Set Constraints in Logic Programming

Victor W. Marek[1] and Jeffrey B. Remmel[2]

[1] Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA
marek@cs.uky.edu
[2] Department of Mathematics
University of California
La Jolla, CA 92093, USA
jremmel@ucsd.edu

Abstract. We investigate a generalization of weight-constraint programs with stable semantics, as implemented in the ASP solver *smodels*. Our programs admit atoms of the form $\langle X, \mathcal{F}\rangle$ where X is a finite set of propositional atoms and $\mathcal{F}$ is an arbitrary family of subsets of X. We call such atoms *set constaints* and show that the concept of stable model can be generalized to programs admitting set constraints both in the bodies and the heads of clauses. Natural tools to investigate the fixpoint semantics for such programs are *nondeterministic* operators in complete lattices. We prove two fixpoint theorems for such operators.

1 Introduction

This paper is concerned with extensions of the Answer Set Programming (ASP) paradigm [SK92,CMT96,NS97,ELM+98,KS99,MT99,NS00,ASP01,Ba03]. These extensions allow a programmer to use various type of aggregation expressions in both the head and the body of clauses of a program. Under the ASP paradigm, a problem is encoded as a program in a declarative language so that the preferred models of the program encode the solutions to the problem. A typical example of an ASP formalism is DATALOG$^\neg$ where the set of preferred models is the set of stable models of the program. In this case, one can use a solver such as *smodels* [NS00] to compute the preferred answers. In [MR02], we studied an extension of DATALOG$^\neg$ called cardinality constraint programming developed by Niemelä, Simons, and Soininen in [NSS99,NS00]. In cardinality constraint (CC) programming one allows atoms of the form kXl, where $k \leq l$ are nonnegative integers and X is a finite set of atoms, to appear in both the head and the body of clauses. The meaning of the atom kXl is "at least k but not more than l of atoms from X belong to the intended model M". In fact, the work of [NSS99,NS00] allows for more general weight constraint atoms. That is, if $wt(\cdot)$ is a nonnegative rational valued function on the underlying set of literals of a logic program, then we interpret kXl to mean that in an intended model M of P, $k \leq [\sum_{a\in X\cap M} wt(a) + \sum_{a\in X\setminus M)} wt(\neg a)] \leq l$. These extensions have been implemented in *smodels*, see [NS00]. The main purpose of this paper is to introduce

an extension of DATALOG$^{\neg}$ which we call *set constraint programming* which incorporates both cardinality constraint atoms and weight constraint atoms as a special case.

In [NSS99], Niemelä, Simons, and Soininen defined a natural analogue of stable models for CC-logic programs which we called CC-stable models in [MR02]. The construction of [NSS99] significantly generalizes an older proposal due to Sakama and Inoue [SI94] of stable semantics for programs admitting (in modern notation) constraints of the form $1X$ in the heads of the clauses. Niemelä, Simons, and Soininen used a modification of the Gelfond-Lifschitz transform [GL88], which we called the NSS transform in [MR02], to define CC-stable models. However, the presence of expressions of the form kXl in cardinality constraint programs forces one to abandon one of fundamental properties of stable models of normal logic programs, namely, that stable models of a normal program are minimal and hence form an antichain with respect to set inclusion. Once atoms of the form kXl are allowed, even CC-programs in which all clauses have empty bodies can have collections of CC-stable models that do not form an antichain and hence not all CC-stable models are minimal. The results of [MR02] show that there is a direct connection between stable models of normal logic programs and CC-stable models of CC-logic programs. That is, CC-stable models are *projections* of a stable models of a suitably chosen normal program in an extended language with a larger set of atoms. This means that the computation of CC-stable models of [NSS99] can be viewed as computing the stable model semantics for that extended program but then one *hides* all the atoms that do not occur in the original program. A similar result, but with respect to a different formalism, the so called answer sets with nested expressions, has been obtained by Ferraris and Lifschitz [FL01].

The main purpose of this paper is to define an extension of CC-logic programs, called *SC-logic programs* where one replaces atoms of the form kXl by a more general set constraint of the form $\langle X, \mathcal{F}\rangle$ where $\mathcal{F}$ is an *arbitrary* family of subsets of X. Here the intended meaning of $\langle X, \mathcal{F}\rangle$ is that in an intended model M, $M \cap X \in \mathcal{F}$. We will call such atoms *set-constraint atoms* or *SC-atoms* and the corresponding programs *SC-logic programs*. It is easy to see that a CC-atom kXl is just a SC-atom $\langle X, \mathcal{F}_{k,l}\rangle$ where $\mathcal{F}_{k,l}$ is a family of subsets of X

$$\mathcal{F}_{k,l} = \{Y \subseteq X : k \leq |Y| \leq l\}.$$

We shall show that results of [NSS99] and the results of [MR02] can be extended to the setting of SC-logic programs. For example, we shall show that one can extend the ideas of [NSS99] to define a natural notion of SC-stable models. Moreover, while the complexity of various problems associated with SC-stable models may be high due to the fact that the Kolmogorov complexity of $\mathcal{F}$ in a SC-atom $\langle X, \mathcal{F}\rangle$ may be large, the basic interpretation result that SC-stable models of SC-programs are traces of stable models of normal logic programs over an extended language continues to hold. We shall show that SC-atoms can incorporate arbitrary aggregation functions and that SC-atoms allow one to express other classes that may be important in applications. We should note that the

SAT community has studied similar issues such as considering *pseudo-Boolean* constraints [ARMS02]. Various applications such as wire outlay on the chips, model checking, and timing of chips motivate the study of such extensions. In the ASP community, more limited extensions of DATALOG$^{\neg}$ have been studied in [ET01] and other papers.

While there seems to be a natural notion of stable models for SC-programs, the notion of a supported model for SC-programs is not so straightforward. One immediate problem is that the natural extension of the one-step provability operator T_P for a normal logic program P leads to a non-deterministic operator. For example, even for the simple program P which consists of a single clause

$$\langle X, \mathcal{F} \rangle \leftarrow$$

one would naturally define $T_P(A)$ to be $\{Y : Y \in \mathcal{F}\}$ for any A which is a nondeterministic operator. We shall show that one can define a natural notion of SC-supported model for SC-logic programs as a model of P which is a fixed point of an appropriate one-step provability operator. However this approach will force us to investigate nondeterministic operators which have not been previously considered in the logic programming literature. As we shall see the properties of nondeterministic operators are very different from the properties of deterministic operators. For example, there are natural notions of fixed points and monotonicity for nondeterministic operators which reduce to the usual notions of fixed points and monotonicity if the operator is deterministic. However, we shall give an example of a monotone nondeterministic operator that does not have a fixed point. Thus the straightforward generalization of the Tarski-Knaster theorem fails for nondeterministic operators. Instead, we prove two generalizations of Knaster-Tarski theorem that are applicable to nondeterministic operators. One of these generalizations can be applied to show that a SC-stable model of a SC-logic program P is always a SC-supported model.

2 Set constraints and logic programs

Let X be a set. The power set of X, $\mathcal{P}(X)$, is the collection of all subsets of X. A *set constraint atom* for a set X of atoms is a pair $\langle X, \mathcal{F} \rangle$ where $\mathcal{F} \subseteq \mathcal{P}(X)$. Given a set of atoms M and a set constraint atom $\langle X, \mathcal{F} \rangle$, we say that M *satisfies* $\langle X, \mathcal{F} \rangle$, in symbols $M \models \langle X, \mathcal{F} \rangle$, if $M \cap X \in \mathcal{F}$. We say that M satisfies a collection B of set constraint atoms if M satisfies all set constraint atoms in B. A *set constraint clause* (SC-clause for short) is an expression of the form

$$s \leftarrow s_1, \ldots s_k \tag{1}$$

where s and each s_i are set constraint atoms. The body of the clause (1) is the set of set constraint atoms $\{s_1, \ldots, s_k\}$. A *set constraint logic program* (SC-logic program for short) is a collection P of set constraint clauses.

We can now introduce a notion of a *model* of a SC-logic program. A set of atoms M satisfies a clause $C = s \leftarrow s_1, \ldots s_k$ if the fact that all set constraint

atoms in the body of C are satisfied by M implies that M satisfies s as well. A set of atoms M satisfies a SC-logic program P if it satisfies all clauses of P.

We note that the satisfaction of atoms can be easily expressed in terms of the satisfaction of set constraints. Namely, $M \models a$ if and only if $M \models \langle\{a\}, \{\{a\}\}\rangle$. Similarly, satisfaction of negated atoms can also be expressed in terms of the satisfaction of set constraints. Namely, $M \models \neg a$ if and only if $M \models \langle\{a\}, \{\emptyset\}\rangle$. Thus we could write each literal a or $\neg a$ in a normal logic program P as a set constraint atom and hence we can consider each normal logic program as a special case of a SC-logic program. However such a translation makes normal logic programs much harder to read. Thus, in what follows, we shall simply write a for the set constraint atom $\langle\{a\}, \{\{a\}\}\rangle$ and $\neg a$ for the set constraint atom $\langle\{a\}, \{\emptyset\}\rangle$.

As we mentioned in the introduction, set constraint atoms can express general aggregation functions.

Example 1. (**Cardinality and Weight Constraint Atoms**) As described in the introduction a CC-atom kXl can be expressed as the SC-atom $\langle X, \mathcal{F}_{k,l}\rangle$ where $\mathcal{F}_{k,l} = \{Y \subseteq X : k \leq |Y| \leq l\}$. Similarly if we have a weight function wt on literals, the more general weight constraint kXl considered [NS00] where a model M satisfies kXl if and only if

$$k \leq [\sum_{a \in X \cap M} wt(a) + \sum_{b \in X - M} wt(\neg b)] \leq l$$

can be expressed as the SC-atom $\langle X, \mathcal{F}\rangle$ where

$$\mathcal{F} = \{Y \subseteq X : k \leq [\sum_{a \in Y} wt(a) + \sum_{b \in X - Y} wt(\neg b)] \leq l\}.$$

□

Example 2. (**SQL Aggregate Atoms**) Let X be a finite set of atoms and let $\mu : X \rightarrow \mathrm{R}$ be a real function. Each such function μ allows us to construct a variety of set constraint atoms. For example, to each $Y \subseteq X$, we can assign the following functions that are used in SQL queries: $|Y|$, $\mathrm{sum}(Y) = \sum_{y \in Y} \mu(y)$, $\min(Y) = \min_{y \in Y} \mu(y)$, $\max(Y) = \max_{y \in Y} \mu(y)$, $\mathrm{avg}(Y)$, where avg assigns to Y the real number 0 if $Y = \emptyset$ and assigns the real number $\frac{\mathrm{sum}(Y)}{|Y|}$, otherwise. For every two real numbers a, b such that $a \leq b$, we define the following families of sets:

1. $C_X^{a,b} = \{Y : a \leq |Y| \leq b\}$
2. $S_X^{a,b} = \{Y : a \leq \mathrm{sum}(Y) \leq b\}$
3. $\mathrm{Max}_X^{a,b} = \{Y : a \leq \max(Y) \leq b\}$
4. $\mathrm{Min}_X^{a,b} = \{Y : a \leq \min(Y) \leq b\}$
5. $\mathrm{avg}_X^{a,b} = \{Y : a \leq \mathrm{avg}(Y) \leq b\}$

For each family $\mathcal{F}$ described in (1)-(5), we obtain a set constraint $\langle X, \mathcal{F}\rangle$. □

Example 3. (**Programs with External Modules**) In [EGV97], Eiter, Gottlob and Veith studied logic programs whose clauses contain *modules* in their bodies. Modules are programs π (written in some fixed programming language) that return subsets of some finite set of atoms X. Let us define R_π as the set of those subsets of X that can be returned by π. Eiter, Gottlob and Veight show how a stable semantics can be assigned to programs that contain atoms of the form $\langle X, R_\pi \rangle$ in the body of clauses. Our construction of SC-stable models below extends the work of [EGV97] in that SC-logic programming allows modules to occur both in the heads and in the bodies of clauses. □

It should be however clear, that there are other families of subsets of a set that are of interest.

Example 4. Given a finite set of atoms, let $\mathcal{F}_{even} = \{Y \subseteq X : |Y| \textit{ is even}\}$ and $\mathcal{F}_{odd} = \{Y \subseteq X : |Y| \textit{ is odd}\}$. Then $\langle X, \mathcal{F}_{even} \rangle$ and $\langle X, \mathcal{F}_{odd} \rangle$ are set constraint atoms. □

Clearly, the notion of satisfaction defined above generalizes the usual notion of satisfaction for Horn logic programming clauses and programs. Unlike Horn logic programs, SC-logic programs do not have to have models even in the case where the body of each SC-clause in empty.

Example 5. Consider the SC-logic program P which consisting of the following two clauses. $\langle \{a, b\}, \mathcal{F}_{Even} \rangle \leftarrow \quad \langle \{a, b\}, \mathcal{F}_{Odd} \rangle \leftarrow$. It is easy to see that P has no model M since to be a model of P would require that $|M \cap \{a, b\}|$ is both even and odd. □

We can, however, prove the following.

Proposition 1. *If a SC-logic program P possesses a model, then it possesses an inclusion-minimal model.*

3 Stable Models of Set Constraint Logic Programs

In this section, we shall generalize the notion of CC-stable models introduced by Niemelä, Simons and Soininen [NSS99] to the class of SC-logic programs. To understand our extension, we first formally define cardinality constraint logic programs (CC-logic programs). The syntax of CC-logic programs admits two types of atoms: (i) ordinary atoms from a set At and (ii) atoms of the form kXl where X is a finite set of atoms from At, k is a natural number (i.e. $k \in \omega$), $l \in \omega \cup \{\infty\}$ and $k \leq l$. When $l = \infty$, we abbreviate kXl as kX. Such new atoms will be called *cardinality constraints*. The intended meaning of an atom kXl is "out of atoms in X at least k but not more than l belong to the intended model." Notice that the meaning of the negated atom, $\neg p$ is precisely the same as that of $0\{p\}0$. Therefore we shall assume that the bodies of rules of CC-logic programs contain only atoms of the form kXl and atoms from At. That is, a CC-clause is either a clause of the form

$$p \leftarrow q_1, \ldots, q_m, k_1 X_1 l_1, \ldots, k_n X_n l_n \tag{2}$$

or

$$kXl \leftarrow q_1, \ldots, q_m, k_1X_1l_1, \ldots, k_nX_nl_n. \quad (3)$$

We note that either m or n can be zero. Thus the head of CC-clauses is either of the form p where p is an atom from At or kXl where k, X, and l satisfy the conventions described above. We say that a set of atoms $M \subseteq At$ satisfies the cardinality constraint kXl, in symbols $M \models kXl$, if $k \leq |X \cap M| \leq l$. Similarly we say that $M \models p$ where $p \in At$, if $p \in M$. By treating the commas in the bodies of clauses as conjunctions, we say that $M \models body(C)$ if all atoms occurring in $body(C)$ belong to M and all cardinality constraints occurring in $body(C)$ are satisfied by M. Finally, we say that M satisfies a clause C, $M \models C$, if either M does not satisfy the body of C or M satisfies the head of C.

A CC-logic program is a set of CC-clauses of the form (2) or (3). We say that M is model of P, $M \models P$, if M satisfies all CC-clauses $C \in P$.

A class of programs called Horn CC-programs play a role similar to that of Horn programs in ordinary logic programming. A *Horn CC-clause* is a CC-clause where the head of the clause is an ordinary atom and all the cardinality constraint atoms $k_iX_il_i$ in the body have $l_i = \infty$, i.e., it is of the form

$$H = p \leftarrow q_1, \ldots q_m, k_1X_1, \ldots, k_nX_n$$

Niemelä, Simons and Soininen observe that the one-step provability operator associated with a Horn CC-program is monotone and hence a Horn CC-program P has a least fixed point, M^P. Moreover, they show that M^P is the least model of P.

Next we introduce the analogue of the Gelfond-Lifschitz reduct for CC-logic logic programs which we call the NSS-reduct. The NSS-reduct of a CC-logic program P with respect to a set M of ordinary atoms is defined as follows. First we eliminate all clauses D of P such that M does not satisfy the body of D. For the remaining clauses C of P, replace C by C^M where

1. $C^M = p \leftarrow q_1, \ldots, q_m, k_1X_1 \ldots, k_nx_n$ if $C = p \leftarrow q_1, \ldots, q_m, k_1X_1l_1, \ldots, k_nX_nl_n$ and

2. C^M is a collection of Horn constraint clauses of the form $p \leftarrow q_1, \ldots, q_m, k_1X_1, \ldots, k_nX_n$ for each $p \in X \cap M$ if $C = kXl \leftarrow q_1, \ldots, q_m, k_1X_1l_1, \ldots, k_nX_nl_n$.

We let P^M denote the Horn CC-program consisting of the set of all C^M such that $C \in P$ and M satisfies the body of C. Following [NSS99], we say that M is a *CC-stable model* of P if (i) M is a model of P and (ii) M is the least model of the Horn CC-program P^M.

To define the notion of a SC-stable models for a SC-logic program, we first must define an extension of the NSS-reduct. Our first step is to define an appropriate analogue of clauses of the form $kX\infty$. To this end, define the upper-closure of $\mathcal{F}$ with respect to X to be the family $\overline{\mathcal{F}}_X$ where

$$\overline{\mathcal{F}}_X = \{Y \subseteq X : \exists_Z(Z \in \mathcal{F} \wedge Z \subseteq Y)\}.$$

We will drop the subscript X when it is determined by the context. A family of subsets of X is *closed* if $\overline{\mathcal{F}} = \mathcal{F}$. A closed family is nothing more than an upper ideal in the partially ordered set $\langle \mathcal{P}(X), \subseteq \rangle$. Notice that closure of a closed family $\mathcal{F}$ of subsets of X is $\mathcal{F}$ itself.

Example 6. The closure kXl is $kX\infty$ (that is, kX), i.e. family $\{Y \subseteq X : k \leq |Y|\}$. The closure of $\mathcal{F}_{even}$ is the entire powerset of X (recall that X is finite). The closure of $\mathcal{F}_{odd}$ is the set of all non-empty subsets of X.

Observe that an atom a is shorthand for the SC-atom $\langle \{a\}, \{\{a\}\} \rangle$ which is automatically closed. However the atom $\neg a$ is shorthand for the SC-atom $\langle \{a\}, \{\emptyset\} \rangle$ whose closure is $\langle \{a\}, \{\emptyset, \{a\}\} \rangle$. □

Clearly, different families of sets may generate the same closure. However, closure of a family $\mathcal{F}$, $\overline{\mathcal{F}}$, has precisely the same inclusion-minimal elements as $\mathcal{F}$. We define the *closure* of an SC-atom $\langle X, \mathcal{F} \rangle$ to be $\langle X, \overline{\mathcal{F}} \rangle$.

This given, we can now define the analogue of Horn program. A *Horn SC-clause* is a SC-clause where the head of the clause is an ordinary atom and all SC-atoms in the body are closed, i.e. a clause of the form

$$H = p \leftarrow q_1, \ldots q_m, \langle X_1, \mathcal{F}_1 \rangle, \ldots, \langle X_n, \mathcal{F}_m \rangle.$$

where $\mathcal{F}_i = \overline{\mathcal{F}_i}$ for all i. A Horn SC-logic program is a SC-program consisting entirely of Horn SC-clauses. As in [NSS99], one can show 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, M^P, which is a unique minimal model of P. Thus we have the following.

Proposition 2. *Let P be a Horn SC-logic program. Then:*

1. *There is a least model of P, M_P.*
2. *There is a deterministic monotone operator S_P such that M_P is the least fixed point of S_P. The fixed point of S_P is reached in at most ω steps regardless of the size of P.*

We will now define the NSS transform of a SC-logic program P with respect to a set of atoms M. Let P be a SC-logic program and let M be a subset of At. The NSS transform , $\mathrm{NSS}(P, M)$, of P with respect to M is defined in two steps. First, eliminate from P all clauses whose bodies are *not* satisfied by M. In the second step, in each remaining clause we execute the same operations as in the original NSS transform, except that the closure of the atoms in the bodies of clauses is as defined above. That is for each clause $\langle X, \mathcal{F} \rangle \leftarrow q_1, \ldots, q_m, \langle X_1, \mathcal{F}_1 \rangle, \ldots, \langle X_k, \mathcal{F}_k \rangle$, and for each $a \in X \cap M$, we generate the clause $a \leftarrow q_1, \ldots, q_m, \langle X_1, \overline{\mathcal{F}_1} \rangle, \ldots, \langle X_k, \overline{\mathcal{F}_k} \rangle$.

It is easy to see that the resulting program $\mathrm{NSS}(P, M)$ is a Horn SC-logic program. Consequently, $\mathrm{NSS}(P, M)$ has a least model $N_{P,M}$. We then say that M is an *SC-stable model of* P if (I) M is a model of P and (II) $M = N_{P,M}$.

We note that the first condition that M is model of P need not to be required for normal logic program P because the least model of the Gelfond-Lifschitz transform of P is automatically a model of P. This is not true for SC-logic programs as our next example will show.

Example 7. Let P be a SC-logic program consisting of the following two clauses:

$$1\{a,b,c,d\}2 \leftarrow \quad 3\{a,b,c,d\}4 \leftarrow$$

Note that $M = \{a,b,c,d\}$ is not a model of P. In fact, it is easy to see that P has no models. However $\mathrm{NSS}(P,M)$ consists of four atomic clauses:
$a \leftarrow \quad b \leftarrow \quad c \leftarrow \quad d \leftarrow$. Thus $M = N_{P,M}$. However, since M is *not* a model of P, M is not a stable model of P. □

We note that pruning process for the NSS-transform is more extensive than in the case of Gelfond-Lifschitz (GL) transform of normal logic programs. That is, for the GL-transform, we prune those clauses where M contradicts the negative part of the body, while for the NSS-transform, we eliminate clauses with bodies *not* satisfied by M. As shown by Truszczynski [MT95], this stronger Gelfond-Lifschitz-like transformation leads to the same class of stable models.

Let P be a normal logic program. We identify P with a SC-logic program where each atom and each negated atom are expressed by set constraints (a is expressed by $\langle\{a\},\{\{a\}\}\rangle$, $\neg a$ is expressed by $\langle\{a\},\{\emptyset\}\rangle$). We then have the following result which is essentially the same result that Niemelä, Simons and Soininen established for CC-logic programs.

Proposition 3. *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 a SC-logic program.*

We end this section with an analogue of the main result of [MR02] for SC-logic programs.

Theorem 1. *Let P be a SC-logic program over a language $\mathcal{L}$. Then there is a normal logic program $\overline{P}$ over a extended language $\overline{\mathcal{L}}$ of $\mathcal{L}$ such that*

(i) *For each SC-stable model M of P, there is a unique stable model $\overline{M}$ of $\overline{P}$ such that M is the restriction of $\overline{M}$ to the language $\mathcal{L}$.*
(ii) *For each stable model $\overline{M}$ of $\overline{P}$, the restriction of $\overline{M}$ to the language $\mathcal{L}$ is a SC-stable model of P.*

4 Nondeterministic lattice operators

We observed that introduction of set constraints leads naturally to investigation of non-deterministic operators in complete lattices. Consequently, in this section, we will investigate nondeterministic operators and establish some of their properties.

Let $\langle L, \leq_L\rangle$ be a complete lattice. A *(nondeterministic) operator* in L is any function O from L to the powerset of L. We say that an operator O is *deterministic*, if for every $x \in L$, the size of $O(x)$, $|O(x)|$, is equal to 1. If the operator O is deterministic, we can identify O with a mapping from L to L, namely, assigning to $x \in L$ the unique element of $O(x)$. Conversely, a mapping

Q from L to L can be identified with a nondeterministic operator O_Q from L to $\mathcal{P}(L)$ by assigning to every x the set consisting of a single lattice element $Q(x)$.

Nondeterministic operators naturally occur in the context of set constraint programs as our next example will show.

Example 8. Let P consist of a single clause

$$1\{p_1, p_3\}2 \leftarrow 2\{p_2, p_4, p_5\}3$$

It is natural to assign to $M = \{p_1, p_2, p_5\}$ each of the following values $\{p_1\}$, $\{p_3\}$, and $\{p_1, p_3\}$. Unless additional criteria are used, each of these values is a correct value because the intention of the programmer described in this clause is that *any* of these values is acceptable. □

We say that a nondeterministic operator $O : L \to \mathcal{P}(L)$ is *monotone* if for every $x, y \in L$ such that $x \leq_L y$, it is the case that for every $z \in O(x)$, there is $t \in O(y)$ such that $z \leq_L t$. A fixed point of an operator $O : L \to \mathcal{P}(L)$ is any x such that $x \in O(x)$.

Proposition 4. *If $O : L \to L$ is a monotone operator, then its interpretation as a nondeterministic operator from L to $\mathcal{P}(L)$ is also monotone.*

Thus a monotone *deterministic* operator always possesses a fixed point. It is tempting to conjecture that a nondeterministic monotone operator always possesses a fixed point. However, this is not always the case as our next example will show.

Example 9. Let L be the Boolean lattice $\mathcal{P}(N)$ where N is the set of non-negative integers. The structure $\langle L, \subseteq \rangle$ forms a complete lattice. Define a nondeterministic operator O from $\mathcal{P}(N)$ to $\mathcal{P}(\mathcal{P}(N))$ as follows.

(a) If $X \subseteq N$ is finite set of cardinality n, then $O(X)$ is a family consisting of a single set $\{0, \ldots, n\}$.
(b) If $X \subseteq N$ is an infinite set, then $O(X)$ is the family of all finite subsets of N, $\mathcal{P}_{fin}(N)$.

First, observe that O is a monotone nondeterministic operator. Indeed, assume $X \subseteq Y$.
Case 1. Both X and Y are finite. If $X \subseteq Y$ so that $|X| \leq |Y|$, then $O(X) = \{\{0, \ldots, |X|\}\}$, $O(Y) = \{\{0, \ldots, |Y|\}\}$ and every element of $O(X)$ is contained in every element of $O(Y)$ and hence the monotonicity condition holds.
Case 2. Both X and Y are infinite. Then every element of $O(X)$ belongs to $O(Y)$, thus the monotonicity condition holds.
Case 3. X is finite and Y is infinite. Then since $O(Y) = \mathcal{P}_{fin}(N)$, the only element of $O(X)$, being finite, belongs to $O(Y)$. Thus again the monotonicity condition holds.

However, O has no fixed point. For let X be a subset of N. If X is finite, then the only element of $O(X)$ has size bigger than that of X, and thus X cannot be a fixed point. When X is infinite, X does not belong to $O(X)$ at all. □

A close look at Example 9 will show that there are two reasons for the lack of fixed points. First, we allowed $O(X)$ be infinite when X is infinite. Second, the limit of the values of O of a directed family is not a value of O. We will now state two results on the existence of fixed points of monotonic nondeterministic operators.

Proposition 5. *Let O be a monotone nondeterministic operator from a complete lattice L to $\mathcal{P}(L)$ such that $O(\bot)$ is nonempty, and for every X, $O(X)$ is finite. Then O possesses a fixed point.*

Proposition 6. *Assume that L is a complete lattice and $O : L \to \mathcal{P}(L)$ is a nondeterministic operator satisfying following two properties:*

1. *O is monotone*
2. *For every $x \in L$, the family $O(x)$ has a maximum element*

Then O possesses a fixed point.

Notice that both Propositions 5 and 6 are generalizations of Knaster-Tarski theorem in that monotone deterministic operators automatically satisfy the hypotheses of the each theorem.

5 Generalization of van Emden-Kowalski operator

In this section, we develop an analogue of the one-step provability operator for SC-logic programs. This operator is a generalization of the familiar van Emden Kowalski operator [AvE82].

Let P be a SC-logic program. If M be a set of atoms, then we let $Sat_{P,M} = \{C \in P : M \models body(C)\}$. Thus $Sat_{P,M}$ consists of those SC-clauses in P such that M satisfies the body of C. Fix a SC-logic program P. An *M-satisfier* is any function from $Sat_{P,M}$ to $\mathcal{P}(At)$ which assigns to each clause $C \in Sat_{P,M}$, an element of the family $\mathcal{F}$ for which $\langle X, \mathcal{F}\rangle$ is the head of C. Thus an M-satisfier provides values to satisfy the heads of clauses whose bodies are satisfied by M.

Example 10. Let P be this SC-logic program:

$C_1 :\ \langle\{a, b, c\}, \mathcal{F}_{even}\rangle \leftarrow a$
$C_2 :\ 2\{a, b, c, d\}3 \leftarrow 1\{b, c, d\}3$
$C_3 :\ c \leftarrow b$

Take as M the set $\{b, d\}$. It satisfies the second and the third clauses, but not the first one. There are several M-satisfiers for M. Clearly every M-satisfier must assign $\{c\}$ to clause C_3. However for clause C_2, an M-satisfier can assign any two or three element subset of $\{a, b, c, d\}$. □

Now, we are ready to define the nondeterministic operator T_P associated with the program P. Specifically, we define

$$T_P(M) = \{\bigcup Rng(f) : f \text{ is an } M\text{-satisfier}\}$$

Thus the value of the T_P operator on M is the collection of "candidates", each candidate being the union of the range of some M-satisfier.

Example 11. In our previous example there are 7 possible values for $T_P(\{b,d\})$. Each of those contains c. When we inspect $N = \{b,c,d\}$, we again find the same 7 values in $T_P(N)$. It is easy to see that the set N is a fixed point for the operator T_P. □

We note that the nondeterministic operator T_P for SC-logic programs P is, in fact, a generalization of the familiar van Emden-Kowalski operator. Indeed, if P is a normal logic program and M is a subset of the set of atoms At, then there is just one M-satisfier. Specifically, it is a unique function $f : Sat_{P,M} \to \mathcal{P}(At)$ such that $f(C) = \{head(C)\}$ where $head(C)$ is the head of C. Thus for normal logic programs, the operator T_P is deterministic.

Recall that an atom $\langle X, \mathcal{F}\rangle$ *closed*, if $\mathcal{F}$ is an upper ideal in $\mathcal{P}(X)$, that is, if

$$\forall_{Y,Z}(Y \in \mathcal{F} \wedge Y \subseteq Z \subseteq X \Rightarrow Z \in \mathcal{F}).$$

An SC-logic program P is *closed* if all the SC-atoms which occur either in the head or the body of a clause P are closed. Observe that if the atom $\langle X, \mathcal{F}\rangle$ is closed, $M \models \langle X, \mathcal{F}\rangle$ and $M \subseteq M'$, then $M' \models \langle X, \mathcal{F}\rangle$. In fact, this property characterizes the closed SC-atoms. Specifically, if for all $M \subseteq M' \subseteq At$, $M \models \langle X, \mathcal{F}\rangle$ implies $M' \models \langle X, \mathcal{F}\rangle$, then $\mathcal{F}$ must be closed.

We now have the following result.

Proposition 7. *If P is a closed SC-logic program, then T_P possesses a fixed point M. Moreover M can be chosen to be a model of P.*

Unlike the situation for normal logic programs, it is not the case that every fixed point of T_P is a model of P as our next example will show.

Example 12. Let P be this program:

$$1\{p,q,r\}2 \leftarrow p$$
$$2\{p,q,r\}3 \leftarrow p$$

This program has four fixed points: $\emptyset$, $\{p,q\}$, and $\{p,r\}$ and $\{p,q,r\}$. It is easy to see that first three are models of P while the last one is not.

Clearly, it is even easier to find a model which is not a fixed point. Any non-supported model of a normal logic program induces such example.

We call a set M of atoms a *supported* model of a SC-logic program P if M is a model of P and M is a fixed point of the nondeterministic operator T_P.

Our final result of this section shows that stable models of SC-logic programs are fixed points of the nondeterministic operator T_P considered in Section 5. This generalizes the result of Gelfond and Lifschitz which is that stable models of normal logic programs are always supported models.

Proposition 8. *Let P be a SC-logic program and let M be a set of atoms. If M is a stable model of P then M is a fixed point of the nondeterministic operator T_P.*

6 Conclusions and further research

We discuss some issues related to the research presented in this paper. The first question is: "What families are representable as families of stable models of a program?" In the case of a finite collection of finite sets, the answer is obvious. Each such family $\mathcal{X}$ is representable. Indeed, if $\mathcal{X}$ is empty, then any inconsistent program is appropriate. If $\mathcal{X}$ is nonempty, then two cases are possible. If $\mathcal{X}$ consists of an empty set, then $\neg p \leftarrow$ is the appropriate program. Otherwise, take $X = \bigcup \mathcal{X}$ and let P consist of a single clause $\langle X, \mathcal{X} \rangle \leftarrow$. It is easy to see that P can be used to represent $\mathcal{F}$.

It turns out that the assumption that all sets in the family $\mathcal{F}$ are finite is immaterial. That is, we can prove if $\mathcal{F}$ is any finite collection of sets of atoms, then there is a SC-logic program $P_{\mathcal{F}}$ such that the family of SC-stable models of $P_{\mathcal{F}}$ is $\mathcal{F}$.

Unfortunately, these results tell us nothing about what infinite families sets can form the set of SC-stable models of some SC-logic program. In the case stable models of normal logic programs, two characterizations are available. A topological characterization of the families representable as the set of stable modes of a logic program has been found by A. Ferry [Fe94]. An alternative solution, in recursion-theoretic terms, has been found in [MNR90]. No corresponding result for SC-logic programs, or even CC-logic programs, are known.

Another problem which awaits settlement is the problem of well-founded semantics [VRS91] for SC-logic programs. It is clear that *some* form of well-founded semantics can be obtained by the reduction to the stable semantics of normal logic program. We believe, though, that a more direct construction via approximation of the nondeterministic operator in the spirit of [DMT02] exists.

Acknowledgments

The first author's research has been partially supported by the NSF grants IIS-0097278 and IIS-0325063. The second author's research has been partially supported by the ARO contract DAAD19-01-1-0724.

References

[ARMS02] F.A. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A backtrack-search pseudo-boolean solver and optimizer. SAT02, 2002.

[AvE82] K.R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

[CMT96] P. Cholewiński, W. Marek, and M. Truszczyński. Default reasoning system DeReS. KR96, pages 518–528. Morgan Kaufmann, 1996.

[ASP01] Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming, Stanford, CA, USA, 2001.

[Ba03] C. Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, 2003.

[DMT02] M. Denecker, V.W. Marek and M. Truszczyński. Ultimate approximations in nonmonotonic knowledge representation systems. KR02, pages 177–188, Morgan-Kaufmann, 2002.

[ET01] D. East and M. Truszczynski. More on wire-routing. In: [ASP01], 2001.

[ET02] D. East and M. Truszczynski. *aspps* solver. http://www.cs.uky.edu/ai/. 2002.

[EGV97] T. Eiter, G. Gottlob and H. Veith. Modular Logic Programs and General Quantifiers, LPNMR97, pages 290–309, 1997.

[ELM+98] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons, and Benchmarks. KR98, pages 406–417, 1998.

[FL01] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. To appear in *Theory and Practice of Logic Programming*.

[Fe94] A. Ferry. *Topological Characterizations for Logic Programming Semantics*, Ph.D. Dissertation, University of Michigan, 1994.

[GL88] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. ISLP88, pages 1070–1080, 1988.

[KS99] H.A. Kautz and B. Selman. Unifying sat-based and graph-based planning. IJCAI99, pages 318-325. Morgan Kaufmann, 1999.

[MNR90] V. Marek, A. Nerode, and J. B. Remmel. Nonmonotonic rule systems I. *Annals of Mathematics and Artificial Intelligence*, 1: 241-273, 1990.

[MR02] V. W. Marek and J.B. Remmel. On logic programs with cardinality constraints. NMR9, pages 219–228, 2002.

[MT95] V. Marek and M. Truszczyński. Revision Programming. *Theoretical Computer Science* 190(2):241–277, 1995.

[MT99] V. Marek and M. Truszczyński. Stable Models and an Alternative Logic Programming Paradigm. *The Logic Programming Paradigm*, pages 375–398. Springer-Verlag, 1999.

[NS97] I. Niemelä and P. Simons. Smodels — an implementation of the stable model and well-founded semantics for normal logic programs, LPNMR97, pages 420–429, 1997.

[NS00] I. Niemelä and P. Simons. Extending Smodels System with Cardinality and Weight Constraints. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer Academic Publishers, 2000.

[NSS99] I. Niemelä, P. Simons. and T. Soininen. Stable Model Semantics of Weight Constraint Rules. LPNMR99, pages 317–331, 1999.

[SI94] C. Sakama and K. Inoue. An alternative approach to the Semantics of Disjunctive Logic Programs and Deductive Databases. *Journal of Automated Reasoning* 13:145–172. 1994.

[SK92] B. Selman and H. A. Kautz. Planning as satisfiability. ECAI-92, pages 359-363. Wiley, 1992.

[VRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.