc an inclusion based strategy, but does not consider penalty 0 only. If two answer sets are equally good with respect to penalty p, then in a next step the set of orderings satisfied with penalty $p+1$ is considered to compare answer sets. Orderings of this kind were used in (Brewka, Niemelä, & Syrjänen 2002).

$card$ is similar to inc but based on the *number* of orderings satisfied with penalty 0 rather than set inclusion.

$rcard$ is similar to $rinc$ in using increasing penalties to distinguish between answer sets. As $card$, it considers the number of orderings satisfied to a particular degree. Orderings of this kind were used in (Benferhat *et al.* 1993; Brewka, Benferhat, & Le Berre 2002).

$psum$ adds the penalties obtained by the component orderings and prefers answer sets where the sum is smaller.

Course scheduling, revisited

We are now in a position to specify the preferences for our course scheduling program which we discussed informally earlier. We assume that each lecturer can assign penalties to courses he does not like to teach. To make sure penalties do not become arbitrarily high, we allow each lecturer to assign a total of 10 penalty points to arbitrary courses. Each lecturer l_i will thus specify a set C_i of preference atoms of the form $teaches(l_i, c) : p$ such that

$$\sum_{teaches(l_i,c):p} p \;=\; 10.$$

Similarly, each lecturer l_i can express a set P_i of time and room preferences. For instance, the rule:

$$am(S) > pm(S) \leftarrow teaches(l_i, C), at(S, C)$$

expresses that lecturer l_i prefers teaching in the morning. Here we assume that the predicates am and pm are defined accordingly, for instance by specifying $am(7), \ldots, am(12), pm(13), \ldots, pm(18)$. The rule

$$in(r_1, C) > \top \leftarrow teaches(l_i, C)$$

specifies that l_1 prefers to teach in room r_1 and is indifferent about any other possible lecture room.

Finally, we need information about who is a professor and who is an assistant. Let C_p be the union of all C_i such that l_i is a professor, and let C_a be the union of all C_i such that l_i is an assistant. Similarly, let P_p and P_a be the collections of time and room preferences of professors, respectively the corresponding collections for assistants. We mentioned in our informal discussion that professors' preferences are more important than assistants' preferences. More precisely, we want to give C_p more importance than C_a, but consider C_a as more important than P_p.

We slightly abuse notation and write $(comb\ S)$ rather than $(comb\ s_1, \ldots, s_k)$ whenever $S = \{s_1, \ldots, s_k\}$ and the order of elements in S is irrelevant for the combination method $comb$. This is the case for all methods except lex.

The preferences involved in the scheduling problem can now be stated using the following PDL-expression:

$$(lex\ (psum\ C_p)(psum\ C_a)(pareto\ P_p)(pareto\ P_a)).$$

Although the example is still pretty simple it already demonstrates the importance of different preference handling strategies and combination methods.

Now assume a solution for the scheduling problem has been computed and published on the web, but at the last minute one of the lecturers becomes unavailable. In such a situation simply starting from scratch and rerunning the scheduling system would not be a good idea - even if the original preferences are taken into account - as this may lead to a new solution with different assignments of lecturers, rooms and slots for a large number of classes. This is

certainly unwanted: what we would like to have is a new solution which is *as close as possible to the original solution*.

Generating coherent solutions given small changes in the problem description is again an optimization problem. To solve this problem, we need a description of the old solution together with a specification of what we mean by closeness. In a qualitative setting closeness can be described in terms of preferences.

For the class scheduling system different types of new preferences play a role when an existing solution needs to be modified. In addition to the original personal preferences of the lecturers we have the following:

1. in general, not changing the original lecturer, time and room assignments is preferred over changing them,
2. if a change is necessary, then it is more desirable to change the room rather than the time slot of a class (because in that case no email notification is necessary),
3. if the time slot for a course needs to be changed, it is preferable to make changes which require fewer notifications to be sent to students, that is, it is better to reschedule a course with few students.

One can easily think of further preference criteria for such situations. All these preferences need to be taken into account and combined adequately. We are not going to formalize this extended problem here. However, we hope to have convinced the reader that the complex preferences involved in realistic problems require a description language like PDL.

Special cases

One of the nice properties of PDL is that a number of different approaches to be found in the literature can easily be expressed and thus turn out to be simple special cases of our approach. We have the following results:

1. In (Brewka, Niemelä, & Truszczyński 2003) preference programs are used for answer set optimization. Since the rules used in preference programs are a special case of our rules the translation is simple: a preference program

   $$P_{pref} = \{r_1, \ldots, r_k\}$$

   corresponds to the PDL expression

   $$(pareto\ r_1 \ldots r_k).$$

   An ASO program as defined in that paper is thus a simple special case of our approach.

2. The mentioned paper also discusses preference programs with meta-preferences which split the set of preference rules into preference levels $1, \ldots, n$. Assuming $r_{i,j}$ belongs, for each i, to level i we can express programs with meta-preferences as:

   $$(lex\ (pareto\ r_{1,1} \ldots r_{1,k_1}) \ldots (pareto\ r_{n,1} \ldots r_{n,k_n})).$$

3. Cardinality and inclusion based combination strategies as described in (Brewka, Niemelä, & Syrjänen 2002) for $LPOD$s can be described using $rinc$ and $rcard$. Let $P_\times$ be an $LPOD$, P a logic program such that the answer sets of $P_\times$ and P coincide.[2] Let

   $$\{r_1, \ldots, r_k\}$$

   be the set of preference rules obtained from $P_\times$ by replacing $\times$ with $>$. Let

   $$prex = (rinc\ r_1 \ldots r_k).$$

   Then S is an optimal answer set of $P_\times$ under the inclusion based strategy iff S is a solution for the answer set optimization problem $(P, prex)$. Similarly, by replacing $rinc$ with $rcard$ in the preference expression, we obtain solutions corresponding to optimal answer sets of $P_\times$ under the cardinality based strategy.

4. Weak constraints of the form

   $$\leftarrow body.\ [w]$$

   as implemented in dlv can be represented as preference rules of the form

   $$\top\colon w \leftarrow body$$

   where $\top$ is a tautology of the form $a \vee \text{not } a$, using the $psum$-strategy. Alternatively we can use the preference fact (preference rule with empty body) $body'\colon w$ where $body'$ is the conjunction of literals in the body. Weak constraints with priority levels of the form

   $$\leftarrow body.\ [w\colon l]$$

   have the same translation as above, but must be grouped according to their priority level l. For each priority level l let

   $$\{r_{l,1}, \ldots, r_{l,k_l}\}$$

   be the translations of the weak constraints of that level. The preference strategy can be expressed as:

   $$(lex\ (psum\ r_{1,1} \ldots r_{1,k_1}) \ldots (psum\ r_{n,1} \ldots r_{n,k_n}))$$

   where n is the greatest priority level (with minimal priority).

5. *Smodels* statements of the form

   $$minimize\{a_1 = w_1, \ldots, a_k = w_k\}$$

   can be represented as

   $$(psum\ a_1\colon w_1 \ldots a_k\colon w_k),$$

   sequences of such statements as

   $$(lex\ (psum \ldots) \ldots (psum \ldots)).$$

It is obvious that PDL allows us to express a lot more combination strategies than the ones discussed in this section.

[2] Note that this requires a nonstandard definition of answer sets for P since the answer sets of $P_\times$ are not necessarily inclusion minimal.

Complexity

The complexity results established for ASO-programs in (Brewka, Niemelä, & Truszczyński 2003) also hold for answer set optimization problems $(P, prex)$ and can be shown using similar proofs. The exact complexity depends on the class of generating programs P. To simplify the treatment, we consider here only generating programs where deciding existence of an answer set is **NP**-complete (Simons, Niemelä, & Soininen 2002). This class of programs includes ground normal programs (possibly extended with strong negation or weight and cardinality constraints). The following two results indicate that - as for ASO-programs - allowing preferences adds an extra layer of complexity.

Theorem 1 *Let $O = (P, prex)$ be an answer set optimization problem and S an answer set of P. Then deciding whether S is a solution of O is* **coNP***-complete.*

Theorem 2 *Given an an answer set optimization problem $O = (P, prex)$ and a literal l, deciding whether there is a solution S of O such that $l \in S$ is $\Sigma_2^{\mathbf{P}}$-complete.*

The complexity results imply that (unless the polynomial hierarchy collapses) the problem of finding a solution for an answer set optimization problem cannot be mapped in polynomial time to a problem of finding an answer set of a program P' obtained by translating $(P, prex)$. As mentioned above, the proofs of these theorems are similar to the proofs of the corresponding results in (Brewka, Niemelä, & Truszczyński 2003) and are therefore omitted.

Implementation

In (Brewka, Niemelä, & Truszczyński 2003) an implementation technique for computing optimal answer sets for ASO programs on top of a standard answer set prover has been developed. A similar technique has earlier been used in (Janhunen *et al.* 2000) to compute stable models of disjunctive logic programs using *Smodels*, and in (Brewka, Niemelä, & Syrjänen 2002) to compute preferred answer sets of logic programs with ordered disjunction. The computation is based on a tester program that takes as input an answer set and generates a strictly better one if such an answer set exists. The computation starts with an arbitrary answer set generated by the generator. This answer set is given to the tester program. If the tester fails to generate a strictly better one, we have found an optimal answer set and we are done. If a strictly better answer set is discovered, the tester is run with the new answer set as input. This continues until an optimal answer set is reached.

This technique can be adapted for computing optimal answer sets of an answer set optimization problem $(P, prex)$ by compiling $prex$ to a suitable tester program for a given answer set M. The tester program $T(P, M, prex) =$

$$P \cup D(M) \cup Trans(prex, 1) \cup \{\leftarrow \text{not } better_1\}$$

consists of the original generating program P, a description of the answer set M to be tested for optimality, denoted $D(M)$ (a description of the penalty values is sufficient, see below), the translation $Trans(prex, 1)$ of the preference expression, and a constraint eliminating answer sets which are not better than M with respect to $prex$. As our target language we will use the language of *Smodels* with cardinality constraints here (Simons, Niemelä, & Soininen 2002) which we also used for our course scheduling example. This language is particularly well-suited for the translation of the cardinality based strategies.

For the translation we need an indexing scheme for all subexpressions of $prex$. Expressions have a tree structure and the indexing can be done in a standard manner. We assume index 1 is assigned to the whole expression (this is the reason for index 1 in the constraint $\leftarrow$ not $better_1$ and in $Trans(prex, 1)$ above), 1.1 to 1.k to its immediate subexpressions, 1.1.1 to the first subexpression of expression 1.1 etc.

For each subexpression with index i we define predicates geq_i and $better_i$ which express that the new generated answer set is at least as good as the old one, respectively strictly better than the old one, according to the preorder represented by expression e_i. For the penalty producing expressions we have additionally a predicate pen_i. A preference rule $r =$

$$C_1\colon p_1 > \ldots > C_k\colon p_k \leftarrow a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m$$

with index i can be translated according to the techniques discussed in (Brewka, Niemelä, & Truszczyński 2003). In the translation we assume that atoms of the form $oldpen_i$ are used to represent the answer set M to be tested. $Trans(r, i)$ consists of the following rules:

$$\begin{array}{rcl}
geq_i & \leftarrow & pen_i(P_1), oldpen_i(P_0), P_1 \leq P_0 \\
better_i & \leftarrow & pen_i(P_1), oldpen_i(P_0), P_1 < P_0 \\
body_i & \leftarrow & a_1, \ldots, a_n, \text{not } b_1, \ldots, \text{not } b_m \\
heads_i & \leftarrow & c_j \quad \text{(for each } C_j\text{)} \\
pen_i(0) & \leftarrow & \text{not } body_i \\
pen_i(0) & \leftarrow & \text{not } heads_i \\
pen_i(p_1) & \leftarrow & c_1, body_i \\
pen_i(p_2) & \leftarrow & \text{not } c_1, c_2, body_i \\
 & \ldots & \\
pen_i(p_k) & \leftarrow & \text{not } c_1, \ldots, \text{not } c_{k-1}, c_k, body_i
\end{array}$$

Here c_j is an atom representing the fact that boolean combination C_j is satisfied. Such atoms are needed for all subexpressions of the boolean combinations. Additionally we have to add rules capturing the conditions under which the combinations are satisfied. For example, if C_j is a disjunction $C_h \vee C_l$, then we add rules $c_j \leftarrow c_h$ and $c_j \leftarrow c_l$.

For complex expressions $prex = (comb\ e_1 \ldots e_k)$ with index i ($comb$ is one of our 7 combinators) the translation consists of the translations of the subexpressions $e_1 \ldots e_k$ together with new rules for the whole expression. More precisely, we have $Trans(prex, i) =$

$$T_{comb}(i, k) \cup Trans(e_1, i.1) \cup \ldots \cup Trans(e_k, i.k).$$

We define the rules $T_{comb}(i, k)$ below for each of the 7 cases. The variable Z ranges over $\{1, \ldots, k\}$, J (possibly with index) over $\{0, \ldots, k\}$, and P, Q range over penalty values:

- $prex = (pareto\ e_1 \ldots e_k)$:
  $geq_i \leftarrow geq_{i.1}, \ldots, geq_{i.k}$
  $better_i \leftarrow geq_i, better_{i.1}$
  ...
  $better_i \leftarrow geq_i, better_{i.k}$
- $prex = (lex\ e_1 \ldots e_k)$:
  $geq_i \leftarrow geq_{i.1}, \ldots, geq_{i.k}$
  $geq_i \leftarrow better_i$
  $better_i \leftarrow better_{i.1}$
  $better_i \leftarrow geq_{1.1}, better_{i.2}$
  ...
  $better_i \leftarrow geq_{i.1}, \ldots, geq_{i.k-1}, better_{i.k}$
- $prex = (inc\ e_1 \ldots e_k)$:
  $geq_i \leftarrow \text{not } worse_i(0)$
  $better_i \leftarrow better_i(0), \text{not } worse_i(0)$
  $better_i(0) \leftarrow pen_{i.Z}(0), oldpen_{i.Z}(P), 0 < P$
  $worse_i(0) \leftarrow oldpen_{i.Z}(0), pen_{i.Z}(P), 0 < P$
- $prex = (rinc\ e_1 \ldots e_k)$:
  $geq_i \leftarrow \text{not } worse_i$
  $geq_i \leftarrow better_i$
  $worse_i \leftarrow worse_i(P)$
  $better_i \leftarrow better_i(P), \text{not } worse\text{-}upto_i(P)$
  $better_i(P) \leftarrow pen_{i.Z}(P), oldpen_{i.Z}(Q), P < Q$
  $worse_i(P) \leftarrow oldpen_{i.Z}(P), pen_{i.Z}(Q), P < Q$
  $worse\text{-}upto_i(P) \leftarrow worse_i(Q), Q \leq P$
- $prex = (card\ e_1 \ldots e_k)$:
  $geq_i \leftarrow card_i(0, J_1), oldcard_i(0, J_2), J_1 \geq J_2$
  $better_i \leftarrow card_i(0, J_1), oldcard_i(0, J_2), J_1 > J_2$
  $card_i(0, J) \leftarrow J\{pen_{i.1}(0), \ldots, pen_{i.k}(0)\}J$
  $oldcard_i(0, J) \leftarrow J\{oldpen_{i.1}(0), \ldots, oldpen_{i.k}(0)\}J$
- $prex = (rcard\ e_1 \ldots e_k)$:
  $geq_i \leftarrow \text{not } diffcard_i$
  $geq_i \leftarrow better_i$
  $diffcard_i \leftarrow card_i(P, J_1), oldcard_i(P, J_2), J_1 \neq J_2$
  $better_i \leftarrow better_i(P), \text{not } worse\text{-}upto_i(P)$
  $better_i(P) \leftarrow card_i(P, J_1), oldcard_i(P, J_2), J_1 > J_2$
  $worse_i(P) \leftarrow card_i(P, J_1), oldcard_i(P, J_2), J_1 < J_2$
  $worse\text{-}upto_i(P) \leftarrow worse_i(Q), Q \leq P$
  $card_i(P, J) \leftarrow J\{pen_{i.1}(P), \ldots, pen_{i.k}(P)\}J$
  $oldcard_i(P, J) \leftarrow J\{oldpen_{i.1}(P), \ldots, oldpen_{i.k}(P)\}J$
- $prex = (psum\ e_1 \ldots e_k)$:
  $pen_i(X) \leftarrow pen_{i.1}(X_1), ..., pen_{i.k}(X_k), X = X_1 + ... + X_k$
  $geq_i \leftarrow pen_i(X), oldpen_i(Y), X \leq Y$
  $better_i \leftarrow pen_i(X), oldpen_i(Y), X < Y$

The space required for the translation to non-ground programs is polynomial in the size of $prex$. For the ground instantiation we need to restrict the possible penalty values to a finite subset of the integers. Not surprisingly, the translations of the ranked strategies $rinc$ and $rcard$ are the most involved ones. Some of the rules for $rinc$ have $k \times p^2$ instances, where p is the number of different penalty values. It is thus advisable to keep p rather low. A complete translation example is contained in the Appendix.

The description $D(M)$ of the candidate answer set M consists of ground atoms of the form $oldpen_i(v)$ for all penalty producing expressions (preference rules and $psum$-expressions) in $prex$, the preference expression of the current optimization problem. The easiest way to produce these atoms is to add to the original program P a variant of the rules defining pen_i in the translations of preference rules and of $psum$-expressions, where each occurrence of pen_i for some index i is replaced by $oldpen_i$. We have the following theorem:

Theorem 3 *Let M be an answer set of P. If $T(P, M, prex)$ has an answer set S, then S restricted to the original language of P is an answer set for P that is strictly preferred to M according to $Ord(prex)$, and if $T(P, M, prex)$ has no answer set, then no answer set strictly preferred to M exists.*

Proof: (Sketch) Consider first the resulting program without the constraint $\leftarrow \text{not } better_1$. This program contains the original generating program P. As can easily be verified, the remaining rules are stratified. Moreover, apart from the lowest stratum which contains literals from P, the language of P and that of the rest of the program, that is, of $D(M) \cup Trans(prex, 1)$, is different. We can thus apply the splitting set theorem (Lifschitz & Turner 1994) and show that there is a bijection between answer sets of P and answer sets of $P \cup D(M) \cup Trans(prex, 1)$ such that for each answer set S of P there is an answer set S' of $P \cup D(M) \cup Trans(prex, 1)$ with $S = S' \cap L(P)$, where $L(P)$ is the language of P.

It can be shown by induction on the structure of $prex$ that an answer set S' of $P \cup D(M) \cup Trans(prex, 1)$ contains geq_1 iff $(S' \cap L(P), M) \in Ord(prex)$, and S' contains $better_1$ iff $(S' \cap L(P), M)$ is an element of the strict partial order induced by $Ord(prex)$. The proof is somewhat tedious, but not difficult since the different combination methods are represented declaratively in the programs resulting from the translation.

With these results it immediately follows that adding the constraint $\leftarrow \text{not } better_1$ has the effect of destroying all answer sets S' of $T(P, M, prex)$ for which $S = S' \cap L(P)$ is not strictly better than M. □

Discussion

The main contribution of this paper is the definition of the language PDL for specifying complex preferences. We showed that several existing preference handling methods turn out to be special cases of our approach. We also demonstrated that PDL expressions can be compiled to logic programs to be used as tester programs in a generate-and-improve method for finding optimal answer sets.

Answer set programming is basically a propositional approach, and the currently available answer set solvers work for ground programs only. Nevertheless, rule schemata are widely used and sophisticated ground instantiation techniques have been implemented. They can be used in PDL as well in almost all cases because most of the PDL combination strategies are independent of the order of their immediate subexpressions. The single exception is lex where the order clearly matters. A rule schema can be used as a direct subexpression of lex only if there is an additional specification of the order of the ground instances, that is, if a *list* of ground instances is represented rather than a set.

The work presented in this paper shares some motivation with (Brewka 2004b). Also in that paper a language, called LPD, for expressing complex preferences is presented. However, there are several major differences which are due to the fact that PDL is taylored towards answer set optimization:

1. LPD is goal based rather than rule based. The basic building blocks are ranked knowledge bases consisting of ranked goals rather than rules with prioritized heads.
2. LPD is used to describe the quality of models. Since PDL is used to assess the quality of answer sets (i.e., sets of literals) rather than models, it becomes important to distinguish between an atom not being in an answer set and its negation being in an answer set. In other words, the distinction between classical negation and default negation (negation as failure) is relevant. This distinction does not play a role in LPD.
3. PDL distinguishes between penalty producing and other strategies. Both numerical and qualitative combination strategies are thus used. LPD focuses entirely on qualitative methods.

An interesting related paper is (Son & Pontelli 2004) which introduces a preference language for planning. The language is based on a temporal logic and is able to express preferences among trajectories. Preferences can be combined via certain binary operators. The major difference certainly is that our approach aims at being application-independent, whereas (Son & Pontelli 2004) is specifically geared towards planning.

Also related is (Andreka, Ryan, & Schobbens 2002). The authors investigate combinations of priority orderings based on a generalized lexicographic combination method. This method is more general than usual lexicographic orderings - including the ones expressible through our lex operator - since it does not require the combined orderings to be linearly ordered. It is based on so-called priority graphs where the suborderings to be combined are allowed to appear more than once. The authors also show that all orderings satisfying certain properties derived from Arrow's conditions (Arrow 1950) can be obtained through their method. This is an interesting result. On the other hand, we found it somewhat difficult to express examples like our course scheduling problem using this method. We believe our language is closer to the way people actually describe their preferences.

In (Boutilier *et al.* 1999; 2004; Brafman & Dimopoulos 2003) CP-networks are introduced, together with corresponding algorithms. These networks are a graphic representation, somewhat reminiscent of Bayes nets, for conditional preferences among feature values under the *ceteris paribus* principle. Our approach differs from CP-networks in several respects:

- Preferences in CP-networks are always total orders of the possible values of a single variable. We are able to relate arbitrary formulas in the heads of rules - and to express partial preference information, that is, to refrain from judgement if we do not care.
- The ceteris paribus interpretation of preferences is very different from our rule-based interpretation. The former views the available preferences as (hard) constraints on a global preference order. Each preference relates only models which differ in the value of a single variable. Our preference rules, on the other hand, are more like a set of different criteria in multi-criteria optimization. In particular, rules can be conflicting. Conflicting rules may neutralize each other, but do not lead to inconsistency.

Many related ideas can be found in constraint satisfaction, in particular valued (sometimes also called weighted) constraint satisfaction (Freuder & Wallace 1992; Fargier, Lang, & Schiex 1993; Schiex, Fargier, & Verfaillie 1995; Bistarelli, Montanari, & Rossi 1997). Here a solution is an assignment of values to variables. A valued constraint, rather than specifying hard conditions a solution has to satisfy, yields a ranking of solutions. A global ranking of solutions then is obtained from the rankings provided by the single constraints through some combination rule. This is exactly what happens in our approach based on preference rules. Also in constraint satisfaction we find numerical as well as qualitative approaches. In MAX-CSP (Freuder & Wallace 1992), for instance, constraints assign penalties to solutions, and solutions with the lowest penalty sum are preferred. In fuzzy CSP (Fargier, Lang, & Schiex 1993) each solution is characterized by the worst violation of any constraint. Preferred solutions are those where the worst violation is minimal. We are not aware of any approach in constraint satisfaction trying to combine different strategies. For this reason we believe the language developed here will be of interest also for the constraint community.

Although we presented PDL in the context of answer set optimization, it should be obvious that the language can be used in other optimization contexts - like constraint optimization - as well. To use PDL it is only required that candidate solutions can be evaluated with respect to the expressions in the rules. We also want to emphasize that we consider PDL as an extendible language. We certainly do not claim that the 7 combinators used in this paper are the only interesting ones. Many other combination methods have been used in different areas of AI, for an excellent overview see (Lang 2004). The reader should be aware, though, that for the compilation of preference expressions to logic programs it is essential that two answer sets can be directly compared.

Acknowledgements

The author acknowledges support from the EC (IST-2001-37004, WASP) and DFG (BR 1817/1-5, Computationale Dialektik). Thanks to R. Booth for proofreading.

Appendix: Translation Example

In this appendix we illustrate our compilation method and give the complete translation of the preference expression

$$(lex\ r_1\ (pareto\ r_2\ r_3))$$

with

$$r_1:\ a \vee b > g \leftarrow f$$
$$r_2:\ a \wedge \text{not}\ d > e \leftarrow \text{not}\ c$$
$$r_3:\ a \vee \neg b > f$$

Our indexing scheme assigns index 1 to lex, 1.1 to r_1, 1.2 to $pareto$, 1.2.1 to r_2 and 1.2.2 to r_3. The translation of lex yields:

$$
\begin{array}{rcl}
geq_1 & \leftarrow & geq_{1.1}, geq_{i.2} \\
geq_1 & \leftarrow & better_1 \\
better_1 & \leftarrow & better_{1.1} \\
better_1 & \leftarrow & geq_{1.1}, better_{1.2}
\end{array}
$$

For r_1 we obtain:

$$
\begin{array}{rcl}
geq_{1.1} & \leftarrow & pen_{1.1}(P_1), oldpen_{1.1}(P_0), P_1 \leq P_0 \\
better_{1.1} & \leftarrow & pen_{1.1}(P_1), oldpen_{1.1}(P_0), P_1 < P_0 \\
body_{1.1} & \leftarrow & f \\
heads_{1.1} & \leftarrow & c_{a \vee b} \\
heads_{1.1} & \leftarrow & c_g \\
c_{a \vee b} & \leftarrow & a \\
c_{a \vee b} & \leftarrow & b \\
c_g & \leftarrow & g \\
pen_{1.1}(0) & \leftarrow & \text{not } body_{1.1} \\
pen_{1.1}(0) & \leftarrow & \text{not } heads_{1.1} \\
pen_{1.1}(0) & \leftarrow & c_{a \vee b}, body_{1.1} \\
pen_{1.1}(1) & \leftarrow & \text{not } c_{a \vee b}, c_g, body_{1.1}
\end{array}
$$

From $pareto$ we get:

$$
\begin{array}{rcl}
geq_{1.2} & \leftarrow & geq_{1.2.1}, geq_{1.2.2} \\
better_{1.2} & \leftarrow & geq_{1.2}, better_{1.2.1} \\
better_{1.2} & \leftarrow & geq_{1.2}, better_{1.2.2}
\end{array}
$$

The rule r_2 gives us:

$$
\begin{array}{rcl}
geq_{1.2.1} & \leftarrow & pen_{1.2.1}(P_1), oldpen_{1.2.1}(P_0), P_1 \leq P_0 \\
better_{1.2.1} & \leftarrow & pen_{1.2.1}(P_1), oldpen_{1.2.1}(P_0), P_1 < P_0 \\
body_{1.2.1} & \leftarrow & \text{not } c \\
heads_{1.2.1} & \leftarrow & c_{a \wedge \text{not } d} \\
heads_{1.2.1} & \leftarrow & c_e \\
c_{a \wedge \text{not } d} & \leftarrow & a, \text{not } d \\
c_e & \leftarrow & e \\
pen_{1.2.1}(0) & \leftarrow & \text{not } body_{1.2.1} \\
pen_{1.2.1}(0) & \leftarrow & \text{not } heads_{1.2.1} \\
pen_{1.2.1}(0) & \leftarrow & c_{a \wedge \text{not } d}, body_{1.2.1} \\
pen_{1.2.1}(1) & \leftarrow & \text{not } c_{a \wedge \text{not } d}, c_e, body_{1.2.1}
\end{array}
$$

Finally, r_3 yields:

$$
\begin{array}{rcl}
geq_{1.2.2} & \leftarrow & pen_{1.2.2}(P_1), oldpen_{1.2.2}(P_0), P_1 \leq P_0 \\
better_{1.2.2} & \leftarrow & pen_{1.2.2}(P_1), oldpen_{1.2.2}(P_0), P_1 < P_0 \\
body_{1.2.2} & & \\
heads_{1.2.2} & \leftarrow & c_{a \vee \neg b} \\
heads_{1.2.2} & \leftarrow & c_f \\
c_{a \vee \neg b} & \leftarrow & a \\
c_{a \vee \neg b} & \leftarrow & \neg b \\
c_f & \leftarrow & f \\
pen_{1.2.2}(0) & \leftarrow & \text{not } body_{1.2.2} \\
pen_{1.2.2}(0) & \leftarrow & \text{not } heads_{1.2.2} \\
pen_{1.2.2}(0) & \leftarrow & c_{a \vee \neg b}, body_{1.2.2} \\
pen_{1.2.2}(1) & \leftarrow & \text{not } c_{a \vee \neg b}, c_f, body_{1.2.2}
\end{array}
$$

This completes the translation. Of course, the translation could be simplified. For instance, an atom like c_l representing an option in the head of a rule consisting of a literal can immediately be replaced by l. For sake of clarity we did not make simplifications of this kind above.

References

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

Arrow, K. 1950. A difficulty in the concept of social welfare. *Journal of Political Economy* 58:328–346.

Baral, C., and Uyan, C. 2001. Declarative specification and solution of combinatorial auctions using logic programming. In *Proceedings LPNMR-01*, 186–198. Springer LNCS 2173.

Baral, C. 2003. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press. ISBN 0521818028.

Benferhat, S.; Cayrol, C.; Dubois, D.; Lang, J.; and Prade, H. 1993. Inconsistency management and prioritized syntax-based entailment. In *Proceedings International Joint Conference on Artificial Intelligence, IJCAI-93*, 640–645. Morgan Kaufmann.

Bistarelli, S.; Montanari, U.; and Rossi, F. 1997. Semiring-based constraint solving and optimization. *Journal of the ACM* 44(2):201–236.

Boutilier, C.; Brafman, R.; Hoos, H.; and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proc. Uncertainty in Artificial Intelligence, UAI-99*, 71–80.

Boutilier, C.; Brafman, R.; Domshlak, C.; Hoos, H.; and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* to appear.

Brafman, R., and Dimopoulos, Y. 2003. A new look at the semantics and optimization methods of CP-networks. In *Proceedings International Joint Conference on Artificial Intelligence, IJCAI-03*, 1033–1038. Morgan Kaufmann.

Brewka, G.; Benferhat, S.; and Le Berre, D. 2002. Qualitative choice logic. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference, KR-02*, 158–169. Morgan Kaufmann. journal version to appear in Artificial Intelligence.

Brewka, G.; Niemelä, I.; and Syrjänen, T. 2002. Implementing ordered disjunction using answer set solvers for normal programs. In *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA 2002)*, 444–455. Springer Verlag.

Brewka, G.; Niemelä, I.; and Truszczyński, M. 2003. Answer set optimization. In *Proceedings International Joint Conference on Artificial Intelligence, IJCAI-03*, 867–872. Morgan Kaufmann.

Brewka, G. 2004a. Answer sets: From constraint programming towards qualitative optimization. In *Proceedings LPNMR-04*, 34–46. Springer Verlag.

Brewka, G. 2004b. A rank based description language for qualitative preferences. In *submitted for publication*.

Buccafurri, F.; Leone, N.; and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering* 12(5):845–860.

Eiter, T.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1998. The KR system dlv: Progress report, comparisons and benchmarks. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 6th International Conference, KR-98*, 406–417. Morgan Kaufmann.

Eiter, T.; Faber, W.; Leone, N.; and Pfeifer, G. 1999. The diagnosis frontend of the dlv system. *AI Communications* 12(1-2):99–111.

Eiter, T.; Faber, W.; Leone, N.; Pfeifer, G.; and Polleres, A. 2002a. Answer set planning under action costs. In *Proc. JELIA 2002*, volume 12(1-2), 186–197. Springer LNCS 2424.

Eiter, T.; Fink, M.; Sabbatini, G.; and Tompits, H. 2002b. A generic approach for knowledge-based information-site selection. In *Proc. 8th Intl. Conference on Principles of Knowledge Representation and Reasoning, KR-02*, 459–469. Morgan Kaufmann.

Fargier, H.; Lang, J.; and Schiex, T. 1993. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies*.

Freuder, E., and Wallace, R. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58(1):21–70.

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

Janhunen, T.; Niemelä, I.; Simons, P.; and You, J.-H. 2000. Unfolding partiality and disjunctions in stable model semantics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference*, 411–419. Morgan Kaufmann Publishers.

Lang, J. 2004. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence* to appear.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In *International Conference on Logic Programming*, 23–37.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence Journal* 138(1-2):39–54.

Marek, V., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, 375–398. Springer Verlag.

Niemelä, I., and Simons, P. 1997. Efficient implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*, 421–430. Springer Verlag.

Niemelä, I., and Simons, P. 2000. Extending the Smodels system with cardinality and weight constraints. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25(3,4):241–273.

Schaub, T., and Wang, K. 2001. A comparative study of logic programs with preference. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI-01*, 597–602.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, 631–637.

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

Soininen, T. 2000. *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks*. Ph.D. Dissertation, Helsinki University of Technology, Finland.

Son, R., and Pontelli, E. 2004. Planning with preferences using logic programming. In *Proc. 7th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 04*, 247–260. Springer LNAI 2923.