# On Logic Programs with Cardinality Constraints

**Victor W. Marek**
Department of Computer Science
University of Kentucky
Lexington, KY 40506, USA
marek@cs.uky.edu

**Jeffrey B. Remmel**
Department of Mathematics
University of California
La Jolla, CA 92093, USA
jremmel@ucsd.edu

## Abstract

We investigate cardinality-constraint (CC) logic programs as implemented in the ASP solver *smodels*. Niemelä, Simons and collaborators in [NSS99, NS00, Syr01] defined a notion of stable model, which we call CC-stable, for CC-logic programs that differs from the usual notion of stable model of logic programs in a variety of ways. For example, it is not always the case that the set of CC-stable models of a CC-program form an anti-chain. The main result of this paper is to show that there is a natural transformation of a CC-logic program $P$ into a standard logic program $Q$ in an extended language such that the CC-stable models of $P$ are the projections of the stable models of $Q$ to the original language. Niemelä, Simons and Soinenen [NSS99] proved that the existence problem for CC-stable models of CC-logic programs is NP-complete. We show that the existence problem for CC-stable models for several restricted classes of CC-logic programs is also NP-complete. For example, the existence problem for CC-stable models for CC-logic programs where each clause has an empty body is already NP-complete.

## 1 Introduction

The current progress in computational knowledge representation and, in particular, the success of the current Answer Set Programming (ASP) engines [SK92, KS99, NS00, MT99, ASP01] makes it important to understand the semantics of underlying logical formalisms. In this extended abstract, we investigate cardinality-constraint logic programs (CC-logic programs) introduced by Niemelä, Simons and their collaborators in [NSS99, NS00, Syr01]. In particular, we study the notion of stable models for CC-logic programs as introduced by Niemelä, Simons and Soinene [NSS99], which we will call CC-stable models, and relate it to the standard notion of stable models for logic programs as introduced by Gelfond and Lifschitz [GL88].

CC-logic programs form a subclass of a larger class of programs called weight-constraint logic programs. The CC-stable semantics for this wider class of programs have been already implemented in the current version of stable semantics solver *smodels*. Niemelä and Simons [NS00] give numerous applications of CC-logic programs to various combinatorial optimization problems which show that CC-logic programs are natural and useful. Thus, since the CC-logic programs are accepted by a widely used solver, it is important to understand the CC-stable semantics of such programs. The goal of this paper is to show how the CC-stable semantics of a CC-logic program $P$ can be viewed as the projection of the stable logic semantics of a logic program $Q$ in an extended language. This will allow the programmer to understand the semantics of CC-logic programs as implemented by *smodels* as a natural generalization of the semantics of programs that were available in earlier versions.

A detailed explication of the notion of CC-stable models for CC-logic programs is given in [NSS99, NS00]. In those papers, the authors show that the CC-stable model semantics coincides with the usual stable model semantics of logic programs for those CC-logic programs with no cardinality constraints. However, the CC-stable model semantics for CC-logic programs differs from the usual stable model semantics for logic programs in two important ways. First, CC-stable models of CC-logic programs can be included one in another while a salient feature of stable models of ordinary logic programs is that they are minimal and, hence, they do not nest. Second, the definition of a

CC-stable model $M$ of a CC-logic program $P$ requires that $M$ must first be a model of $P$, while in case of ordinary logic programs $Q$, the fact that a stable model is a model of $Q$ comes for free. We will see that, with an appropriate point of view, both problems turn out to be illusory. We will show that there is a syntactic transformation $t(\cdot)$ which assigns to every CC-logic program $P$, an ordinary logic program in an extended language (i.e. one that admits more atoms) which preserves stable models in the following sense. Specifically, the restriction of the stable models of $t(P)$ to the language of $P$ is a one-to-one projection. That is, for every stable model $M$ of $t(P)$, the intersection $M$ with the Herbrand universe of $P$ is a CC-stable model of $P$. Conversely, every CC-stable model of $P$ uniquely extends to a stable model of $t(P)$. Thus our mapping $t(\cdot)$ is a meta-interpretation of CC-logic programs with CC-stable model semantics in logic programs with the usual stable model semantics of Gelfond and Lifschitz.

Another way to view this result is that CC-stable models of a CC-logic program $P$ is the result of implementing a **hide** operation, see [Syr01], p. 46, for the program $t(P)$. The operation **hide** is a useful construct that can improve the readability of an *smodels* output. For example, when $\textbf{hide}(p)$ is declared, the atoms of the form $p(\bar{a})$ that belong to the computed stable model are not displayed. In our case, the auxiliary atoms that occur in $t(P)$ but not in $P$, together with denotations for cardinality constraints that occur explicitly in $P$, are not shown. Thus conceptually, one could view the operation of *smodels* on a CC-logic program as a solver which processes the larger program $t(P)$, but only outputs those atoms which explicitly occur in $P$. Thus one can think of the use of **hide** as eliminating the atoms that are employed in the search for a stable model but which are not important from the point of view of the problem solved by the program. It is easy to see that the **hide** operation can result in projected stable models which are included in each other. For example, consider the following program $P$, which relative to the usual stable semantics, can be used to generate all subsets of a given set $A$.

**Example 1.1** The language of $P$ has two predicates: $in(\cdot)$ and $out(\cdot)$. Then $P$ contains the following pair of clauses for each $a \in A$.

$$in(a) \leftarrow \neg out(a)$$
$$out(a) \leftarrow \neg in(a)$$

It is easy to see that for any subset $S \subseteq A$,

$$\{in(a) : a \in S\} \cup \{out(a) : a \in A \setminus S\}$$

constitutes a stable model for $P$. It is also easy to see that when one hides the predicate $out(\cdot)$, the parts of stable models that are returned by the solver can nest.

For example, let $A = \{x, y, z\}$. Then $M_1 = \{in(x), out(y), out(z)\}$ and $M_2 = \{in(x), in(y), out(z)\}$ are two stable models of $P$. $M_1$ and $M_2$ do not nest, of course, but, if we declare $\textbf{hide}(out)$ in *smodels*, then *smodels* will display: $M_1 = \{in(x)\}$ and $M_2 = \{in(x), in(y)\}$. □

We view our results as a confirmation of the correctness of the approach of the authors of [NSS99]. While there may exist other semantics of CC-logic programs that behave similarly to the stable semantics when limited to ordinary logic programs, our meta-interpretation result, as well our characterization of proof schemes for CC-logic programs to follow, seem to indicate that the semantics implemented by *smodels* is the correct one.

An alternative approach to relating the CC-stable model semantics of CC-logic program to a more standard semantics is offered by Ferraris and Lifschitz in [FL01]. While we focus on CC-logic programs, Ferraris and Lifschitz look at the entire class of weight-constraint programs (WC-logic programs). Ferraris and Lifschitz translate WC-programs into *nested expressions* theories, that is, theories in the formalism of *nested expressions* of Lifschitz, Tang and Turner [LTT99]. The translation provided in [FL01] has the advantage that the set of CC-stable models of a WC-program coincides with the set of answer sets of nested expression image of the translated program. The drawback of the approach of [FL01] from our point of view is that it does not explain the CC-stable model semantics of CC-logic programs in terms of the simpler stable model semantics of logic programs. Instead, the translation of [FL01] explains the extension of the stable model semantics for the richer class of CC-logic programs in terms of the extension of the stable model semantics for an alternative extension of logic programs.

This paper is structured as follows. In section 2, we review the original definition of CC-stable model semantics for CC-logic programs [NSS99]. Then we will show how the basic proof-theoretic characterization of stable models in terms of proof schemes [MNR90] can be lifted to the case of CC-logic programs. In Section 3, we introduce our translation of CC-logic programs into ordinary logic programs. In Section 4, we prove some simple complexity results for CC-logic programs. In particular, we shall show that the basic NP-completeness result for the existence of CC-stable models of [NSS99] already holds for a very restricted class of CC-logic programs. Specifically, while

Niemela, Simons, and Soinenen show that the existence problem for stable models of CC-logic programs is NP-complete, we show that this result holds for CC-programs where every clause has an empty body. We also give a reduction of propositional satisfiability to the existence of CC-stable models for CC-logic programs that have no constraints in the bodies of clauses. This leads to yet another strengthening of basic complexity results of [NS00]. Finally, in Section 5, we indicate some other directions for future research on CC-logic programs and their extensions.

## 2 Logic programs, CC-logic programs and their stable semantics

Recall that a clause $C$ of for a logic program $P$ is a rule of the form

$$p \leftarrow q_1, \ldots, q_m, \neg r_1, \ldots, \neg r_n. \tag{1}$$

where $p, q_1, \ldots, q_m, r_1, \ldots, r_n$ are atoms from the set of atoms $At$ of the program. We shall refer to $p$ as the head of $C$, $head(C)$, $\{q_1, \ldots, q_m\}$ as the premises of $C$, $prem(C)$, $\{r_1, \ldots, r_n\}$ as the constraints of $C$, $cons(C)$, and $q_1 \wedge \ldots \wedge q_m \wedge \neg r_1 \wedge \ldots \wedge \neg r_n$ as the body of $p$, $body(C)$. A logic program is a set $P$ of clauses. To distinguish them from other clauses described below for CC-logic programs, we shall refer to clause of the form of (1) as an *ordinary* logic program clause. When $n = 0$, the clause $C$ is called a *Horn clause*. A Horn program is a set of Horn clauses. A Horn program always has a least model which we denote that model by $M_P$.

The operator $GL(\cdot, \cdot)$ [GL88] assigns to a logic program $P$ and a set of atoms $M$, the least model of the Horn program $P^M$ where $P^M$ consists of the set of all Horn clauses $C^M$ obtained from a clause $C$ of $P$ of the form of (1) as follows.

$$C^M = \begin{cases} \textbf{nil} & \text{if for some } i, 1 \leq i \leq n, \\ & r_i \in M \\ p \leftarrow q_1, \ldots, q_m & \text{otherwise} \end{cases} \tag{2}$$

We say that $M$ is a stable model of $P$ if $M = GL(P, M)$. Gelfond and Lifschitz proved that every stable model $M$ of $P$ is a model of $P$, in fact, a minimal and supported model of $P$.

Next we define a natural proof-theoretic construct associated to logic programs called *proof schemes*. Let $P$ be a logic program, then the set of proof schemes of $P$ can be defined inductively as follows.

1. If $C = p \leftarrow \neg r_1, \ldots, \neg r_n$ is a clause of $P$ (the case $n = 0$ is allowed), then $\langle\langle p\rangle, \langle C\rangle, \{r_1, \ldots, r_n\}\rangle$ is a proof scheme for $p$ in $P$
2. If $\langle\langle s_1, \ldots, s_k\rangle, \langle C_1, \ldots, C_k\rangle, \{t_1, \ldots, t_l\}\rangle$ is a proof scheme and $C = p \leftarrow q_1, \ldots, q_m, \neg r_1, \ldots, \neg r_n$ is a clause in $P$ and $\{q_1, \ldots, q_m\} \subseteq \{s_1, \ldots, s_k\}$, then

$$\langle\langle s_1, \ldots, s_k, p\rangle, \langle C_1, \ldots, C_k, C\rangle, \\ \{t_1, \ldots, t_l, r_1, \ldots, r_n\}\rangle$$

   is a proof scheme for $p$ in $P$.

If $\mathfrak{S} = \langle\langle s_1, \ldots, s_k\rangle, \langle C_1, \ldots, C_k\rangle, \{t_1, \ldots, t_l\}\rangle$ is a proof scheme in $P$, we refer to $s_k$ as the conclusion of $\mathfrak{S}$, $concl(\mathfrak{S})$, and say that $\mathfrak{S}$ is a proof scheme for $s_k$ in $P$. We also refer to $\{t_1, \ldots, t_k\}$ as the constraints of of $\mathfrak{S}$, $const(\mathfrak{S})$.

One can think of a proof scheme for a logic program as the analogue of a derivation or proof in classical logic. However, a proof scheme $\mathfrak{S}$ for $p$ not only contains the clauses that can be used to derive $p$ but also keeps track of the set of atoms that must be absent from prospective stable model $M$, namely $const(\mathfrak{S})$, if $p$ is to be an element of $GL(P, M)$. Thus we say that a proof scheme $\mathfrak{S}$ is *admitted* by $M$ if $M \cap const(\mathfrak{S}) = \emptyset$.

**Example 2.1** Let $P$ consist of clauses:

$$C_1 = p \leftarrow q, \neg r$$
$$C_2 = q \leftarrow \neg s$$
$$C_3 = s \leftarrow \neg q.$$

The triple

$$\mathfrak{S}_1 = \langle\langle q, p\rangle, \langle C_2, C_1\rangle, \{r, s\}\rangle$$

is a proof scheme for $p$ in $P$. Similarly, $\mathfrak{S}_2 = \langle\langle q\rangle, \langle C_2\rangle, \{s\}\rangle$ is a proof scheme for $q$ and $\mathfrak{S}_3 = \langle\langle s\rangle, \langle C_3\rangle, \{q\}\rangle$ is a proof scheme for $s$. The set $\{p, q\}$ is a stable model of $P$. It admits $\mathfrak{S}_1$ and $\mathfrak{S}_2$, but not $\mathfrak{S}_3$. □

The following result is proven in [MNR90].

**Proposition 2.1** *Let $M$ be a set of atoms contained the Herbrand base $At_P$ of the a logic program $P$. Then $M$ is a stable model of $P$ if and only if*

1. *Every atom $p$ of $M$ possesses a proof scheme $\mathfrak{S}_p$ in $P$ such that $M$ admits $\mathfrak{S}_p$*
2. *No atom $p$ in $At \setminus M$ possesses a proof scheme admitted by $M$.*

Proposition 2.1 implies the following property of models of programs.

**Corollary 2.2** *Let $P$ be a logic program and let $M$ be a model of $P$. Then $M$ is a stable model of $P$ if and only if every element of $M$ possesses a proof scheme in $P$ admitted by $M$.*

The advantage of proof schemes is that they are entities associated with programs and atoms and not with models. Proof schemes carry within themselves the information about their own applicability. Notice that Corollary 2.2 establishes a condition for models of $P$ that is easier to check than the conditions given in Proposition 2.1. Below we will extend the notion of proof scheme to CC-logic programs and prove a result analogous to Corollary 2.2. This is one reason why we believe that the definition of CC-stable models of cardinality-constraint programs is a natural generalization of stable semantics for ordinary logic programs.

We now formally define cardinality-constraint logic programs (CC-logic programs). The syntax of CC-logic programs admits two types of atoms: (i) ordinary atoms from 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 $k \leq l$. 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."[1] Notice that the meaning of the negated atom, $\neg p$ is precisely the same as that of $\{p\}0$. 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

$$p \leftarrow q_1, \dots, q_m, k_1X_1l_1, \dots, k_nX_nl_n \qquad (3)$$

or

$$kXl \leftarrow q_1, \dots, q_m, k_1X_1l_1, \dots, k_nX_nl_n. \qquad (4)$$

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 $kXl$, $M \models kXl$ 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 $M \models \mathit{body}(C)$ if all atoms occurring in $\mathit{body}(C)$ belong to $M$ and all cardinality constraints occurring in $\mathit{body}(C)$ 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 (3) or (4). We say that $M$ is model of $P$, $M \models P$, if $M$ satisfies all CC-clauses $C \in P$.

There is a particular class of programs called Horn constraint programs that play a role similar to that of Horn programs in ordinary logic programming. A *Horn constraint 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

$$H = p \leftarrow q_1, \dots q_m, k_1X_1, \dots, k_nX_n.$$

Niemelä, Simons and Soininen observe that the one step provability operator associated with a such Horn constraint program is monotone and hence a Horn constraint 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 clauses which we call the NSS-reduct. The NSS-reduct of a cardinality-constraint clause $C$ with respect to a set $M$ of ordinary atoms is defined as follows. First, eliminate all clauses $C$ such $M \not\models \mathit{body}(C)$. Next,

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

Given a CC-program $P$, we let $P^M$ denote a Horn constraint program consisting of all NNS-reducts of clauses $C \in P$. 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 constraint program $P^M$. It appears that, in the case of ordinary programs, the NSS-reduct prunes more clauses than GL-reduct. M. Truszczyński (unpublished) proved that this reduct results in the same notion of stable model.

Next we define the notion of a proof scheme for a CC-logic program and state a result analogous to Corollary 2.2. The basic idea is that a proof scheme should carry along all the information that is needed to see that an element $p$ is in a CC-stable model $M$. In particular, when we deal with atoms of the form $kXl$, we need to know the information that $k \leq |M \cap X| \leq l$. Thus our

[1] Customarily we do not write the lower bound if it is 0 nor the upper bound, if it is $\infty$.

proof schemes will carry along the information about what we expect $M \cap X$ to be. Formally, the notion of CC-proof scheme for a CC-logic program $P$ is defined inductively as follows.

1. Whenever $C = p \leftarrow l'_1 X_1 l''_1, \ldots, l'_n X_n l''_n$ is a clause in $P$ and for all $1 \leq i \leq n$, $Y_i$ is a subset of $X_i$ such that $l'_i \leq |Y_i| \leq l''_i$ then

   $$\langle\langle p\rangle, \langle C\rangle, \langle(l'_1 X_1 l''_1, Y_1), \ldots, (l'_n X_n l''_n, Y_n)\rangle\rangle$$

   is a CC-proof scheme for $P$. (The case $n = 0$ is allowed.)

2. Whenever $l' X l'' \leftarrow l'_1 X_1 l''_1, \ldots, l'_n X_n l''_n$ is a clause in $P$ and for all $1 \leq i \leq n$, $Y_i$ is a subset of $X_i$ such that $l'_i \leq |Y_i| \leq l''_i$ and $Y$ is a subset of $X$ such that $k \leq |Y| \leq l$, then for every $p \in Y$

   $$\langle\langle p\rangle, \langle C\rangle, \langle(l' X l'', Y), (l'_1 X_1 l''_1, Y_1), \ldots, (l'_n X_n l''_n, Y_n)\rangle\rangle$$

   is a CC-proof scheme for $P$. ( Again, the case $n = 0$ is allowed.)

3. Whenever

   $$\mathfrak{S} = \langle\langle s_1, \ldots, s_w\rangle, \langle C_1, \ldots, C_w\rangle, \langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_k)\rangle\rangle$$

   is a CC-proof scheme in $P$ and

   $$C = p \rightarrow q_1, \ldots, q_m, l'_1 Z_1 l''_1, \ldots, l'_n Z_n l''_n$$

   is a clause in $P$ such that $\{q_1, \ldots, q_m\} \subseteq \{s_1, \ldots, s_w\}$ and for all $1 \leq i \leq n$, $T_i$ is a subset of $Z_i$ such that $l'_i \leq |T_i| \leq l''_i$, then

   $$\langle\langle s_1, \ldots, s_w, p\rangle, \langle C_1, \ldots, C_w, C\rangle, \langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_r), (l'_1 Z_1 l''_1, T_1) \ldots, (l'_n Z_n l''_n, T_n)\rangle\rangle$$

   is a CC-proof scheme for $P$.

4. Whenever

   $$\mathfrak{S} = \langle\langle s_1, \ldots, s_w\rangle, \langle C_1, \ldots, C_w\rangle, \langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_k)\rangle\rangle$$

   is a proof scheme in $P$ and

   $$C = l' Z l'' \rightarrow q_1, \ldots, q_m, l'_1 Z_1 l''_1, \ldots, l'_n Z_n l''_n$$

   is a clause in $P$ such that $\{q_1, \ldots, q_m\} \subseteq \{s_1, \ldots, s_w\}$, for all $1 \leq i \leq n$, $T_i$ is a subset of $Z_i$ such that $l'_i \leq |T_i| \leq l''_i$ and $Y$ is a subset of $Z$ such that $l' \leq |Y| \leq l''$, then for every $p \in Y$

   $$\langle\langle s_1, \ldots, s_w, p\rangle, \langle C_1, \ldots, C_w, C\rangle, \langle(l' Z l'', Y), (k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_r), (l'_1 Z_1 l''_1, T_1) \ldots, (l'_n Z_n l''_n, T_n)\rangle\rangle$$

   is a proof scheme for $P$.

Now, given a CC-proof scheme

$$\mathfrak{S} = \langle\langle s_1, \ldots, s_k\rangle, \langle C_1, \ldots, C_k\rangle, \langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_r)\rangle\rangle$$

for $P$, we say that $\mathfrak{S}$ is a CC-proof scheme for $s_k$ in $P$. We shall refer to the sequence $\langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_r)\rangle$ as the *cardinality constraint sequence* of $\mathfrak{S}$. If $M \subseteq At$, then we say that $\mathfrak{S}$ is *admitted* by $M$ if $M \cap X_1 = Y_1, \ldots, M \cap X_k = Y_k$. It is easy to see that if $M$ admits a proof scheme

$$\mathfrak{S} = \langle\langle s_1, \ldots, s_k\rangle, \langle C_1, \ldots, C_k\rangle, \langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_r)\rangle\rangle$$

then $M \models k'_1 X_1 k''_1, \ldots, M \models k'_r X_r k''_r$ since the sets $Y_i$, $i = 1, \ldots, k$ witnesses that the corresponding constraints are satisfied.

Thus a proof scheme provides a derivation of an atom and proposes a way of satisfying constraints occurring in bodies of all clauses used in that derivation. Moreover, the proof schemes for ordinary programs can be easily transformed into the CC-proof schemes for the corresponding cardinality-constraint program. That is, instead of having an element $r$ be in set of constraints in the third component of a proof scheme for an ordinary logic program, we simply add a pair $(\{r\}0, \emptyset)$ to the cardinality constraint sequence of the corresponding CC-proof scheme because a set $M \subseteq At$ will admit such a proof scheme if and only if $r \notin M$.

**Example 2.2** Let $P$ be the following CC-logic program:

$$C_1 = 1\{p, q\}2 \leftarrow r, \{t\}0$$
$$C_2 = r \leftarrow \{s\}0$$
$$C_3 = s \leftarrow \{r\}0$$

The CC-program $P$ has four stable models: $M_1 = \{r, p\}$, $M_2 = \{r, q\}$, $M_3 = \{r, p, q\}$ and $M_4 = \{s\}$. $M_1$ and $M_2$ are included in $M_3$.

The triple $\langle\langle s\rangle, \langle C_3\rangle, \langle\{r\}0, \emptyset\rangle\rangle$ is a proof scheme for $s$ in $P$. This proof scheme is not admitted by

$M_1, M_2$ or $M_3$. It is admitted by $M_4$. The triple $\langle\langle r, p\rangle, \langle C_2, C_1\rangle, \langle(1\{p,q\}2, \{p\}), (\{s\}0, \emptyset), (\{t\}0, \emptyset)\rangle\rangle$ is admitted by $M_1$, but not by $M_2$. Also, the scheme $\mathfrak{S}_3$

$$\langle\langle r, p, q\rangle, \langle C_2, C_1, C_1\rangle, \langle(1\{p,q\}2, \{p,q\}), (\{s\}0, \emptyset), (\{t\}0, \emptyset)\rangle\rangle$$

is admitted by $M_3$ but not by $M_1$, because atom $q$ does not belong to $M_1$. Notice that clause $C_1$ is used in $\mathfrak{S}_3$ twice, once to derive $p$ and again to derive $q$. □

This given, we can prove the following analogue of Corollary 2.2.

**Proposition 2.3** *Let $P$ be a CC-logic program, and let $M \subseteq At$, $M \models P$. Then $M$ is a stable model of $P$ if and only if every element $p$ of $M$ possesses a proof scheme $\mathfrak{S}_p$ such that $\mathfrak{S}_p$ is admitted by $M$.*

As Proposition 2.3 is a natural generalization of Corollary 2.2, it provides an evidence that the stable models of [NSS99] correctly generalize the ordinary stable models.

## 3 Reduction of CC-logic programs to ordinary programs

In this section we construct a transformation of CC-logic programs to ordinary logic programs with the desired property of "lifting" CC-stable models of a CC-logic program to stable models of its image. Specifically, for every CC-logic program $P$ we define an ordinary logic program $t(P)$ with the following properties:

1. For every stable model $N$ of $t(P)$, the set $N \cap At$ is a CC-stable model of $P$
2. Every CC-stable model $M$ of $P$ is of the form $N \cap At$ for a unique stable model $N$ of $t(P)$.

Thus the correspondence between the CC-stable models $M$ of $P$ and the stable models $N$ of $t(P)$ will be one-to-one.

Let $P$ be a CC-logic program and let $At$ be the set of all ordinary atoms occurring in $P$. Let $\mathcal{X}$ be the set of all cardinality constraints, that is, the expressions of the form $kXl$, occurring in $P$. Here $k \in \omega$, $X$ is a finite subset of $At$, $l \in \omega \cup \{\infty\}$, and $k \leq l$.

Next we describe the language of $t(P)$. For each such cardinality constraint $kXl$, we add two new atoms $Q^b_{k,l,X}$ and $Q^h_{k,l,X}$ to the language of $t(P)$. Here the superscript $b$ stands for *body* and the superscript $h$ stands for *head.* We need two atoms for $kXl$ because cardinality constraint atoms play different roles in CC-programs depending on whether they are in the body or the head. For each $kXl \in \mathcal{X}$ and every atom $a \in At$, we add two new atoms, $P_{k,l,X,a}$ and $\bar{P}_{k,l,X,a}$, to the language of $t(P)$. Finally we add one additional new atom, $f$, which will help us write various constraint clauses. The idea is that $f$ will always be used in the context $f \leftarrow \mathit{condition}, \neg f$ where *condition* is set of atoms or constraint conditions. Moreover, these will be the only type of clauses that have $f$ in the head. This means that $f$ cannot be part of any stable model $M$ of $t(P)$ since the only way to derive $f$ from $t(P)^M$ would be to have $f \notin M$. Thus clauses of the form $f \leftarrow \mathit{condition}, \neg f$ state that *condition* cannot hold in a stable model[2].

Next we describe the program $t(P)$. It will consist of several groups of rules. In addition to a suitably modified copy of the program $P$ itself, it will have additional rules used to enforce the correct interpretation of the new atoms that we added to the language of $t(P)$. The program $t(P)$ consists of seven groups of rules.

(I) (Translation) Substitute in $P$, $Q^b_{k,l,X}$ for every occurrence of the cardinality constraint $kXl$ in the body of a clause and $Q^h_{k,l,X}$ for every occurrence of $kXl$ in the head of a clause. The result of this translation is an ordinary, in fact a Horn, program.

(II) (Constraining size of each $X$) For each atom $Q^b_{k,l,X}$ such that $l \neq \infty$, for each pair of disjoint sets $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_{|X|-l}\}$ contained in $X$, add the following clause to $t(P)$.

$$Q^b_{k,l,X} \leftarrow a_1, \ldots, a_k, \neg b_1, \ldots, \neg b_{|X|-l}$$

For each atom $Q^b_{k,l,X}$ such that $l = \infty$ and for each $\{a_1, \ldots, a_k\} \subseteq X$, add the following clause to $t(P)$.

$$Q^b_{k,l,X} \leftarrow a_1, \ldots, a_k$$

(III) (Constraining $Q^c_{k,l,X}$ to behave properly)

(III1) For each atom $Q^c_{k,l,X}$ with $c \in \{b,h\}$ and $l \in \omega$ and every set of $l+1$ elements $a_1, \ldots, a_{l+1}$ in $X$ add the following clause to $t(P)$.

$$f \leftarrow P_{k,l,X,a_1}, \ldots, P_{k,l,X,a_{l+1}}, Q^c_{k,l,X}, \neg f$$

(III2) For each atom $Q^c_{k,l,X}$ where $c \in \{b,h\}$ and every set of $|X|-k+1$ elements $b_1, \ldots, b_{|X|-l+1}$ in $X$, add the following clause to $t(P)$.

$$f \leftarrow \bar{P}_{k,l,X,b_1}, \ldots, \bar{P}_{k,l,X,b_{|X|-l+1}}, Q^c_{k,l,X}, \neg f$$

[2] Of course, when writing *smodels* programs we do not need to use the special symbol $f$ - instead of $f \leftarrow \mathit{condition}, \neg f$ we would simply write $: -\mathit{condition}$.

(IV) (Choice) For each $kXl \in \mathcal{X}$ and $a \in At$, add the following clauses to $t(P)$ for both $c = b$ and $c = h$.

(IV1) $P_{k,l,X,a} \leftarrow \neg \bar{P}_{k,l,X,a}, Q^c_{k,l,X}$

(IV2) $\bar{P}_{k,l,X,a} \leftarrow \neg P_{k,l,X,a}, Q^c_{k,l,X}$

(IV3) $f \leftarrow \bar{P}_{k,l,X,a}, P_{k,l,X,a}, Q^c_{k,l,X}, \neg f$

(V) (No extraneous atoms in $X$) For $kXl \in \mathcal{X}$ and $a \in At \setminus X$, add the following clause to $t(P)$ for both $c = b$ and $c = h$.

$$f \leftarrow P_{k,l,X,a}, Q^c_{k,l,X}, \neg f$$

(VI) (Alternative way of deriving ordinary atoms) For each $kXl \in \mathcal{X}$ and $a \in At$, add the following clause to $t(P)$.

$$a \leftarrow P_{k,l,X,a}, Q^h_{k,l,X}$$

(VII) (Limiting choice of additional atoms via ordinary atoms) For each $kXl \in \mathcal{X}$ and $a \in At$, add the following clauses to $t(P)$ for both $c = b$ and $c = h$.

(VII1) $P_{k,l,X,a} \leftarrow a, Q^c_{k,l,X}$

(VII2) $\bar{P}_{k,l,X,a} \leftarrow \neg a, Q^c_{k,l,X}$

The intention of the new atoms in the program $t(P)$ is as follows. First the atom $Q^b_{k,l,X}$ replaces the cardinality constraint $kXl$ in bodies of clauses and $Q^h_{k,l,X}$ replaces $kXl$ in the head of clauses. If $M$ is a stable model of $t(P)$, $c \in \{b, h\}$ and $Q^c_{k,l,X} \in M$, then our intention is that $P_{k,l,X,a}$ holds if $a \in M$ and $\bar{P}_{k,l,X}$ holds if $a \notin M$.

Observe that an atom $Q^h_{k,l,X}$ can be derived via a translation of a clause in $P$. Similarly an atom $Q^b_{k,l,X}$ can be derived via an application of a clause in (II) which can happen only if $k \leq |M \cap X| \leq l$. This reflects exactly how $kXl$ can participate in CC-proof schemes in the original CC-logic program $P$.

Atoms of the form $P_{k,l,X,a}$ and $\bar{P}_{k,l,X,a}$ can also be derived relative to $t(P)^M$ in two ways assuming that $Q^c_{k,l,X} \in M$ where $c \in \{b, h\}$. That is, the clauses of type (IV) say that we can derive $P_{k,l,X,a}$ from the absence of $\bar{P}_{k,l,X,a}$ and we can derive $\bar{P}_{k,l,X,a}$ from the absence of $P_{k,l,X,a}$. However the clauses of the type (IV3) ensure that it can not be the case that both $\bar{P}_{k,l,X,a}$ and $P_{k,l,X,a}$ hold. Then clauses of type (VII) ensure that $P_{k,l,X,a}$ holds if $a \in M$ and $\bar{P}_{k,l,X,a}$ if $a \notin M$ and the clauses of type (V) ensure that $P_{k,l,X,a}$ holds only if $a \in X$.

More generally, we can summarize the role the various sets of clauses as follows. The clauses of type (I) are just translations of clauses of $P$ except that they treat cardinality constraints as new atoms. The remaining clauses (groups II–VII) make sure that the semantics assigned to cardinality constraints $Q^c_{k,l,X}$ is correct. The clauses of type (II) ensure that for any stable model $M$ of $t(P)$, if $k \leq |M \cap X| \leq l$, then $Q^b_{k,l,X}$ can be derived. The clauses of type (IV) and (V) ensure that $\{a : P_{k,l,X,a} \in M\}$ is a subset of $X$. The clauses of type (III) then ensure that if $Q^c_{k,l,X} \in M$ for $c \in \{b, h\}$, then $k \leq |\{a : P_{k,l,X,a} \in M\}| \leq l$. These clauses combined with the clauses of type (VI) and (VII), then ensure that if $Q^c_{k,l,X} \in M$, then $k \leq |M \cap X| \leq l$.

This given, we then have the following theorem.

**Theorem 3.1** *Let $P$ be a CC-logic program, $At$ denote the set of atoms that occur in $P$ and $t(P)$ be the ordinary logic program consisting of clauses (I)–(VII). Then the mapping $N \mapsto M = N \cap At$ is a one-to-one projection from the set of stable models of $t(P)$ to the set of CC-stable models of $P$.*

We will briefly indicate how Theorem 3.1 is proven.

First we must show that given a CC-stable model $M$ of $P$, there is a unique stable model $N_M$ of $t(P)$ such that $N_M \cap At = M$. Given our discussion above, it is clear what $N_M$ has to be. That is, let $\mathcal{X}^b_M = \{kXL \in \mathcal{X} : k \leq |M \cap X| \leq l\}$. Similarly let $\mathcal{X}^h_M$ denote the set of all $kXl \in \mathcal{X}$ such that there is a CC-proof scheme

$$\mathfrak{S} = \langle\langle s_1, \ldots, s_w\rangle, \langle C_1, \ldots, C_k\rangle, \\ \langle(k'_1 X_1 k''_1, Y_1), \ldots, (k'_r X_r k''_r, Y_r), \rangle\rangle$$

admitted by $M$ where the clause $C_k$ has its head equal to $kXL$. Then

$$\begin{array}{rcl} N_M & = & M \cup \{Q^b_{k,l,X} : kXl \in \mathcal{X}^b_M\} \cup \\ & & \{Q^h_{k,l,X} : kXl \in \mathcal{X}^b_M\} \cup \\ & & \{P_{k,l,X,a} : kXl \in \mathcal{X}^b_M \ \&\ a \in M\} \cup \\ & & \{\bar{P}_{k,l,X,a} : kXl \in \mathcal{X}^b_M \ \&\ a \notin M\}. \end{array}$$

The key idea of the proof of the fact that $N_M$ is a stable model of $t(P)$ is to show that every CC-proof scheme admitted by $M$ can be expanded to a proof scheme admitted by $N_M$. This argument relies on the fact that $M$ is a model of $P$. Then it is easy to see that the clauses in (VII) and (III) will allow us to easily derive the rest of the atoms in $N_M$.

Conversely, suppose that we are given stable model $N$ of $t(P)$. Then let $M = N \cap At$. First it is easy to see from the clauses of type (II) that $Q^b_{k,l,X} \in M$ if $k \leq |M \cap X| \leq l$. Next the clauses of type (III), (IV), (V), and (VIII) ensure that for $c \in \{b, h\}$, $Q^c_{k,l,X} \in N$ only if $k \leq |M \cap X| \leq l$. Thus $Q^b_{k,l,X} \in M$ if and only if $k \leq |M \cap X| \leq l$. It then follows that $M$ is a model of $P$. That is, if there is some clause $C$ such that

$$C = p \leftarrow q_1, \ldots, q_m, k_1 X_1 l_1, \ldots, k_n X_n l_n$$

such that $M$ satisfies the body of $C$, then for all $1 \leq i \leq n$, $k_i \leq |M \cap X_i| \leq l_i$ so that $Q^b_{k_i,l_i,X_i} \in N$. But then the clause $C'$ in $t(P)$ where

$$C' = p \leftarrow q_1, \ldots, q_m, Q^b_{k_1,l_1,X_1}, \ldots, Q^b_{k_n,l_n,X_n}.$$

Since $N$ is a stable model, $p$ must be in $N$ and hence $p \in M$. Similarly if there is a clause $D$ such that

$$D = kXl \leftarrow q_1, \ldots, q_m, k_1 X_1 l_1, \ldots, k_n X_n l_n$$

such that $M$ satisfies the body of $D$, then again we can conclude that $Q^b_{k_i,l_i,X_i} \in N$ for $i = 1, \ldots, n$. But then there is a clause $D'$ is in $t(P)$ such that

$$D' = Q^h_{k,l,X} \leftarrow q_1, \ldots, q_m, Q^b_{k_1,l_1,X_1}, \ldots, Q^b_{k_n,l_n,X_n}.$$

Thus since $N$ is a stable model, $Q^h_{k,l,X} \in N$ and hence $k \leq |M \cap X| \leq l$. Thus $M$ satisfies $kXl$.

Finally one can prove by induction on the length of proof schemes that an atom $a \in At$ is in $M$ if only if there is CC-proof scheme for $P$ of $a$ admitted by $M$. That is, suppose

$$\mathfrak{S} = \langle \langle s_1, \ldots, s_w, a \rangle, \langle C_1, \ldots, C_k, C \rangle, \{r_1, \ldots, r_k\} \rangle$$

is a proof scheme for $a$ admitted by $N$. Then there are two cases to consider:

(I) $C = a \leftarrow q_1, \ldots, q_m, Q^b_{k_1,l_1,X_1}, \ldots, Q^b_{k_n,l_n,X_n}$ where

$$C' = a \leftarrow q_1, \ldots, q_m, k_1 X_1 l_1, \ldots, k_n X_n l_n$$

is a clause in $P$ or

(II) $C = a \leftarrow P_{k,l,X,a}, Q^h_{k,l,X}$.

In case (I), it must be case that $\{q_1, \ldots q_m\} \subseteq \{s_1, \ldots, s_k\}$. Hence they all have shorter proof schemes admitted by $N$ so that, for $i = 1, \ldots, m$, there are CC-proof schemes $\mathfrak{S}_i$ for $q_i$ admitted by $M$. Since the only rules that have $Q^b_{k_j,l_j,X_j}$ in the head are of type (II), it follows that for each $j = 1, \ldots, n$, there must be a rule $C_{t_j}$ with $t_j \leq k$ of the form

$$C_{t_j} = Q^b_{k_j,l_j,X_j} \leftarrow a^j_1, \ldots, a^j_{r_j}, \neg b^j_1, \ldots, \neg b^j_{s_j}.$$

Again it must be the case that $\{a^j_1, \ldots a^j_{r_j}\} \subseteq \{s_1, \ldots, s_k\}$. Hence they all have shorter proof schemes admitted by $N$ so that, for $i = 1, \ldots, r_j$, there are CC-proof schemes $\mathfrak{T}^j_i$ for $a^j_i$ admitted by $M$. We can then easily combine all these CC-proofs schemes admitted by $M$ together with the rule $C'$ to construct a CC-proof scheme for $a$ admitted by $M$.

Similarly in case (II), it must be the case some $t \leq k$ such that (I) $C_t = Q^h_{k,l,X} \leftarrow q_1, \ldots, q_m, Q^b_{k_1,l_1,X_1}, \ldots, Q^b_{k_n,l_n,X_n}$ where

$$D_t = kXl \leftarrow q_1, \ldots, q_m, k_1 X_1 l_1, \ldots, k_n X_n l_n$$

is a clause in $P$. Again it must be case that $\{q_1, \ldots q_m\} \subseteq \{s_1, \ldots, s_k\}$. Hence they all have shorter proof schemes admitted by $N$ so that, for $i = 1, \ldots, m$, there are CC-proof schemes $\mathfrak{S}_i$ for $q_i$ admitted by $M$. Since the only rules that have $Q^b_{k_j,l_j,X_j}$ in the head are of type (II), it follows that for each $j = 1, \ldots, n$, there must be a rule $C_{t_j}$ with $t_j \leq t$ of the form

$$C_{t_j} = Q^b_{k_j,l_j,X_j} \leftarrow a^j_1, \ldots, a^j_{r_j}, \neg b^j_1, \ldots, \neg b^j_{s_j}.$$

Again it must be the case that $\{a^j_1, \ldots a^j_{r_j}\} \subseteq \{s_1, \ldots, s_k\}$. Hence they all have shorter proof schemes admitted by $N$ so that, for $i = 1, \ldots, r_j$, there are CC-proof schemes $\mathfrak{T}^j_i$ for $a^j_i$ admitted by $M$. We can then easily combine all these CC-proofs schemes admitted by $M$ together with the rule $D_t$ to construct a CC-proof scheme for $a$ admitted by $M$. □

## 4 Some complexity issues

We will now investigate some complexity issues related to CC-logic programs. In [NSS99], Niemelä, Simons and Soininen show that the existence problems for stable models of CC-logic programs is NP-complete. In fact, as we will see, a much smaller class of CC-logic programs has this property. In [FMT02], a class of *generator* CC-logic programs is introduced which consists of all CC-logic programs $P$ such that each clause $C$ of $P$ is a single fact, i.e. $C$ is of the form $p \leftarrow$, or of the form $kXl \leftarrow$. A *generator* for a set $At$ is a generator CC-logic program $P$ such that every atom in $At$ occurs in some clause of $P$. The following fact has been proved by M. Truszczyński[FMT02].

**Proposition 4.1** *Let $P$ be a generator for the set of atoms $At$. Then every model of $P$ is a CC-stable model of $P$.*

We notice in passing that Proposition 4.1 follows immediately from Proposition 2.3.

Next we observe the following

**Proposition 4.2** *The existence problem for stable models of generator CC-logic programs is NP-complete.*

Proposition 4.2 follows from the existence of the reduction of the VERTEX COVER problem to the existence problem for CC-stable models of generator CC-logic programs. Indeed, let $G = \langle V, E\rangle$ be a graph and $k \in N$. Consider the following generator CC-logic program.

$$1\{x, y\}2 \leftarrow$$
$$Vk \leftarrow$$

Here the first clause is added for every edge $(x, y) \in E$. Moreover, $V$ is identified with the set of atoms $At$. Call the resulting program $P_{G,k}$. It is then easy to see that models (and thus CC-stable models) of $P_{G,k}$ are vertex covers for $G$ of size at most $k$.

Thus even the existence of models or CC-stable models for CC-logic programs and for generator CC-logic programs is NP-complete.

Finally, we will consider a slightly larger class of CC-logic programs $P$ where the body of clause $C$ of $P$ contains no cardinality constraints. That is, $C$ is of the form

$$p \leftarrow q_1, \ldots, q_m$$

or $C$ is of the form

$$kXl \leftarrow q_1, \ldots, q_m.$$

We call such programs *semi-generator* CC-logic programs.

We will now show how to reduce the satisfiability problem for propositional logic to the existence problem for CC-stable models of semi-generator CC-logic programs. To this end, given a CNF formula $\Phi = C_1 \wedge \ldots \wedge C_m$, we will write a semi-generator CC-logic program $P_\Phi$ as follows. First let $S$ denote the set of propositional letters that occur in $\Phi$. For each $s \in S$, let $d(s) = \bar{s}$ and $d(\neg s) = s$. Next let $p_0$ be some fixed element in $S$ and let $T = \{\bar{p} : p \in S\} \cup S$. The $P_\Phi$ consists of the following set of clauses.

(1) $1\{p, \bar{p}\}1 \leftarrow$

(2) $2\{p_0, \overline{p_0}\}2 \leftarrow d(\neg l_1^i), \ldots, d(\neg l_{n_i}^i)$

The clause (1) is added for every $p \in S$. The clause (2) is added for every clause $C_i = l_1^i \vee \cdots \vee l_{n_i}^i$ in $\Phi$, $i = 1, \ldots, m$.

Note that the clauses of type (1) ensure that for any CC-stable model $M$ of $P_\Phi$, exactly one of $p$ and $\bar{p}$ is in $M$ for each $p \in S$. In particular, we can not have both $p_0$ and $\bar{p}_0$ in $M$. Thus if for all $s \in S$, if we interpret $s \in M$ as $s$ being true and $\bar{s} \in M$ as $s$ being false, then it is easy to see that clauses in (2) ensure that truth assignment determined by $M$ must satisfy all the clauses $C_i$. Then it is clear that there is a one-to-one correspondence between stable models of $P_\Phi$ and valuations of $S$ satisfying $\Phi$. Thus we have proved the following result.

**Proposition 4.3** *1. The problem consisting of pairs $\langle P, a\rangle$ where $P$ is a semi-generator CC-logic program such that $a$ belongs to some CC-stable model of $P$ is NP-complete.*

*2. The problem consisting of pairs $\langle P, a\rangle$ where $P$ is a semi-generator CC-logic program such that $a$ belongs to all CC-stable models of $P$ is co-NP-complete.*

## 5 Conclusions and further research

In this paper, we studied the CC-stable model semantics of CC-logic programs and showed they can be related to the stable logic semantics of ordinary logic programs. Indeed, we showed that one can view the CC-stable stable semantics as arising from the *hide* operation applied to an ordinary logic program with a larger language. While our investigation related CC-stable models of CC-logic programs to stable models of ordinary logic programs, other classes of models for CC-programs can also be studied. For instance, it is possible to define the corresponding $T_P$ operator [AvE82] for CC-logic programs. Its fixed points may play the role of supported models. Likewise, the well-founded semantics and the alternating fixed point construction may be of interest in the case of CC-logic programs. Our investigations indicate that there will always be at least two versions of such concepts, "direct" and "indirect." That is, we can modify a given semantics so that it can be applied in the original language of the program or we can apply the usual semantics to programs in an extended language and then project. Whether these concepts are essentially the same for semantics other than stable semantics remains to be seen. One expects that there are many natural constructs for logic programs that can be extended to CC-logic programs. For example, in another paper, we will show that there are natural analogues of stratified programs for CC-logic programs.

Finally we have not tried to make our translation optimal from the complexity point of view. Instead, we opted to have the translation that would make the proof of Theorem 3.1 as simple as possible. However, complexity results for CC-logic programs indicate that there may be similar translations to one constructed in this paper but of much smaller size.


## Acknowledgments

The first author's research has been partially supported by the NSF grant IIS-0097278. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The second author's research has been partially supported by the ARO contract DAAD19-01-1-0724.

The authors express their gratitude to the anonymous referees for reading the paper and suggesting improvements.


## References


[Ap90] K.R. Apt. Logic Programming. in *Handbook of Theoretical Computer Science*, pages 475–574. Elsevier, 1990.

[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.

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

[FL01] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. Available at: `http://www.cs.utexas.edu/users/vl/papers.html`, Unpublished draft, 2001.

[FMT02] R. Finkel, V. Marek, and M. Truszczynski. Constraint Lingo: Towards high-level answer-set programming, Submitted for publication.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[GL88] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[KS99] H.A. Kautz and B. Selman. Unifying sat-based and graph-based planning. *Proceedings of International Joint Conference on Artificial Intelligence*, pages 318-325. Morgan Kaufmann, 1999.

[Lif99] V. Lifschitz. Answer set planning. *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning, El Paso, TX., Springer Lecture Notes in Computer Science* 1730, pages 373–374, 1999.

[LTT99] V. Lifschitz, L.R. Tang, and H. Turner. Nested expressions in logic programs, *Annals of Mathematics and Artificial Intelligence* 25:369–389, 1999.

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

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

[Nie98] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79, 1998.

[NS97] I. Niemelä and P. Simons. Smodels — an implementation of the stable model and well-founded semantics for normal logic programs, *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning, Dagstuhl, Germany, Springer Lecture Notes in Computer Science* 1265, 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. *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning, El Paso, TX.*, *Springer Lecture Notes in Computer Science* 1730, 1999.

[SK92] B. Selman and H. A. Kautz. Planning as satisfiability. *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359-363. Wiley, 1992.

[Syr01] T. Syrjänen. Manual of Lparse version 1.0, `http://saturn.tcs.hut.fi/Software/smodels`

[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.