Logic programs with monotone cardinality atoms

Victor W. Marek¹, Ilkka Niemelä², and Miroslaw Truszczyński¹

¹ Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA
² Department of Computer Science and Engineering, Helsinki University of Technology, P.O.Box 5400, FIN-02015 HUT, Finland

Abstract. We investigate mca-programs, that is, logic programs with clauses built of monotone cardinality atoms of the form kX, where k is a non-negative integer and X is a finite set of propositional atoms. We develop a theory of mca-programs. We demonstrate that the operational concept of the one-step provability operator generalizes to mca-programs, but the generalization involves nondeterminism. Our main results show that the formalism of mca-programs is a common generalization of (1) normal logic programming with its semantics of models, supported models and stable models, (2) logic programming with cardinality atoms and with the semantics of stable models, as defined by Niemelä, Simons and Soiennien, and (3) of disjunctive logic programming with the possible-model semantics of Sakama and Inoue.

1 Introduction

We introduce and study logic programs whose clauses are built of monotone cardinality atoms (mc-atoms), that is, expressions of the form kX, where k is a non-negative integer and X is a finite set of propositional atoms. Intuitively, kX is true in an interpretation M if at least k atoms in X are true in M. Thus, the intended role for mc-atoms is to represent constraints on lower bounds of cardinalities of sets. We refer to programs with mc-atoms as mca-programs. We are motivated in this work by the recent emergence and demonstrated effectiveness of logic programming extended with means to model cardinality constraints [12, 11, 15], and by the need to develop a sound theoretical basis for such formalisms.

In the paper, we develop a theory of mca-programs. In that we closely follow the development of normal logic programming and lift all its major concepts, techniques and results to the setting of mca-programs. There is, however, a basic difference. Mc-atoms have, by their very nature, a built-in nondeterminism. They can be viewed as shorthands for certain disjunctions and, in general, there are many ways to make an mc-atom kX true. This nondeterminism has a key consequence. The one-step provability operator is no longer deterministic, as in normal logic programming, where it maps interpretations to interpretations. In the case of mca-programs, the one-step provability operator is nondeterministic. It assigns to an interpretation $M$ a set of interpretations, each regarded as possible and equally likely outcomes of applying the operator to $M$. 
Modulo this difference, our theory of mca-programs parallels that of normal logic programs. First, we introduce models and supported models of an mca-program and describe them in terms of the one-step provability operator in much the same way it is done in normal logic programming. To define stable models we first define the class of Horn mca-programs by disallowing the negation operator in the bodies of clauses. We show that the nondeterministic one-step provability operator associates with Horn mca-programs a notion of a (nondeterministic) computation (the counterpart to the bottom-up computation with normal Horn programs) and a class of derivable models (counterparts to the least model of a normal Horn program). We then lift the notion of the Gelfond-Lifschitz reduct \cite{GelfondLifschitz} to the case of mca-programs and define a stable model of an mca-program as a set of atoms that is a derivable model of the reduct. A striking aspect of our construction is that all its steps are literal extensions of the corresponding steps in the original approach. We show that stable models behave as expected. They are supported and, in case of Horn mca-programs, derivable.

An intended meaning of an mc-atom $1\{a\}$ is that $a$ be true. More formally, $1\{a\}$ is true in an interpretation if and only if $a$ is true in that interpretation. That connection implies a natural representation of normal logic programs as mca-programs. We show that this representation preserves all semantics we discuss in the paper. It follows that the formalism of mca-programs can be viewed as a direct generalization of normal logic programming.

As we noted, an extension of logic programming with direct ways to model cardinality constraints was first proposed in \cite{Hayes}. That work defined a syntax of logic programs with cardinality constraints (in fact, with more general weight constraints) and introduced the notion of a stable model. We will refer to programs in that formalism as NSS-programs. One of the results in \cite{Hayes} showed that NSS-programs generalized normal logic programming with the stable-model semantics of Gelfond and Lifschitz \cite{GelfondLifschitz}. However, the notion of the reduct underlying the definition of a stable model given in \cite{Hayes} is different from that proposed by Gelfond and Lifschitz \cite{GelfondLifschitz} and the precise nature of the relationship between normal logic programs and NSS-programs was not clear.

Mca-programs explicate this relationship. We show that the formalism of mca-programs parallels normal logic programming. In particular, major concepts, results and techniques in normal logic programming have counterparts in the setting of mca-programs. We also prove that under some simple transformations, NSS-programs are equivalent to mca-programs. Through this connection, the theory of normal logic programming can be lifted to the setting of NSS-programs leading to new characterizations of stable models of NSS-programs.

Finally, we show that mca-programs not only provide an overarching framework for both normal logic programs and NSS-programs. They are also useful in investigating disjunctive logic programs. In the paper, we show that logic programming with mc-atoms generalize disjunctive logic programming with the possible-model semantics introduced in \cite{Barwise}. 

2 Logic programs with monotone cardinality atoms

Let $At$ be a set of (propositional) atoms. An mc-atom over $At$ (short for a monotone cardinality atom over $At$) is any expression of the form $kX$, where $k$ is a non-negative integer and $X \subseteq At$ is a finite set such that $k \leq |X|$. We call $X$ the atom set of an mc-atom $A = kX$ and denote it by $\text{aset}(A)$. An intuitive reading of an mc-atom $kX$ is: at least $k$ atoms in $X$ are true. The intended meaning of $kX$ explains the requirement that $k \leq |X|$. Clearly, if $k > |X|$, it is impossible to have in $X$ at least $k$ true atoms and the expression $kX$ is equivalent to a contradiction.

An mc-literal is an expression of the form $A$ or $\text{not}(A)$, where $A$ is an mc-atom. An mc-clause (short for a monotone-cardinality-atom clause) is an expression $r$ of the form

$$H \leftarrow L_1, \ldots, L_m,$$

(1)

where $H$ is an mc-atom and $L_i$, $1 \leq i \leq m$, are mc-literals. We call the mc-atom $H$ the head of $r$ and denote it by $\text{hd}(r)$. We call the set $\{L_1, \ldots, L_m\}$ the body of $r$ and denote it by $\text{bd}(r)$. An mc-clause is Horn if its body does not contain literals of the form $\text{not}(A)$. Finally, for an mc-clause $r$, we define the head set of $r$, $\text{hset}(r)$, by setting $\text{hset}(r) = \text{aset}(\text{hd}(r))$.

Mca-clauses form mca-programs. We define the head set of an mca-program $P$, $\text{hset}(P)$, by $\text{hset}(P) = \bigcup \{\text{hset}(r) : r \in P\}$ (if $P = \emptyset$, $\text{hset}(P) = \emptyset$, as well). If all clauses in an mca-program $P$ are Horn, $P$ is a Horn mca-program.

One can give a declarative interpretation to mca-programs in terms of a natural extension of the semantics of propositional logic. We say that a set $M$ of atoms satisfies an mc-atom $kX$ if $|M \cap X| \geq k$, and $M$ satisfies an mc-literal $\text{not}(kX)$ if it does not satisfy $kX$ (that is, if $|M \cap X| < k$). A set of atoms $M$ satisfies an mca-clause (1) if $M$ satisfies $H$ whenever $M$ satisfies all literals $L_i$, $1 \leq i \leq m$. Finally, a set of atoms $M$ satisfies an mca-program $P$ if it satisfies all clauses in $P$. We often say “is a model of” instead of “satisfies”. We use the symbol $\models$ to denote the satisfaction relation.

The following straightforward property of mc-atoms explains the use of the term “monotone” in their name.

**Proposition 1.** Let $A$ be an mc-atom over a set of atoms $At$. For every sets $M, M' \subseteq At$, if $M \subseteq M'$ and $M \models A$ then $M' \models A$.

Mca-clauses also have a procedural interpretation in which they are viewed as derivation rules. Intuitively, if an mca-clause $r$ has its body satisfied by some set of atoms $M$, then $r$ provides support for deriving from $M$ any set of atoms $M'$ such that

1. $M'$ consists of atoms mentioned in the head of $r$ (r provides no grounds for deriving atoms that do not appear in its head)
2. $M'$ satisfies the head of $r$ (since $r$ “fires”, the constraint imposed by its head must hold).
Clearly, the process of deriving \( M' \) from \( M \) by means of \( r \) is **nondeterministic** in the sense that, in general, there are several sets that are supported by \( r \) and \( M \).

This notion of nondeterministic derivability extends to programs and leads to the concept of the nondeterministic one-step provability operator. Let \( P \) be an mca-program and let \( M \subseteq At \) be a set of atoms. We set \( P(M) = \{ r \in P : M \models \text{bd}(r) \} \). We call mca-clauses in \( P(M) \), **M-applicable**.

**Definition 1.** Let \( P \) be an mca-program and let \( M \subseteq At \). A set \( M' \) is nondeterministically one-step provable from \( M \) by means of \( P \), if \( M' \subseteq \text{hset}(P(M)) \) and \( M' \models \text{bd}(r) \), for every mca-clause \( r \) in \( P(M) \).

The nondeterministic one-step provability operator \( T^d_P \), is a function from \( \mathcal{P}(At) \) to \( \mathcal{P}(\mathcal{P}(At)) \) and such that for every \( M \subseteq At \), \( T^d_P(M) \) consists all sets \( M' \) that are nondeterministically one-step provable from \( M \) by means of \( P \).

As we indicate next, for every \( M \subseteq At \), \( T^d_P(M) \) is nonempty. It follows that \( T^d_P \) can be viewed as a formal representation of a **nondeterministic** operator on \( \mathcal{P}(At) \), which assigns to every subset \( M \) of \( At \) a subset of \( At \) arbitrarily selected from the collection \( T^d_P(M) \) of possible outcomes. Since \( T^d_P(M) \) is nonempty, this nondeterministic operator is well defined.

**Proposition 2.** Let \( P \) be an mca-program and let \( M \subseteq At \). Then, \( \text{hset}(P(M)) \in T^d_P(M) \). In particular, \( T^d_P(M) \neq \emptyset \).

The operator \( T^d_P \) plays a fundamental role in our research. It allows us to formalize procedural interpretations of mca-clauses and identify for them matching classes of models that provide the corresponding declarative account.

Our first result characterizes models of mca-programs. This characterization is a generalization of the familiar description of models of normal logic programs as prefixpoints of \( T_P \).

**Theorem 1.** Let \( P \) be an mca-program and let \( M \subseteq At \). The set \( M \) is a model of \( P \) if and only if there is \( M' \in T^d_P(M) \) such that \( M' \subseteq M \).

A straightforward corollary states that every mca-program has a model.

**Corollary 1.** Let \( P \) be an mca-program. Then, \( \text{hset}(P) \) is a model of \( P \).

Models of mca-programs may contain elements that have no support in a program and the model itself. For instance, let us consider an mca-program \( P \) consisting of the clause: \( 1\{p,q\} \leftarrow \text{not}(1\{q\}) \), where \( p \) and \( q \) are two different atoms. Let \( M_1 = \{q\} \). Clearly, \( M_1 \) is a model of \( P \). However, \( M_1 \) has no support in \( P \) and itself. Indeed, \( T^d_P(M_1) = \emptyset \) and so, \( P \) and \( M_1 \) do not provide support for any atom. Similarly, another model of \( P \), the set \( M_2 = \{p,r\} \), where \( r \in At \) is an atom different from \( p \) and \( q \), has no support in \( P \) and itself. We have \( T^d_P(M_2) = \{\{p\}, \{q\}, \{p,q\}\} \) and so, \( p \) has support in \( P \) and \( M_2 \), but \( r \) does not. Finally, the set \( M_3 = \{p\} \), which is also a model of \( P \), has support in \( P \) and itself. Indeed, \( T^d_P(M_3) = \{\{p\}, \{q\}, \{p,q\}\} \) and there is a way to derive \( M_3 \) from \( P \) and \( M_3 \). We formalize now this discussion in the following definition.
Definition 2. Let \( P \) be an mca-program. A set of atoms \( M \) is a supported model of \( P \) if \( M \in T^p_+(M) \).

The use of the term “model” is justified. By Theorem 1, supported models of \( P \) are indeed models of \( P \), as stated in the following result.

Corollary 2. Every supported model of an mca-program \( P \) is a model of \( P \).

Finally, we have the following characterization of supported models.

Proposition 3. Let \( P \) be an mca-program. A set \( M \subseteq \text{At} \) is a supported model of \( P \) if and only if \( M \) is a model of \( P \) and \( M \subseteq hset(P(M)) \).

3 Horn mca-programs

To introduce stable models of mca-programs, we need first to study Horn mca-programs. With each Horn mca-program \( P \) one can associate the concept of a \( P \)-computation. Namely, a \( P \)-computation is a sequence \( (X_n)_{n=0,1,...} \) such that \( X_0 = \emptyset \) and, for every non-negative integer \( n \),

1. \( X_n \subseteq X_{n+1} \), and
2. \( X_{n+1} \in T^p_+(X_n) \).

Given a computation \( t = (X_n)_{n=0,1,...} \), we call \( \bigcup_{n=0}^{\infty} X_n \) the result of the computation \( t \) and denote it by \( R_t \).

Proposition 4. Let \( P \) be a Horn mca-program and let \( t \) be a \( P \)-computation. Then \( R_t \subseteq hset(P(R_t)) \).

If \( P \) is a Horn mca-program then \( P \)-computations exist. Let \( M \) be a model of \( P \). We define the sequence \( t^{P,M} = (X_n^{P,M})_{n=0,1,...} \) as follows. We set \( X_0^{P,M} = \emptyset \) and, for every \( n \geq 0 \), \( X_{n+1}^{P,M} = hset(P(X_n^{P,M})) \cap M \).

Theorem 2. Let \( P \) be a Horn mca-program and let \( M \subseteq \text{At} \) be its model. The sequence \( t^{P,M} \) is a \( P \)-computation.

We call the \( P \)-computation \( t^{P,M} \) the canonical \( P \)-computation for \( M \). Since every mca-program \( P \) has models, we obtain the following corollary.

Corollary 3. Every Horn mca-program has at least one computation.

The results of computations are supported models (and, thus, also models) of Horn mca-programs.

Proposition 5. Let \( P \) be a Horn mca-program and let \( t \) be a \( P \)-computation. Then, the result of \( t \), \( R_t \), is a supported model of \( P \).

We use the concept of a computation to identify a certain class of models of Horn mca-programs.
Definition 3. Let $P$ be a Horn mca-program. We say that a set of atoms $M$ is a derivable model of $P$ if there exists a $P$-computation $t$ such that $M = R_t$.

Derivable models can be obtained as results of their own canonical computations.

Proposition 6. Let $M$ be a derivable model of a Horn mca-program $P$. Then $M = R_{M,M}$.

Proposition 5 and Theorem 2 entail several properties of Horn mca-programs, their computations and models. We gather them in the following corollary.

Corollary 4. Let $P$ be a Horn mca-program. Then:

1. $P$ has at least one derivable model.
2. $P$ has a largest derivable model.
3. Every derivable model of $P$ is a supported model of $P$.
4. For every model $M$ of $P$ there is a derivable model $M'$ of $P$ such that $M' \subseteq M$.
5. Every minimal model of $P$ is derivable.

4 Stable models of mca-programs

We will now use the results of the two previous sections to introduce and study the class of stable models of mca-programs.

Definition 4. Let $P$ be an mca-program and let $M \subseteq \text{At}$. The reduct of $P$ with respect to $M$, $P^M$ in symbols, is a Horn mca-program obtained from $P$ by (1) removing from $P$ every clause containing in the body a literal $\text{not}(A)$ such that $M = A$, and (2) removing all literals of the form $\text{not}(A)$ from all remaining clauses in $P$. A set of atoms $M$ is a stable model of $P$ if $M$ is a derivable model of the reduct $P^M$.

Stable models of an mca-program $P$ are indeed models of $P$. Thus, the use of the term “model” in their name is justified. In fact, a stronger property holds: stable models of mca-programs are supported.

Proposition 7. Let $P$ be an mca-program. If $M \subseteq \text{At}$ is a stable model of $P$ then $M$ is a supported model of $P$.

With the notion of a stable model in hand, we can strengthen Proposition 5.

Proposition 8. Let $P$ be a Horn mca-program. A set of atoms $M \subseteq \text{At}$ is a derivable model of $P$ if and only if $M$ is a stable model of $P$. 
We will now describe a procedural characterization of stable models of mca-programs, relying on a notion of a computation related to but different from the one we discussed in Section 3 in the context of Horn programs. A difference is that now at each stage in a computation we must make sure that once a clause is applied, it remains applicable at any stage of the process. It is not a priori guaranteed due to the presence of negation in the bodies of general mca-clauses.

A formal definition is as follows. Let $P$ be an mca-program. A sequence $\varepsilon = (X_n)_{n=0,1,2,\ldots}$ is a quasi $P$-computation, if $X_0 = \emptyset$ and if for every $n = 0,1,\ldots$ there is a clause $r_n \in P$ such that

1. $X_n \models bd(r_n)$.
2. there is $X \subseteq hset(r_n)$ such that $X \models hd(r_n)$ and $X_{n+1} = X_n \cup X$ (this $X$ is what is “computed” by applying $r_n$).
3. for every $i = 0,1,\ldots,n$ and for every mc-atom $kX$ occurring negated in $bd(r_i)$, $X_{n+1} \not\models kX$.

We call the set $\bigcup_{1 \leq k < \omega} X_k$ the result of the quasi $P$-computation $\varepsilon$.

**Theorem 3.** A set of atoms $M$ is a stable model of $P$ if and only if $M$ is a model of $P$ and for some quasi $P$-computation $\varepsilon$, $M$ is the result of $\varepsilon$.

Theorem 3 states that if we apply clauses carefully, making sure that at no stage we satisfy an mc-atom appearing negated in clauses applied so far (including the one selected to apply at the present stage) and we ever compute a model in this way, then this model is a stable model of $P$. Conversely, every stable model can be obtained as a result of such a careful computation.

## 5 Extension of mca-programs by constraint mca-clauses

We can extend the language of mca-programs by allowing clauses with the empty head. Namely, we define a constraint mca-clause to be an expression $r$ of the form

\[ \leftarrow L_1, \ldots, L_m, \]

where $L_i$, $1 \leq i \leq m$, are mc-literals.

The notion of satisfiability that we introduced for mca-clauses extends to the case of mca-constraints. A set of atoms $M$ satisfies a constraint $r$ if there is a literal $L \in bd(r)$ such that $M \not\models L$. We can now extend the definitions of supported and stable models to the more general class of mca-programs with constraint mca-clauses as follows.

**Definition 5.** Let $P$ be an mca-program with constraint mca-clauses. A set of atoms $M$ is a supported (stable) model of $P$ if $M$ is a supported (stable) model of $P'$, where $P'$ consists of all non-constraint mca-clauses in $P$, and if $M$ is a model of all constraint mca-clauses in $P$.

Let us observe that several of our earlier results such as Proposition 7 and Theorem 3 lift verbatim to the case of programs with constraints.
6 Mca-programs and normal logic programming

An mca-atom $1\{a\}$ is true in a model $M$ if and only if $a$ is true in $M$. Thus, intuitively, $1\{a\}$ and $a$ are equivalent. That suggests a way to interpret normal clauses and programs as mca-clauses and mca-programs. Let

$$r = c \leftarrow a_1, \ldots, a_m, \mathbf{not}(b_1), \ldots, \mathbf{not}(b_n).$$

By mca($r$) we mean the mca-clause

$$1\{c\} \leftarrow 1\{a_1\}, \ldots, 1\{a_m\}, \mathbf{not}(1\{b_1\}), \ldots, \mathbf{not}(1\{b_n\}).$$

(If all $a_i$ and all $b_i$ are distinct, which we can assume without loss of generality, a simpler translation, $1\{c\} \leftarrow m\{a_1, \ldots, a_m\}, \mathbf{not}(m\{b_1, \ldots, b_n\})$, could be used.) Moreover, given a normal program $P$, we set mca($P$) = \{mca($r$): $r \in P$\}.

This encoding interprets normal logic programs as mca-programs so that basic properties and concepts of normal logic programming can be viewed as special cases of properties and concepts in mca-programming. In the following theorem, we gather several results establishing appropriate correspondences.

**Theorem 4.** Let $P$ be a normal logic program and let $M$ be a set of atoms.

1. $P$ is a Horn program if and only if mca($P$) is a Horn mca-program.
2. If $P$ is a Horn program then the least model of $P$ is the only derivable model of mca($P$).
3. \{$T_P(M)$\} = $T_{mca(P)}^d(M)$.
4. mca($P^M$) = mca($P$)$^M$.
5. $M$ is a model (supported model, stable model) of $P$ if and only if $M$ is a model (supported model, stable model) of mca($P$).

Finally, we identify a class of mca-programs, which offers a most direct generalization of normal logic programming.

**Definition 6.** An mca-clause $r$ is deterministic if hd($r$) = $1\{a\}$, for some atom $a$. An mca-program is deterministic if every clause in $P$ is deterministic.

The intuition behind the term is clear. If the head of an mca-clause is of the form $1\{a\}$, then there is only one possible effect of applying the clause: $a$ has to be concluded. Thus, the nondeterminism that arises in the context of arbitrary mca-atoms disappears. Formally, we capture this property in the following result.

**Proposition 9.** Let $P$ be a deterministic mca-program. Then, for every set of atoms $M$, $T_P^d(M) = \{M'\}$, for some set of atoms $M'$.

Thus, for a deterministic mca-program $P$, the operator $T_P^{nd}$ is deterministic and, so, can be regarded as an operator with both the domain and codomain $\mathcal{P}(At)$. We will write $T_P^d$, to denote it. Models, supported models and stable models of a deterministic mca-program can be introduced in terms of the operator $T_P^d$ in exactly the same way the corresponding concepts are defined in normal
logic programming. In particular, the algebraic treatment of logic programming developed in [7, 13, 2] applies literally to deterministic mca-programs and results in a natural and direct extension of normal logic programming. We will explicitly mention just one result here that will be of importance later in the paper.

**Proposition 10.** Let $P$ be a deterministic Horn program. Then $P$ has exactly one derivable model and this model is the least model of $P$.

### 7 Mca-programs and NSS-programs

We will first briefly review the concept of an NSS-program [12], the semantics of stable models of such programs, as introduced in [12], and then relate this formalism to that of mca-programs.

A **cardinality atom** (c-atom, for short) is an expression of the form $kXI$, where $X \subseteq At$, and $l$ and $k$ are integers such that $0 \leq k \leq l \leq |X|$. We call $X$ an atom set of a c-atom $A = kXI$ and, as before, we denote it by $\text{aset}(A)^3$.

We say that a set of atoms $M$ satisfies a c-atom $kXI$ if $k \leq |M \cap X| \leq l$ ($M \models kXI$, in symbols). It is clear that when $k = 0$ or $l = |X|$, the corresponding inequality is trivially true. Thus, we omit from the notation $k$, if equal to 0, and $l$, if equal to $|X|$.

A **cardinality-atom clause** (ca-clause, for short) is an expression $r$ of the form

$$A \leftarrow B_1, \ldots, B_n,$$

where $A$ and $B_i, 1 \leq i \leq n$, are c-atoms. We call $A$ the head of $r$ and $\{B_1, \ldots, B_n\}$ the body of $r$. We denote them by $\text{bd}(r)$ and $\text{bd}(r)$, respectively. A **ca-program** is a collection of ca-clauses.

We say that a set $M \subseteq At$ satisfies a ca-clause $r$ if $M$ satisfies $\text{bd}(r)$ whenever it satisfies each c-atom in the body of $r$. We say that $M$ satisfies a ca-program $P$ if $M$ satisfies each ca-clause in $P$. We write $M \models r$ and $M \models P$ in these cases, respectively.

We will now recall the concept of a stable model of a ca-program [12]. Let $P$ be an NSS-program and let $M \subseteq At$. By the **NSS-reduct** of $P$ with respect to $M$ we mean the NSS-program obtained by:

1. eliminating from $P$ every clause $r$ such that $M \not\models B$, for at least one c-atom $B \in \text{bd}(r)$.
2. replacing each remaining ca-clause $r = kXI \leftarrow k_1Y_1l_1, \ldots, k_nY_nl_n$ with all clauses of the form $1 \{a\} \leftarrow k_1Y_1, \ldots, k_nY_n$, where $a \in X \cap M$.

With some abuse of notation, we denote the resulting program by $P^M$ (the type of the program determines which reduc we have in mind). It is clear that $P^M$ is a deterministic Horn mca-program. Thus, it has a least model, $\text{lm}(P^M)$.

---

3 To be precise, [12] allows also for negated atoms to appear as elements of $X$. One can eliminate occurrences of negative literals by introducing new atoms. Thus, for this work, we decided to restrict the syntax of NSS-programs.
**Definition 7.** Let $P$ be a ca-program. A set $M \subseteq At$ is a stable model of $P$ if $M = \text{lm}(P^M)$ and $M \models P$.

We will now show that the formalisms of mca-programs and ca-programs with their corresponding stable-model semantics are equivalent. We start by describing an encoding of ca-clauses and ca-programs by mca-clauses and mca-programs. To simplify the description of the encoding and make it uniform, we assume that all bounds are present (we recall that whenever any of the bounds are missing from the notation, they can be introduced back). Let $r$ be the following ca-clause: $kX1 \leftarrow k_1X_1, \ldots, k_mX_m, \text{not}(l_1 + 1)X_1, \ldots, \text{not}(l_m + 1)X_m$. We represent this ca-clause by a pair of mca-clauses, $e_{mca}^1(r)$ and $e_{mca}^2(r)$ that we define as the following two mca-clauses, respectively:

$$kX \leftarrow k_1X_1, \ldots, k_mX_m, \text{not}(l_1 + 1)X_1, \ldots, \text{not}(l_m + 1)X_m,$$

and

$$\leftarrow (l_1 + 1)X_1, k_1X_1, \ldots, k_mX_m, \text{not}(l_1 + 1)X_1, \ldots, \text{not}(l_m + 1)X_m.$$

Given a ca-program $P$, we translate it into an mca-program

$$e_{mca}(P) = \bigcup_{r \in P} \{e_{mca}^1(r), e_{mca}^2(r)\}.$$

**Theorem 5.** Let $P$ be a ca-program. A set of atoms $M$ is a stable model of $P$, as defined for ca-programs, if and only if $M$ is a stable model of $e_{mca}(P)$, as defined for mca-programs.

This theorem shows that the formalism of mca-programs is at least as expressive as that of ca-programs. The converse is true as well: ca-programs are at least as expressive as mca-programs. Let $r$ be the following mca-clause:

$$kX \leftarrow k_1X_1, \ldots, k_mX_m, \text{not}(l_1Y_1), \ldots, \text{not}(l_nX_n).$$

We define $e_{ca}(r)$ as follows. If there is $i$, $1 \leq i \leq n$, such that $l_i = 0$, we set $e_{ca}(r) = kX \leftarrow kX$ (in fact any tautology would do). Otherwise, we set

$$e_{ca}(r) = kX \leftarrow k_1X_1, \ldots, k_mX_m, Y_1(l_1 - 1), \ldots, Y_n(l_n - 1).$$

Given an mca-program $P$, we define $e_{ca}(P) = \{e_{ca}(r) : r \in P\}$.

**Theorem 6.** Let $P$ be an mca-program. A set of atoms $M$ is a stable model of $P$, as defined for mca-programs, if and only if $M$ is a stable model of $e_{ca}(P)$, as defined for ca-programs.

Theorems 5 and 6 establish the equivalence of ca-programs and mca-programs with respect to the stable model semantics. The same translations also preserve the concept of a model. Finally, Theorem 5 suggests a way to introduce the notion of a supported model for a ca-program: a set of atoms $M$ is defined to
be a supported model of a ca-program $P$ if it is a supported model of the mca-program $\text{mca}_e(P)$. With this definition, the two translations $\text{mca}_e$ and $\text{e}_c$ also preserve the concept of a supported model.

We also note that this equivalence demonstrates that ca-programs with the semantics of stable models as defined in [12] can be viewed as a generalization of normal logic programming. It follows from Theorems 4 and 6 that the encoding of normal logic programs as ca-programs, defined as the composition of the translations $\text{mca}$ and $\text{e}_c$, preserves the semantics of models, supported models and stable models (an alternative proof of this fact, restricted to the case of stable models only was first given in [12] and served as a motivation for the class of ca-programs and its stable-model semantics). This result is important, as it is not at all evident that the NSS-reduct and Definition 7 generalize the semantics of stable models as defined in [8].

Given that the formalisms of ca-atoms and mca-atoms are equivalent, it is important to stress what differs them. The advantage of the formalism of ca-programs is that it does not require the negation operator in the language. The strength of the formalism of mca-programs lies in the fact that its syntax so closely resembles that of normal logic programs, and that the development of the theory of mca-programs so closely follows that of the normal logic programming.

8 Mca-programs and disjunctive logic programs

The formalism of mca-programs also extends an approach to disjunctive logic programming, proposed in [14]. In that paper, the authors introduced and investigated a semantics of possible models for disjunctive logic programs. We will now show that disjunctive programming with the semantics of possible models is a special case of the logic mca-programs with the semantics of stable models.

Let $r$ be a disjunctive logic program clause of the form:

$$c_1 \lor \ldots \lor c_k \leftarrow a_1, \ldots, a_m, \text{not}(b_1), \ldots, \text{not}(b_n),$$

where all $a_i$, $b_i$ and $c_i$ are atoms. We define an mca-clause

$$\text{mca}_d(r) = 1\{c_1, \ldots, c_k\} \leftarrow 1\{a_1\}, \ldots, 1\{a_m\}, 1\{\text{not}(b_1)\}, \ldots, 1\{\text{not}(b_n)\}.$$ 

For a disjunctive logic program $P$, we define $\text{mca}_d(P) = \{ \text{mca}_d(r) \mid r \in P \}$. We have the following theorem.

**Theorem 7.** Let $P$ be a disjunctive logic program. A set of atoms $M$ is a possible model of $P$ if and only if $M$ is a stable model of the mca-program $\text{mca}_d(P)$.

We also note that there are strong analogies between the approach we propose here and some of the techniques discussed in [14]. In particular, [14] presents a computational procedure for disjunctive programs without negation that is equivalent to our notion of a $P$-computation. We stress however, that the class of mca-programs is more general and that our approach, consistently exploiting properties of an operator $T_{P^d}$, is better aligned with a standard development of normal logic programming.
9 Discussion

Results of our paper point to a central position of mca-programs among other logic programming formalisms. First, mca-programs form a natural generalization of normal logic programs, with most concepts and techniques closely patterned after their counterparts in normal logic programming. Second, mca-programs with the stable-model semantics generalize disjunctive logic programming with the possible-model semantics of [14]. Third, mca-programs provide direct means to model cardinality constraints, a feature that has become broadly recognized as essential to computational knowledge representation formalisms. Moreover, it turns out that mca-programs are, in a certain sense that we made precise in the paper, equivalent to logic programs with cardinality atoms proposed and studied in [12]. Thus, mca-programs provide a natural link between normal logic programs and the formalism of [12], and help explain the nature of this relationship, hidden by the original definitions in [12].

In this paper, we outlined only the rudiments of the theory of mca-programs. There are several questions that follow from our work and that deserve more attention. First, our theory can be extended to the case of programs built of monotone-weight atoms, that is, expressions of the form $a \{ p_1 : w_1, \ldots, p_k : w_k \}$, where $a, w_1, \ldots, w_k$ are non-negative reals and $p_1, \ldots, p_k$ are propositional atoms. Intuitively, such an atom is satisfied by an interpretation (set of atoms) $M$ if the sum of weights assigned to atoms in $M \cap \{ p_1, \ldots, p_k \}$ is at least $a$.

Next, there is a question whether Fages lemma [6] generalizes to mca-programs. If so, for some classes of programs, one could reduce stable-model computation to satisfiability checking for propositional theories with cardinality atoms [4, 9]. That, in turn, might lead to effective computational methods, alternative to direct algorithms such as smodels [10] and similar in spirit to the approach of cmodels [5, 1].

Another interesting aspect concerns some syntactic modifications and “normal form representations” for mca-programs. For instance, at a cost of introducing new atoms, one can rewrite any mca-program into a simple mca-program in which every mca-clause contains at most one mca-literal in its body and in which the use of negation is restricted (but not eliminated). We will present these results in a full version of the paper.

The emergence of a nondeterministic one-step provability operator is particularly intriguing. It suggests that, as in the case of normal logic programming [7, 13], the theory of mca-programs can be developed by algebraic means. For that to happen, one would need techniques for handling nondeterministic operators on lattices, similar to those presented in the deterministic operators in [2, 3]. That approach might ultimately lead to a generalization of the well-founded semantics to the case of mca-programs.
Acknowledgments

The second author was supported by the Academy of Finland grant 53695. The other two authors were supported by the NSF grants IIS-0097278 and IIS-0325063.

References