Expressing Preferences using Preference Set Constraint Atoms

Alex Brik, Jeffrey B. Remmel
Department of Mathematics, UC San Diego, USA

Abstract

This paper introduces an extension of Answer Set Programming called Preference Set Constraint Programming which is a convenient and general formalism to reason with preferences. PSC programming extends Set Constraint Programming introduced by Marek and Remmel (Marek and Remmel 2004) by introducing two types of preference set constraint atoms, measure preference set constraint atoms and pre-ordered preference set constraint atoms, which are extensions of set constraint atoms. We show that the question of whether a PSC program has a preferred stable model is CoNP-complete. We give examples of the uses of the preference set constraint atoms and show that Answer Set Optimization (Brewka, Niemelä, and Truszczynski 2003) and General Preference (Son and Pontelli 2006) can be expressed using preference set constraint atoms.

Introduction

The notion of a set constraint (SC) atom and a set constraint logic program was introduced by Marek and Remmel in (Marek and Remmel 2004). In this paper we extend these notions to define preference set constraint (PSC) atoms and PSC logic programs. The purpose of these extensions is to use PSC atoms to express preferences.

PSC programming is an intuitive and general formalism for expressing preferences. We demonstrate its generality by showing that PSC programing can be used to express optimal stable models of Answer Set Optimization (ASO) of (Brewka, Niemelä, and Truszczynski 2003) and general preferences of (Son and Pontelli 2006). An extension of PSC programming can be used to express preferred answer sets and weakly preferred answer sets of (Brewka and Eiter 1999). However, due to space limitations, we will not discuss the last two examples.

In this paper, we shall focus on the formal definitions of PSC programming. However, there are a number of interesting issues concerning the best way to implement PSC programs. While such issues are for the most part outside of the scope of this paper, we note that an implementation of PSC programming will not necessarily be a simple application of the definitions. The question of what is the best way to implement PSC programming efficiently is one that requires additional research, and we will only briefly discuss a few salient implementation issues in this article.

The subject of expressing preferences using Answer Set Programming (ASP) has been discussed extensively in the literature. Various approaches have been proposed. We refer the reader to the article (Brewka, Niemelä, and Truszczynski 2008) for an accessible overview of the subject and to a more detailed albeit older survey (Delgrande et al. 2004).

In (Delgrande et al. 2004) various approaches are classified for handling preferences in nonmonotonic reasoning. The paper identified the following criteria.

- **Host system**. This is a particular formalism which is extended to handle the preferences. In our case the host system is a SC logic program.
- **What is the preference ordering an ordering on**? PSC programming allows the user to directly specify preference orderings on sets of atoms.
- **Meta-level vs. object-level preferences**. This criteria identifies whether preferences are imposed "externally" on the host system, or the preferences are used within the object theory. Our approach is a meta-level approach.
- **Static vs. dynamic preferences**. This criteria specifies whether the preferences are fixed at the time that the theory is specified or can be determined "on the fly". PSC programming implements static preferences.
- **Properties of the preference ordering**. PSC programming enforces a pre-order on the set of PSC stable models.
- **Prescriptive vs. descriptive preferences**. This criteria concerns preference orderings on the rules and is not applicable to PSC programming.
- **From preference to preferred results**. This criteria identifies broad categories for methods that generate preferred answer sets from a theory and a set of preferences. In PSC programming preferred PSC stable models are standard PSC stable models satisfying additional criteria.

As noted in (Delgrande et al. 2004), the majority of ASP type systems that reason about preferences use a preference ordering on the set of rules to express preferences. An example of such an approach is given in (Brewka and Eiter 1999).

There are exceptions such as for instance (Sakama and Inoue 2000), (Brewka 2002) that specify preferences on the literals and more recently ASO which uses a second preference program to specify preferences on the answer sets. ASO is in fact the approach that is closest to PSC programming.

Delgrande et al. note that to have the most general ASP type system that can reason about preferences, one must be able to handle preferences on sets of objects. PSC programming allows the user to specify preferences on subsets of atoms. This fact distinguishes PSC programming from most other approaches for expressing preferences in ASP. PSC programming is different from ASO in that the host system of PSC programming is a SC logic program, whereas the host system of ASO is an ASP logic program. We will show that ASO programming can be viewed as a particular case of PSC programming.

Since literals can be viewed as sets of cardinality 1, there are similarities between PSC programming and some of the proposals that specify preferences on the literals. Due to the space constraints we will limit the related discussion to few remarks.

In (Brewka 2002) Brewka introduces Logic Programs with Ordered Disjunction (LPODs). The key feature of LPODs are ordered disjunctions of the form $C_1 \times ... \times C_n$ in the heads of the rules. The meaning is that if possible C_1, if not possible C_1 then C_2, ..., if not possible C_1, ...,C_{n-1} then C_n. The priorities among the answer sets of LPODs are generated using the indices of the disjuncts included in the stable models. As shown in (Brewka, Niemelä, and Syrjänen 2004) deciding whether S is a preferred answer set of LPOD P is coNP-complete, which is the complexity of the same problem for PSC programs. Thus a polynomial time translation between the formalisms exists.

In (Buccafurri, Leone, and Rullo 2000), Buccafurri et al. extend the language of Disjunctive Datalog by weak constraints (DATALOG$^{\vee,\neg,c}$). The weak constraints are constructs of the form $\Leftarrow L_1, ..., L_n$ where L_1, ..., L_n are literals. The weak constraints effectively allow to specify sets of atoms, i.e. a weak constraint specifies those sets of atoms that violate the constraint. Γ is a model of a DATALOG$^{\vee,\neg,c}$ program if Γ is a model of the underlying Disjunctive Datalog program and if it minimizes the sum of the weights of the violated weak constraints. DATALOG$^{\vee,\neg,c}$ programs possess a certain similarity with the measure PSC programs. In particular both specify preferences on sets and both use numeric weights for the models. However, there are significant differences between the two approaches. First, the host system of DATALOG$^{\vee,\neg,c}$ is Disjunctive Datalog programs while for measure PSC programs, it is SC programs. Second, integer weights associated with candidate models in DATALOG$^{\vee,\neg,c}$ allow the use of a binary search procedure in finding models, whereas the use of binary search is not helpful for the measure PSC programs since the stopping criteria is not easily determined due to the arbitrary weights. Thus if the complexities of the host systems were the same, the complexities of the search for optimal models would be different in the two approaches.

The rest of the paper is structured as follows. In section 2, we will describe the preliminaries of ASP and SC logic programming. In section 3, we will introduce PSC programming and provide a brief discussion of some key implementation issues for PSC programming. In section 4 we will show that PSC programming can be used to express optimal stable models of ASO. In section 5 we will show that the general preferences of (Son and Pontelli 2006) can be expressed using PSC atoms. In section 6, we will discuss computational complexity of PSC programming. In section 7, we will give conclusions and directions for further research.

Preliminaries

Answer Set Programming.

Answer set programming is logic programming with stable model or answer set semantics (Gelfond and Lifschitz 1988), (Gelfond and Lifschitz 1991). ASP systems are ideal logic-based systems to reason about a variety of types of data and integrate quantitative and qualitative reasoning. The question of whether a finite propositional logic program has a stable model is NP-complete (Elkan 1990), (Marek and Truszczynski 1991). It is also the case that any NP search problem can be (uniformly) reduced to the problem of finding a stable model of a finite propositional logic program (Marek and Remmel 2003).

A *normal propositional logic program* P consists of rules of the form

$$C = a \leftarrow a_1, ..., a_m,\ not\ b_1, ...,\ not\ b_n$$

where $a, a_1, \ldots, a_m, b_1, \ldots, b_n$ are atoms and not is a non-classical negation operator. The set $prem\,(C) = \{a_1,..., a_m\}$ is called the set of *premises* of rule C and the set $cons\,(C) = \{b_1, \ldots, b_n\}$ is called the set of *constraints* of rule C. The atom a is called the *conclusion* of rule C and is denoted by $c\,(C)$. Either $prem(C)$, $cons(C)$, or both may be empty. Let $H(P)$ denote the Herbrand base of P. A subset $M \subseteq H(P)$ is called a model of a rule C if $prem(C) \subseteq M$ and $cons(C) \cap M = \emptyset$ implies $c(C) \in M$. M is a model of a program P if it is a model of every rule of P.

Given $M \subseteq H(P)$, the Gelfond-Lifschitz transform P^M of P with respect to M is obtained by removing every rule C such that $cons\,(C) \cap M \neq \emptyset$ and then removing the constraints from all the remaining rules. M is called a **stable model** of P if M is the least model of P^M.

Set Constraint Logic Programming.

Set constraint programming was introduced in (Marek and Remmel 2004) as an extension of $DATALOG^{\neg}$. It generalized Answer Set Programming (ASP) with cardinality constraint atoms or weight constraint atoms as defined in (Niemelä, Simons, and Soininen 1999), (Niemelä and Simons 2000).

Suppose that we are given a finite set of atoms X. We let $\mathcal{P}\,(X)$ denote the set of all subsets of X. A *set constraint atom* over X is a pair $\langle X, F\rangle$ where $F \subseteq \mathcal{P}\,(X)$. Given a set of atoms M and a SC atom $\langle X, F\rangle$, we say that M satisfies $\langle X, F\rangle$ (or M is a model of $\langle X, F\rangle$) and write $M \models \langle X, F\rangle$ if $M \cap X \in F$.

A *set constraint* (SC) rule C is an expression of the form

$$s \leftarrow s_1, ..., s_k$$

where s, s_1, ..., s_k are SC atoms. The set $body\,(C) = \{s_1, ..., s_k\}$ will be referred to as the body of the rule C and the atom $c\,(C) = s$ will be referred to as the conclusion of the rule. A set of atoms M is a *model* of C (or M satisfies C) if $M \models s_1$, ..., $M \models s_k$ implies $M \models s$. An SC program P is a set of SC rules and a set of atoms M is a model of P if M is a model of every rule in P.

We note that the satisfaction of literals can be easily expressed in terms of the satisfaction of set constraints. That is, for an atom a, $M \models a$ if and only $M \models \langle \{a\}\,,\ \{\{a\}\}\rangle$ and $M \models not\ a$ if and only if $M \models \langle \{a\}\,,\ \{\emptyset\}\rangle$. Thus each literal a or $not\ a$ in a normal logic program can be written as a SC atom and hence each normal logic program can be considered as SC logic program. However such a translation makes normal logic programs harder to read. Thus, in what follows, we shall write a for the SC atom $\langle \{a\}\,,\ \{\{a\}\}\rangle$ and $not\ a$ for a SC atom $\langle \{a\}\,, \{\emptyset\}\rangle$.

Given an SC atom $\langle X, F\rangle$, the upper-closure $\overline{F}$ of F with respect to X is the family $\overline{F} = \{Y \subseteq X |\ \exists Z\ (Z \in F \wedge Z \subseteq Y)\}$. A family F of subsets of X is *closed* if $\overline{F} = F$. Notice that the closure of a closed family F of subsets of X is F itself. The *closure* of a SC atom $\langle X, F\rangle$ is $\langle X, \overline{F}\rangle$.

A *Horn SC rule* is a SC rule where the head of the rule is an ordinary atom and all SC atoms in the body are closed, i.e. a rule of the form

$$p \leftarrow \langle X, F_1\rangle, ..., \langle X, F_n\rangle$$

where for $i = 1, 2, ..., n$, $F_i = \overline{F_i}$. The reason for calling such a rule Horn is that if M satisfies a Horn SC rule C, then all the supersets of M will satisfy C. A *Horn SC logic program* (Horn SC program for short) is a SC program consisting entirely of Horn SC rules. Given a Horn SC program P, we define the one-step provability operator T_P by letting $T_P(M)$ equal to the set of all p such that there exists a rule $p \leftarrow \langle X, F_1\rangle$,...,$\langle X, F_n\rangle$ in P where $M \models \langle X, F_i\rangle$ for $i = 1, \ldots, n$. It is easy to prove that the one-step provability operator associated with a Horn SC program is monotone and hence a Horn SC program P has a least fixed point which is the smallest model of P.

The notion of SC stable model of a SC logic program is defined using a modification of the Gelfond-Lifschitz transform called NSS transform. That is, let P be a SC program and let M be a subset of atoms. The *NSS transform, $NSS\,(P, M)$ of P with respect to M* is defined in two steps. First, eliminate from P all rules whose bodies are not satisfied by M. In the second step, for each remaining rule $\langle X, F\rangle \leftarrow \langle X_1, F_1\rangle$, ..., $\langle X_k, F_k\rangle$ and for each $a \in X \cap M$ generate the rule $a \leftarrow \langle X_1, \overline{F_1}\rangle$,..., $\langle X_k, \overline{F_k}\rangle$. The resulting program $NSS\,(P, M)$ is a Horn SC program. Consequently, $NSS(P, M)$ has a least model $N_{P,M}$. M is called a *SC stable model* of P if M is a model of P and $M = N_{P,M}$.

Marek and Remmel in (Marek and Remmel 2004) proved the following proposition showing the equivalence of a normal logic program and its representation as SC program.

Proposition 1. Let P be a normal logic program and let M be a set of atoms. Then M is a stable model of P in the sense of Gelfond and Lifschitz if and only if M is a stable model of P viewed as SC program.

Preference Set Constraints in Logic Programs

In this section, we will define two types of preference set constraint atoms, measure PSC atoms and pre-ordered PSC atoms, and two types of preference set constraint programs, measure PSC programs and pre-ordered PSC programs which are extensions of SC atoms and SC programs, respectively. We will also discuss some of the issues related to the possible implementations of PSC programming.

A *measure preference set constraint* (measure PSC) atom is a triple $\langle X, F, \rho_F\rangle$ where $\langle X, F\rangle$ is a SC atom and $\rho_F : F \rightarrow [-\infty, \infty]$ is a *measure function*. The SC reduct of $\langle X, F, \rho_F\rangle$, $red(\langle X, F, \rho_F\rangle)$, is just the SC atom $\langle X, F\rangle$. A *pre-ordered preference set constraint* (pre-ordered PSC) atom is a triple $\langle X, F, \leq_F\rangle$ where $\langle X, F\rangle$ is a SC atom and $\leq_F$ is a pre-order on F. The SC reduct of $\langle X, F, \leq_F\rangle$, $red(\langle X, F, \leq_F\rangle)$, is just the SC atom $\langle X, F\rangle$. If M is a set atoms, we say that M is a model of $\langle X, F, \rho_F\rangle$ if $M \models \langle X, F\rangle$ and M is a model of $\langle X, F, \leq_F\rangle$ if $M \models \langle X, F\rangle$. If $\langle X, F\rangle$ is a SC atom, then we let $red(\langle X, F\rangle) = \langle X, F\rangle$.

Given a measure PSC atom, a pre-ordered PSC atom or a SC atom as above, an *elementary PSC operation* is one of the following: computing $M \cap X$, determining if $M \cap X \in F$, computing $\rho_F\,(M \cap X)$, determining whether $M_1 \cap X \leq_F M_2 \cap X$ holds.

A measure PSC rule C is a rule of the form

$$s \leftarrow s_1, \ldots, s_k$$

where $s_1, \ldots, s_k$ are SC atoms and s is either a SC atom or a measure PSC atom. We define the SC reduct of C, denoted by $red(C)$, to be the rule $red(s) \leftarrow s_1, \ldots, s_k$. A measure PSC program P is a set of measure PSC rules and we define the SC reduct of P to be the set of $red(C)$ such that C is in P. We say that a set of atoms M is a model of P if and only if M is a model of $red(P)$ and M is a stable model of P if and only if it is a stable model of $red(P)$.

Similarly, a pre-ordered PSC rule C is a rule of the form $s \leftarrow s_1, \ldots, s_k$ where $s_1, \ldots, s_k$ are SC atoms and s is either a SC atom or a pre-ordered PSC atom. We define the SC reduct of C, denoted by $red(C)$, to be the rule $red(s) \leftarrow s_1, \ldots, s_k$. A pre-ordered PSC program P is a set of pre-ordered PSC rules and we define the SC reduct of P to be the set of $red(C)$ such that C is in P. Again we say that a set of atoms M is a model of P if and only if M is a model of $red(P)$ and M is a stable model of P if and only if it is a stable model of $red(P)$.

Our idea is that if we are given a measure PSC program or a pre-ordered PSC program P, then the preference set constraint atoms can be used to induce a pre-order on the set of stable models of P. Before we can talk about this induced pre-order, we need to define a pre-order on the models of a set of PSC atoms T. That is, suppose that T is a set of pre-ordered PSC atoms and we are given two sets of atoms M_1 and M_2 which satisfy every element of T. Then we say that M_1 *is preferred to* M_2 *relative to* T, written $T \models M_1 \prec M_2$, if for all $\langle X, F, \leq_F\rangle \in T$, $M_1 \cap X \leq_F M_2 \cap X$ and there is at least one $\langle X, F, \leq_F\rangle \in T$ such that $M_1 \cap X <_F M_2 \cap X$. (Here, as usual for two sets A and B and a pre-order $\leq$, $A < B$ denotes $A \leq B$ and $B \not\leq A$). We say that M_1 *is equivalent to* M_2 *relative to* T, written $T \models M_1 \sim M_2$,

if for all $\langle X, F, \leq_F \rangle \in T$, $M_1 \cap X \leq_F M_2 \cap X$ and $M_2 \cap X \leq_F M_1 \cap X$. Hence, our pre-order on the models of T is essentially a product order over the set of local preference orders induced by each of pre-ordered PSC atoms in T. We say that M_1 *is indistinguishable from* M_2 *relative to* T, written $T \models M_1 \approx M_2$, if $T \not\models M_1 \prec M_2$ and $T \not\models M_2 \prec M_1$.

A slightly weaker type of pre-order on models can be induced by measure PSC atoms. That is, given a set T of measure PSC atoms and two sets of atoms M_1 and M_2 which are models of T, we say that M_1 *is weakly preferred to* M_2 *relative to* T, written $T \models M_1 \prec_w M_2$ if

$$\sum_{\langle X,F,\rho_F\rangle \in T} \rho_F(M_1 \cap X) < \sum_{\langle X,F,\rho_F\rangle \in T} \rho_F(M_2 \cap X). \quad (1)$$

Note that for M_1 to be weakly preferred to M_2 relative to T, we do not require that for every $\langle X, F, \rho_F \rangle \in T$, $\rho_F(M_1 \cap X) \leq \rho_F(M_2 \cap X)$, but only in the aggregate M_1 is preferred to M_2. This type of pre-order induced on models of sets of measure preference atoms allows the user more flexibility in specifying preferences. This is because one is allowed to weigh local preferences so that the weight coming from the ρ_F associated with the measure PSC atom $\langle X, F, \rho_F \rangle$ makes a much bigger contribution to (1) than the weight coming from the ρ_G associated with the measure PSC atom $\langle Y, G, \rho_G \rangle$. Thus it is possible to make sure that the local preferences specified by $\langle X, F, \rho_F \rangle$ are much more important than the local preferences specified by $\langle Y, G, \rho_G \rangle$. We say that M_1 *is indistinguishable from* M_2 *relative to* T, written $T \models M_1 \approx_w M_2$,

$$\sum_{\langle X,F,\rho_F\rangle \in T} \rho_F(M_1 \cap X) = \sum_{\langle X,F,\rho_F\rangle \in T} \rho_F(M_2 \cap X).$$

We are now in position to define how we can use PSC programs to specify preferences on stable models. We will start out considering what we call *simple PSC programs*. A simple pre-ordered PSC program is a pre-ordered PSC program P which consists of two types of rules:

$$C_1 = s \leftarrow s_1, \ldots, s_k$$

where $s, s_1, \ldots, s_k$ are SC atoms and

$$C_2 = s \leftarrow \quad (2)$$

where s is a pre-ordered PSC atom. Given a simple pre-ordered PSC program P, we let $pref\,(P)$ denote the set of pre-ordered PSC atoms that appear in a rule of type (2) in P. Note that any stable model M of P must satisfy all the pre-ordered PSC atoms in $pref(P)$. Given two stable models M_1 and M_2 of P, we say that M_1 *is preferred to* M_2 *relative to* P, written $P \models M_1 \prec M_2$, if $pref(P) \models M_1 \prec M_2$.

Similarly, a simple measure PSC program is a measure PSC program P which consists of two types of rules:

$$D_1 = s \leftarrow s_1, \ldots, s_k$$

where $s, s_1, \ldots, s_k$ are SC atoms and

$$D_2 = s \leftarrow \quad (3)$$

where s is a measure PSC atom. Given a simple measure PSC program P, we let $pref\,(P)$ denote the set of measure PSC atoms that appear in a rule of type (3) in P. Note that any stable model M of P must satisfy all the measure PSC atoms in $pref(P)$. Given two stable models M_1 and M_2 of P, M_1 *is weakly preferred to* M_2 *relative to* P, written $P \models M_1 \prec_w M_2$, if $pref(P) \models M_1 \prec_w M_2$.

To be practical, any implementation of PSC programming will have to be restricted to a class of programs that can be represented in a compact way. The idea is that we should think of an SC atom $\langle X, F \rangle$ as being implemented by an intersection algorithm A_X for X and a membership algorithm A_F for F. That is, for any set of atoms M, the algorithm A_X returns a representation for $M \cap X$ as a word w in $\{0,1\}^{|X|}$ and then $A_F(w) = 1$ if $M \cap X \in F$ and $A_F(w) = 0$ if $M \cap X \notin F$. For a pre-ordered PSC atom $\langle X, F, \leq_F \rangle$, we assume that there is an additional algorithm $A_\leq$ which for any pair (M_1, M_2) of sets of atoms takes as input (w_1, w_2) where w_1 is the representation of $M_1 \cap X$ produced by $A_X(M_1)$ and w_2 is the representation of $M_2 \cap X$ produced by $A_X(M_2)$ and returns 1 if $M_1 \cap X$ and $M_2 \cap X$ are in F and $M_1 \cap X \leq_F M_2 \cap X$ and returns 0 otherwise. For a measure PSC atom $\langle X, F, \rho_F \rangle$, we assume that there is an addition algorithm A_{ρ_F} which for any set of atoms M, takes as an input the representation w of $M \cap X$ produced by $A_X(M)$ and returns $\rho_F(M_1 \cap X)$ if $M_1 \cap X \in F$ and returns $*$ otherwise. For example, suppose that for a SC atom $\langle X, F \rangle$, F contains only those subsets of X that consist of an even number of elements. Implementation of this atom by enumerating F is clearly inefficient as there are $2^{|X|-1}$ sets with even number of elements in F. Thus we do not want to represent F by an enumeration of all the subsets of X of even cardinality. However, an algorithm returning 1 if and only if the input set has even number of elements is trivial to implement and will run in $O\,(|X|)$ time. An obvious implementation can be obtained as follows. Let $\{x_1, ..., x_n\}$ be an enumeration of X. Thus any subset of X can be represented as a vector of n binary digits (bits). For instance, suppose that $X = \{x_1, ..., x_8\}$. Consider $Y \subseteq X$ where $Y = \{x_1, x_3, x_6\}$. Then an 8 bit representation of A is 101001. Assuming this representation we have the following pseudocode for A_F.

```
boolean AF (Y)
  n = size(Y);
  result = true;
  for i=1:n if Y[i] == 1 then result = ˜result; endfor;
 return result;
end
```

where Y[i] is the ith bit of the bit vector Y, and ˜result is the boolean NOT operation. Suppose instead of the subsets of X of even cardinality, we want to enforce the cardinality constraint on the subsets of Y of X such that $3 \leq |Y| \leq |X|-3$. There are $2^{|X|} - 2(|X| \cdot (|X|-1)/2 + |X| + 1)$ such subsets and we clearly do not want to list all of them. However, as in the previous example there is a simple algorithm that will implement this cardinality constraint by counting the number of bits N of Y that have value equal to 1 and checking the condition $3 \leq N \leq |X| - 3$.

PSC semantics specifies what an implementation of PSC programming should do. It is not a prescription for how PSC programming should be implemented. Thus the fact that PSC programming deals with sets should not be understood to mean that PSC programming has to be implemented by enumerating sets. Efficient implementations of PSC programming will not use such an approach. We will use two examples to motivate the fact that semantics does not necessarily prescribe how the formalism is to be implemented. We note that an extension of ASP that uses arbitrary algorithms was considered in (Brik and Remmel 2011).

First, consider stable model semantics. The formalism shows that a stable model can be found by choosing a subset of the Herbrand base of a logic program and then checking that the subset is a stable model. Now, while undoubtedly useful, the formalism is impractical if implemented as stated. Indeed, modern ASP solvers such as *smodels* (Simons, Niemelä, and Soininen 2002) and *clasp* (Gebser et al. 2007) use efficient algorithms to implement stable model semantics, where the algorithms do not rely on searching through the entire powerset of the Herbrand base of a logic program, which would be the case if the semantics was considered as a prescription for implementations.

Second, consider cardinality constraint programming (Niemelä, Simons, and Soininen 1999). The formalism has been implemented in *smodels-2* (Simons 1999) and is generally considered to be a practical extension of ASP. However, its practicality follows from the fact that efficient implementations of cardinality constraint programming exist.

To illustrate how the algorithmic approach for implementing PSC programming might work we will consider the problem of finding a vertex cover of size less than K for a given graph, with a preference for the covers that include vertex w.

For a given graph a measure PSC program can be constructed as follows. For every edge (u, v) include a rule$\langle \{u, v\}, \{\{u\}, \{v\}\}\rangle \leftarrow$, specifying that either u or v is in a stable model. Also include the rule $\langle V, F, \rho_F\rangle \leftarrow$ where V is the set of all the vertices of the graph and, for any $U \subseteq V$, $U \in F$ if $|U| < K$ and

$$\rho_F(U) = \begin{cases} 0 \text{ if } w \in U \\ 1 \text{ if } w \notin U. \end{cases}$$

Now, an algorithmic implementation of V is very simple. Namely, for any subset U of the Herbrand base $A_V(U)$ returns a representation of U as a word b_U in $\{0,1\}^{|V|}$. Then the membership algorithm for F, A_F, can easily check whether $|U| \leq K$ by making one pass through b_U. Finally the algorithm $A_{\rho_F}(b_U)$ simply has to check the condition $b_U[i_w] == 1$ where i_w is the index in the bit array corresponding to the atom w. This example shows that compact representations of PSC programs can be created and that such representations allow one to efficiently implement PSC programming in many cases.

For many applications, simple PSC programs are adequate to express preferences. In fact, pre-ordered simple PSC programs are sufficient to express optimal stable models of ASO.

We will now give some examples which illustrate how PSC programming can be used.

Example 1. Bob is a Ph.D. student who is about to graduate from his university. Bob is guessing that he will have multiple job offers and wants to determine a method by which he will make his decision. Bob identifies two important criteria in making a decision. These are the type of institution and its location. He thus introduces the following atoms: R - for the job at a research university, T - for the job at a teaching university, C - for the job in a company, CAL-for the job located in California, and NCal-for the job not located in California. Thus the set of atoms is $X = \{R, T, C, CAL, NCAL\}$. Then any offer is described by a set of atoms from the following family of sets:

$$F = \{\{A, B\} : A \in \{R, T, C\} \text{ and } B \in \{CAL, NCAL\}\}.$$

Finally Bob decides on the following ordering of sets $\leq_F$: $\{R, CAL\} <_F \{R, NCAL\} <_F \{T, CAL\} <_F \{T, NCAL\} <_F \{C, CAL\} <_F \{C, NCAL\}$.

In this example and all the following examples in this section, we will assume that Bob is trying to decide between two jobs j_1 and j_2. Thus we introduce a base program P_0. P_0 contains the rule

$$\langle \{j_1, j_2\}, \{\{j_1\}, \{j_2\}\}\rangle \leftarrow .$$

This rule says that, in any stable model M, exactly one of j_1 and j_2 is contained. Then we add rules to specify the relevant information about jobs j_1 and j_2. For example, if j_1 is a job at a research university in California and j_2 is a job at a teaching university outside of California, we would add the following rules: $R \leftarrow j_1$, $Cal \leftarrow j_1, T \leftarrow j_2$, $NCal \leftarrow j_2$.

Then Bob's preferences can be described by the simple pre-ordered PSC program P_1 which consists of P_0 plus the rule $\langle X, F, \leq_F\rangle \leftarrow$.

Example 2. Bob soon realizes that not all locations outside of California have the same weight. He is actually more likely to consider an offer from a location which is near California than from a location which is far from California. Bob thinks that a job at a research university is preferable to a job at a teaching university and that a job at a teaching university is preferable to a job at a company. Yet, Bob notices that he will prefer a job from a teaching university in California to a job from a research university which is more than 500 miles away from California.

He thus revises his original approach. There are still predicate atoms R, T, C to specify a research university, or a teaching university or a company respectively. However, now Bob introduces a set of predicate atoms $\widehat{D} = \{D(x) | x \in \mathbb{N}\}$, where $\mathbb{N}$ is the set of natural numbers and $D(x)$ indicates a distance x from California. Thus $D(0)$ indicates that the location of the job is in California. Note that in an implementation of PSC programming $\widehat{D}$ can be an algorithm that on an input M will simply return the set of atoms in M of the form $D(x)$ for $x \in \mathbb{N}$. While $\widehat{D}$ represents an infinite set, its implementation can be compact and efficient.

Now let $Z = \{R, T, C\} \cup \widehat{D}$ and let $H = \{\{A, B\} | A \in \{R, T, C\}, B \in \widehat{D}\}$. Bob defines a measure function as follows $\rho_H(\{A, D(x)\}) = \tau(A) + x$ where $\tau(R) = 0$, $\tau(T) = 500$ and $\tau(C) = 1000$.

Thus Bob's preferences are specified by the simple measure PSC program which consists of P_0 plus the rule $\langle Z, H, \rho_H \rangle \leftarrow$.

If we consider more general pre-ordered PSC programs P and measure PSC programs, then we have several natural choices for how to induce a pre-order on the set of stable models of P. If P is a pre-ordered (measure) PSC program and M is a stable model of P, then we let $pref(P, M)$ denote the set of all pre-ordered (measure) PSC atoms s such that there is a rule $C = s \leftarrow s_1, \ldots, s_k$ where s is a pre-ordered (measure) PSC atom and M satisfies the body of C. Since all stable models of P are models of P by definition, M must be a model of $pref(P, M)$.

Note, however, that if M_1 and M_2 are stable models of P it is not necessarily the case that $pref(P, M_1) = pref(P, M_2)$. Now in the case where $pref(P, M_1) = pref(P, M_2)$, the obvious thing to do for the pre-ordered PSC programs is to say that M_1 is preferred to M_2 relative to P if and only if $pref(P, M_1) \models M_1 \prec M_2$. However, if $pref(P, M_1) \neq pref(P, M_2)$, then one has several natural choices. First, one can simply consider the pre-ordered PSC atoms in $pref(P, M_1) \cap pref(P, M_2)$, i.e. the pre-ordered PSC atoms which are the conclusions of rules of P which are satisfied by both M_1 and M_2. Thus we say that M_1 is *in common preferred* to M_2 relative to P, written $P \models M_1 \prec_{ic} M_2$, if and only if $pref(P, M_1) \cap pref(P, M_2) \models M_1 \prec M_2$. A second natural choice that one might want to use in certain situations is to take the point of view that satisfying a pre-ordered PSC atom s that appears in the head of a rule in P is more preferable than not satisfying s. Thus we say that M_1 is *in total preferred* to M_2 relative to P, written $P \models M_1 \prec_{it} M_2$, if and only if either (a) $pref(P, M_1) \supset pref(P, M_2)$ and either $pref(P, M_1) \cap pref(P, M_2) \models M_1 \prec M_2$ or $pref(P, M_1) \cap pref(P, M_2) \models M_1 \sim M_2$ or (b) $pref(P, M_1) = pref(P, M_2)$ and $pref(P, M_1) \cap pref(P, M_2) \models M_1 \prec M_2$.

Example 3. As Bob has more time to contemplate the job offers, he realizes that his life will be simplified if the job is in a town where there is a good public transportation system. Also, being a classical music lover, Bob considers an easy access to live classical concerts as one of the factors in making his decision. Not being sure about the weight that public transportation and live classical music concerts should have in the decision making process, he reverts to the pre-ordered PSC program from Example 1. Bob reasons that he will keep his preferences as they already are, except when there is an access to live classical music concerts in the area. In that case, a location outside of California with a good system of public transportation is preferable to a location in California without such a system. Thus Bob introduces two new predicate atoms: CM - to indicate the presence of local access to live classical music concerts and PT to indicate the presence of a good system of public transportation. Bob adds a new rule to the program P_1 to produce a new program P_3. P_3 consists of P_0 plus the following two rules:

$$\langle X, F, \leq_F \rangle \leftarrow \quad \langle Y, G, \leq_G \rangle \leftarrow CM$$

where $Y = \{CAL, NCAL, PT\}$, $G = \{\{CAL, PT\}, \{NCAL, PT\}, \{CAL\}, \{NCAL\}\}$ and $\{CAL, PT\} <_G \{NCAL, PT\} <_G \{CAL\} <_G \{NCAL\}$.

Now $P_3 \models M_2 \prec_{ic} M_1$ since $pref(P_3, M_1) \cap pref(P_3, M_2) = \{\langle X, F, \leq_F \rangle\}$ and $\langle X, F, \leq_F \rangle \models M_2 \prec M_1$. However $P_3 \models M_1 \approx_{it} M_2$ because $pref(P_3, M_1) \supset pref(P_3, M_2)$ and $\langle X, F, \leq_F \rangle \models M_2 \prec M_1$.

Similarly we can define a pre-order on the set of stable models of measure PSC programs. In the case where $pref(P, M_1) = pref(P, M_2)$, the obvious thing to do is to say that M_1 is weakly preferred to M_2 relative to P if and only if $pref(P, M_1) \models M_1 \prec_w M_2$. However, if $pref(P, M_1) \neq pref(P, M_2)$, then one has several natural choices. One is to simply consider the measure PSC atoms in $pref(P, M_1) \cap pref(P, M_2)$, i.e. the measure PSC atoms which are the conclusions of rules of P which are satisfied by both M_1 and M_2. Thus we say that M_1 is *in common weakly preferred* to M_2 relative to P, written $P \models M_1 \preceq_{w,ic} M_2$, if and only if $pref(P, M_1) \cap pref(P, M_2) \models M_1 \prec_w M_2$. As before, our second natural choice is to take the point of view that satisfying a measure PSC atom s that appears in the head of a rule in P is more preferable than not satisfying s. Thus we say that M_1 is *in total weakly preferred* to M_2 relative to P, written $P \models M_1 \prec_{w,it} M_2$, if and only if (a) $pref(P, M_1) \supset pref(P, M_2)$ and either $pref(P, M_1) \cap pref(P, M_2) \models M_1 \prec_w M_2$ or $pref(P, M_1) \cap pref(P, M_2) \models M_1 \approx_w M_2$ or (b) $pref(P, M_1) = pref(P, M_2)$ and $pref(P, M_1) \cap pref(P, M_2) \models M_1 \prec_w M_2$. We also have a third natural choice that one might want to use in certain circumstances which is just to compare the two sums $\sum_{\langle X,F,\rho_F \rangle \in pref(P,M_1)} \rho_F(M_1 \cap X)$ and $\sum_{\langle X,F,\rho_F \rangle \in pref(P,M_2)} \rho_F(M_2 \cap X)$. Thus we say that M_1 is *in sum weakly preferred* to M_2 relative to P, written $P \models M_1 \prec_{w,is} M_2$, if

$$\sum_{\langle X,F,\rho_F \rangle \in pref(P,M_1)} \rho_F(M_1 \cap X) <$$

$$\sum_{\langle X,F,\rho_F \rangle \in pref(P,M_2)} \rho_F(M_2 \cap X).$$

Definition 1. *A set of atoms M is called an in common preferred PSC stable model of a pre-ordered PSC program P if M is a PSC stable model of P and for all PSC stable models M' of P, $P \not\models M' \prec_{ic} M$.*

An *in total preferred PSC stable model*, an *in common weakly preferred PSC stable model*, an *in total weakly preferred PSC stable model*, an *in sum weakly preferred PSC stable model* are defined similarly. To refer to any of these definitions without explicitly naming them we may say that *M is a preferred PSC stable model of a PSC program P.*

PSC Programs and Answer Set Optimization Programs

As was stated in the introduction, the closest approach to PSC programming is ASO. In ASO, one starts with an ASP program P_{gen} over a set of atoms At and then a preference specification is given by a separate preference program P_{pref}. The rules of P_{pref} are of the form

$$C_1 > ... > C_k \leftarrow a_1, ..., a_n, not\ b_1, ..., not\ b_m \quad (4)$$

where a_is and b_js are literals and the C_is are Boolean combinations over At. Here a Boolean combination over At is a formula built of atoms in A by means of disjunction, conjunction, strong negation $\neg$ and default negation not, with the restriction that strong negation is allowed to appear only in front of atoms, and default negation is allowed to appear only in front of literals.

Next suppose that we are given a set of literals S. Then the definition of S satisfying a Boolean combination C, written $S \models C$, uses the standard inductive definition of satisfaction of propositional formulas except that $S \models not\ l$ where l is literal if and only if $l \notin S$. Then we define the satisfaction degree $v_S(r)$ for any rule of the form of (4) by setting (i) $v_S(r) = I$ if either the body of r is not satisfied by S or the body of r is satisfied by S, but none of the C_is are satisfied and (ii) $v_S(r) = \min\{i : S \models C_i\}$ if the body of r is satisfied by S and at least one of the C_is is satisfied by S. This allows one to define a satisfaction vector $V_S = (v_S(r_1), ..., v_S(r_n))$ for any answer set S of P_{gen} for a preference program $P_{pref} = \{r_1, ..., r_n\}$.

One can then use satisfaction vectors to define a pre-order on answer sets of P_{gen} as follows. First for any two possible values a and b of $v_S(r)$ (i) $a \geq b$ if $a = I$ and $b = 1$ or if $a = 1$ and $b = I$, (ii) $a > b$ if $a = I$ and $b \in \{2, 3, ...\}$, and (iii) $a > b$ if $a, b \in \{1, 2, ...\}$ and $a < b$ relative to the usual order on the natural numbers. Then for two sets of literals S_1 and S_2 $V_{S_1} \geq V_{S_2}$ if $v_{S_1}(r_i) \geq v_{S_2}(r_i)$ for every $i \in \{1, ..., n\}$ $V_{S_1} > V_{S_2}$ if $V_{S_1} \geq V_{S_2}$ and for some $i \in \{1, ..., n\}$ $v_{S_1}(r_i) > v_{S_2}(r_i)$. $S_1 \geq S_2$ if $V_{S_1} \geq V_{S_2}$ and $S_1 > S_2$ if $V_{S_1} > V_{S_2}$. Finally, a set of literals S is an *optimal model of an ASO program* (P_{gen}, P_{pref}) if S is an answer set of P_{gen} and there is no answer set S' of P_{gen} such that $S' > S$.

We will now show how optimal models of an ASO program can be expressed using PSC programming. Let (P_{gen}, P_{pref}) be an ASO program.

Let At be the set of all atoms that occur in the rules of P_{gen} and P_{pref}. Note that any stable model A of P_{gen} must be a subset of At. Consequently, any optimal model of (P_{gen}, P_{pref}) must be a subset of At.

Let P_{pref} consist of the rules W_1, W_2, ..., W_γ where W_i is of the form $C_1^i > ... > C_k^i \leftarrow a_1^i, ..., a_{n_i}^i, not\ b_1^i, ..., not\ b_{m_i}^i$.

We will now construct a simple pre-ordered PSC program P and we will show that there is a one-to-one correspondence between the preferred PSC stable models of P and the optimal stable models of (P_{gen}, P_{pref}).

The Herbrand base $H(P)$ of P will consist of At plus the set $\overline{At} = \{\overline{a} |\ a \in A\}$ where for each $a \in At$, $\overline{a}$ is a new atom not in At and where, for all $a, b \in At$, if $a \neq b$, then $\overline{a} \neq \overline{b}$, and a new atom D so that $D \notin At \cup \overline{At}$. That is $H(P) = At \cup \overline{At} \cup \{D\}$.

For a set $A \subseteq At$ define $\overline{A} = \{\overline{a} |\ a \in A\}$. For a rule $W_i \in P_{pref}$ let $At(W_i)$ be the set of all the atoms that occur in W_i. We define a PSC program P'_{pref} to be the set of all rules

$$\left\langle \overline{At(W_i)},\ \mathcal{P}\left(\overline{At(W_i)}\right),\ \leq_i \right\rangle \leftarrow .$$

The pre-order $\leq_i$ is defined as follows. For $A \subseteq At$, $B \subseteq At$ $\overline{A} \leq_i \overline{B}$ if one of the following conditions hold.

1. $A \not\models body(W_i)$; 2. $A \models body(W_i)$ and $A \not\models c(W_i)$; 3. $A \models body(W_i)$ and $A \models C_1^i$; 4. $A \models body(W_i)$ and $A \models C_z^i$ where z is minimal and $B \models body(W_i)$ and $B \models C_j^i$ where j is minimal and $z \leq j$, i.e., $\overline{A} \leq_i \overline{B}$ iff $v_A(W_i) \geq v_B(W_i)$.

Let P_{cons} be the set of rules consisting of two rules for each $a \in At$ $\overline{a} \leftarrow a$ and $D \leftarrow \overline{a}, not\ a, not\ D$.

Then the PSC program P is defined by $P = P_{gen} \cup P'_{pref} \cup P_{cons}$.

We can prove the following two theorems.

Theorem 1. *1. For all $B \subseteq At$, if B is a stable model of P_{gen} in the sense of Gelfond and Lifschitz, then $B \cup \overline{B}$ is a stable model of P in the set of PSC.*

2. For all $A \subseteq H(P)$, if A is a stable model of P in the sense of PSC, then $A = B \cup \overline{B}$ where B is a stable model of P_{gen} in the sense of Gelfond and Lifschitz.

Theorem 2. *1. If B is an optimal stable model of (P_{gen}, P_{pref}), then $B \cup \overline{B}$ is a preferred PSC stable model of P.*

2. If A is a preferred PSC stable model of P, then $A \cap At$ is an optimal stable model of (P_{gen}, P_{pref}).

A reasonable concern in the construction of P is the efficiency of the implementation of the PSC atoms $\left\langle \overline{At(W_i)},\ \mathcal{P}\left(\overline{At(W_i)}\right),\ \leq_i \right\rangle$.The implementation of such an atom can be a string $\langle X, F, O\rangle$ where X, F, O are the names of the algorithms. The algorithm corresponding to X on the input set of atoms M will return $\overline{At(W_i)} \cap M$. The algorithm corresponding to F will always return 1 since any subset of $\overline{At(W_i)} \cap M$ is in $\mathcal{P}\left(\overline{At(W_i)}\right)$. The algorithm corresponding to O on the inputs $\overline{A}$ and $\overline{B}$ will return 1 iff $\overline{A} \leq_i \overline{B}$ and 0 otherwise. The algorithm for O can be as efficient as the algorithm in the implementation of ASO. Our algorithm can use the ASO algorithm to check the conditions $A \models body(W_i)$, $A \models c(W_i)$, $B \models body(W_i)$, $A \models C_z^i$, $B \models C_z^i$ that are necessary for its evaluation.

Using PSC Programs to Express General Preferences

In this section we will show that the general preferences of (Son and Pontelli 2006) can be expressed using PSC atoms. In (Son and Pontelli 2006), the language $\mathcal{PP}$ for planning preferences specification was introduced. $\mathcal{PP}$ allows users to elegantly express multi-dimensional preferences among plans that achieve the same goal. In $\mathcal{PP}$, users can define

general preferences by building them from simpler *atomic preferences* using a small number of special operators.

A *basic desire formula* (basic desire for short) is a formula expressing a single preference about a trajectory. Son and Pontelli provide a formal definition of the notion of *basic desire*, as well as a formal definition of the notion of a *trajectory*, and define what it means for a trajectory *to satisfy* a basic desire. For the purposes of this paper, it is not necessary to restate these definitions. It will suffice to assume that a basic desire is a formula ϕ in some language and that, for any trajectory α, we can determine whether α *satisfies* ϕ, written $\alpha \models \phi$, or whether α *does not satisfy* ϕ, written $\alpha \not\models \phi$. We will also assume that each basic desire corresponds to a unique predicate atom. In addition, we will use the basic desires and the predicate atoms corresponding to them interchangeably.

Let ϕ be a basic desire formula and let α and β be two trajectories. The trajectory α is *preferred* to the trajectory β, written $\alpha \prec_\phi \beta$, if $\alpha \models \phi$ and $\beta \not\models \phi$. α and β are *indistinguishable with respect to* ϕ, written $\alpha \approx_\phi \beta$, if either (1) $\alpha \models \phi$ and $\beta \models \phi$ or (2) $\alpha \not\models \phi$ and $\beta \not\models \phi$.

An *atomic preference formula* is defined as a formula of the type $\phi_1 \lhd \phi_2 \lhd ... \lhd \phi_n$ where ϕ_1, ..., ϕ_n are basic desire formulas. If α and β are trajectories and $\psi = \phi_1 \lhd \phi_2 \lhd ... \lhd \phi_n$ is an atomic preference formula, then we say α and β are *indistinguishable with respect to* ψ (written as $\alpha \approx_\psi \beta$) if $\forall i\ (1 \leq i \leq n \Rightarrow \alpha \approx_{\phi_i} \beta)$ and α is *preferred to* β with respect to ψ (written as $\alpha \prec_\psi \beta$) if $\exists\, (1 \leq i \leq n)$ such that $\forall\, (1 \leq j < i)\ \alpha \approx_{\phi_j} \beta$ and $\alpha \prec_{\phi_i} \beta$.

The set of *general preference formulas* (general preferences) are defined via the following inductive definition: (i) every atomic preference formula ψ is a general preference formula, (ii) if ψ_1, ψ_2 are general preference formulas, then $\psi_1 \& \psi_2$, $\psi_1 | \psi_2$, and $!\psi_1$ are general preference formulas, and (iii) if ψ_1, ψ_2, ..., ψ_k are general preference formulas, then $\psi_1 \lhd \psi_2 \lhd \ldots \lhd \psi_k$ is a general preference formula.

Let ψ be a general preference formula and let α, β be two trajectories. Then we say α is *preferred to* β *with respect to* ψ (written $\alpha \prec_\psi \beta$) if:

1. ψ is an atomic preference formula and $\alpha \prec_\psi \beta$
2. $\psi = \psi_1 \& \psi_2$ and $\alpha \prec_{\psi_1} \beta$ and $\alpha \prec_{\psi_2} \beta$
3. $\psi = \psi_1 | \psi_2$ and ($\alpha \prec_{\psi_1} \beta$ and $\alpha \approx_{\psi_2} \beta$) or ($\alpha \approx_{\psi_1} \beta$ and $\alpha \prec_{\psi_2} \beta$) or ($\alpha \prec_{\psi_1} \beta$ and $\alpha \prec_{\psi_2} \beta$).
4. $\psi = !\psi_1$ and $\beta \prec_{\psi_1} \alpha$
5. $\psi = \psi_1 \lhd \psi_2 \lhd \ldots \lhd \psi_k$, and there exists $1 \leq i \leq k$ such that ($\forall\, (1 \leq j < i)\ \alpha \approx_{\psi_j} \beta$ and $\alpha \prec_{\psi_i} \beta$)

We say α is *indistinguishable from* β *with respect to* ψ (write $\alpha \approx_\psi \beta$) if:

1. ψ is an atomic preference formula and $\alpha \approx_\psi \beta$
2. $\psi = \psi_1 \& \psi_2$, $\alpha \approx_{\psi_1} \beta$ and $\alpha \approx_{\psi_2} \beta$
3. $\psi = \psi_1 | \psi_2$, $\alpha \approx_{\psi_1} \beta$ and $\alpha \approx_{\psi_2} \beta$
4. $\psi = !\psi_1$ and $\alpha \approx_{\psi_1} \beta$
5. $\psi = \psi_1 \lhd \psi_2 \lhd \ldots \lhd \psi_k$, and for all $1 \leq i \leq k$ $\alpha \approx_{\psi_i} \beta$.

Let Δ be the set of all the basic desire predicate atoms. Let X be a finite subset of Δ and $F \subseteq P(X)$. Let $\leq_F$ be a preorder on F. For a trajectory α let $\Delta(\alpha) \equiv \{d \in \Delta |\ \alpha \models d\}$ i.e. the set of all the basic desires satisfied by α. Then we say that α *satisfies* $\langle X, F, \leq_F \rangle$ if $\Delta(\alpha) \cap X \in F$. We say that a trajectory α *is preferred to a trajectory* β *with respect to* $\langle X, F, \leq_F \rangle$ if $\langle X, F, \leq_F \rangle \models \Delta(\alpha) \prec \Delta(\beta)$. This will be denoted by $\langle X, F, \leq_F \rangle \models \alpha \prec \beta$. We say that trajectories α, β *are indistinguishable with respect to* $\langle X, F, \leq_F \rangle$ if $\langle X, F, \leq_F \rangle \models \Delta(\alpha) \approx \Delta(\beta)$. This will be denoted by $\langle X, F, \leq_F \rangle \models \alpha \approx \beta$.

Theorem 3. *For a general preference formula ϕ there exists a pre-ordered PSC atom $\langle X_\phi, F_\phi, \leq_\phi \rangle$ such that for two trajectories α, β, $\alpha \prec_\phi \beta$ in the sense of (Son and Pontelli 2006) iff $\langle X_\phi, F_\phi, \leq_\phi \rangle \models \alpha \prec \beta$ and $\alpha \approx_\phi \beta$ in the sense of (Son and Pontelli 2006) iff $\langle X_\phi, F_\phi, \leq_\phi \rangle \models \alpha \approx \beta$.*

The proof, omitted due to space limitations shows how to construct $\langle X_\phi, F_\phi, \leq_\phi \rangle$.

The Computational Complexity of the Set of Preferred PSC Stable Models

We can prove the following result on the computational complexity of finding a "preferred" PSC stable models of a PSC program. Please note that the theorem's preconditions are formulated to be useful for analyzing algorithmic implementations of PSC programming as discussed in section 3.

Theorem 4. *Let P be a finite PSC program. Let At be the set of all atoms appearing in P.* Suppose that for any subset of At *and any PSC atom or any SC atom performing an elementary PSC operation can be done in polynomial time on the number of elements in At. Then given $M \subseteq At$, the problem of determining whether M is a preferred PSC stable model of P for any of the induced pre-orders on stable models described in Section 3 is CoNP-complete relative to the size of P and the number of elements in At.*

Proof. The problem is in CoNP since M is not a preferred stable model of P if either M is not a stable model of P or there exists a stable model N of P which is preferred to M.

The completeness is demonstrated by a reduction of 3-SAT problem. Let ψ be a formula $\bigwedge_{i=1}^{n} (a_{i1} \vee a_{i2} \vee a_{i3})$, where a_{ij} are boolean variables. For $i = 1, .., n$ let $A_i = \{s_{i1}, s_{i2}, s_{i3}\}$ be a set of atoms. Consider the following measure PSC program P_ψ.

$\langle \{S\}, \{\{S\}\}, f \rangle \leftarrow \langle A_1, P(A_1) \backslash \emptyset \rangle, ..., \langle A_n, P(A_n) \backslash \emptyset \rangle$ where $f(\{S\}) = 1$. For every $i = 1, ..., n$, for $j = 1, ..., 3$ P_ψ has clauses $\langle \{s_{ij}\}, \{\{s_{ij}\}, \emptyset\} \rangle \leftarrow$

For a set of atoms M let $v_M(a_{ij}) = 1$ iff $s_{ij} \in M$. Then $S \in M$ iff v_M satisfies ψ. Pick $M \subseteq \{s_{11}, ..., s_{n3}\}$ at random. If v_M does not satisfy ψ then a satisfying assignment exists iff M is not a preferred stable model of P_ψ.■

Now, in (Brewka, Niemelä, and Truszczynski 2003), it was shown that the problem of deciding whether there exists an answer set for an ASO program $P = (P_{gen}, P_{pref})$ is NP-complete and the problem of deciding whether S is an optimal model of P is coNP-complete. This is the complexity of the corresponding problems for PSC programs. That is, the problem of deciding whether there exists a stable model for a PSC program P is NP-complete and the problem of deciding whether a PSC stable model M of a PSC program P is a preferred PSC stable model is co-NP complete.

Conclusions and directions for further research

In this paper, we have introduced an approach to specifying preferences in ASP called Preference Set Constraint Programming which is an extension of SC programming of (Marek and Remmel 2004). PSC programming uses two types of PSC atoms: pre-ordered PSC atoms and measure PSC atoms. These atoms can be used to define pre-ordered PSC programs and measure PSC programs. For pre-ordered PSC programs, we have considered two approaches for specifying preferences: "in common" and "in total". For the measure PSC programs we have considered three approaches: "in common", "in total" and "in sum". We show that the problem of determining whether M is a preferred PSC stable models of a PSC program is CoNP-complete. To demonstrate the expressive power of PSC programming, we have shown that PSC programming can be used to express optimal stable models of ASO (Brewka, Niemelä, and Truszczynski 2003), and the general preferences of Son and Pontelli (Son and Pontelli 2006). It is also the case that the preferred stable models and the weakly preferred stable models of (Brewka and Eiter 1999) can be expressed by an extension of PSC programming, although it was not discussed in this paper.

We have only briefly discussed implementations of PSC programming, but clearly it is an important issue for applications.

There are a number of areas for further research on PSC programming. One is to study the exact relationships between PSC programming and other approaches for reasoning with preferences. In particular, one must pay special attention to the efficiency of expressing preferences in any two systems that are being compared. A second is to study extensions of PSC programming where we add the ability of the user to express preference by preference orderings on rules. We will study both of these questions in subsequent papers.

Finally, the five approaches for specifying preferences using PSC programs can be viewed as special cases of the following generalization. A *PSC system* R is a pair $\langle P, \leq \rangle$ where P is a PSC program and $\leq$ is a pre-order on the set of PSC stable models of P with the property that if M_1 and M_2 are PSC stable models of P such that $pref\,(P, M_1) = pref\,(P, M_2)$ and $M_1 \leq M_2$, then $pref\,(P, M_1) \models M_1 \prec M_2$ or $pref\,(P, M_1) \models M_1 \sim M_2$ if P is a pre-ordered PSC program and $pref\,(P, M_1) \models M_1 \prec_w M_2$ or $pref\,(P, M_1) \models M_1 \approx_w M_2$ if P is a measure PSC program. We suggest that one should study abstract properties of PSC systems.

References

Brewka, G., and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artif. Intell.* 109(1-2):297–356.

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

Brewka, G.; Niemelä, I.; and Truszczynski, M. 2003. Answer set optimization. In Gottlob, G., and Walsh, T., eds., *IJCAI*, 867–872. Morgan Kaufmann.

Brewka, G.; Niemelä, I.; and Truszczynski, M. 2008. Preferences and nonmonotonic reasoning. *AI Magazine* 29(4):69–78.

Brewka, G. 2002. Logic programming with ordered disjunction. In *AAAI/IAAI*, 100–105.

Brik, A., and Remmel, J. B. 2011. Hybrid asp. In Gallagher, J. P., and Gelfond, M., eds., *ICLP (Technical Communications)*, volume 11 of *LIPIcs*, 40–50. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Buccafurri, F.; Leone, N.; and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Trans. Knowl. Data Eng.* 12(5):845–860.

Delgrande, J. P.; Schaub, T.; Tompits, H.; and Wang, K. 2004. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence* 20(2):308–334.

Elkan, C. 1990. A rational reconstruction of nonmonotonic truth maintenance systems. *Artif. Intell.* 43(2):219–234.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In Veloso, M. M., ed., *IJCAI*, 386–373.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*, 1070–1080.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Comput.* 9(3/4):365–386.

Gelfond, M.; Leone, N.; and Pfeifer, G., eds. 1999. *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, El Paso, Texas, USA, December 2-4, 1999, Proceedings*, volume 1730 of *Lecture Notes in Computer Science*. Springer.

Marek, V. W., and Remmel, J. B. 2003. On the expressibility of stable logic programming. *TPLP* 3(4-5):551–567.

Marek, V. W., and Remmel, J. B. 2004. Set constraints in logic programming. In Lifschitz, V., and Niemelä, I., eds., *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, 167–179. Springer.

Marek, V. W., and Truszczynski, M. 1991. Autoepistemic logic. *J. ACM* 38(3):588–619.

Niemelä, I., and Simons, P. 2000. Extending the smodels system with cardinality and weight constraints. *Logic-Based Artificial Intelligence* 491–521.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics of weight constraint rules. In Gelfond et al. (1999), 317–331.

Sakama, C., and Inoue, K. 2000. Prioritized logic programming and its application to commonsense reasoning. *Artif. Intell.* 123(1-2):185–222.

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

Simons, P. 1999. Extending the stable model semantics with more expressive rules. In Gelfond et al. (1999), 305–316.

Son, T. C., and Pontelli, E. 2006. Planning with preferences using logic programming. *TPLP* 6(5):559–607.