Properties of Programs with Monotone and Convex Constraints

Lengning Liu and **Mirosław Truszczyński**
Department of Computer Science, University of Kentucky,
Lexington, KY 40506-0046, USA

Abstract

We study properties of programs with *monotone* and *convex* constraints. We extend to these formalisms concepts and results from normal logic programming. They include tight programs and Fages Lemma, program completion and loop formulas, and the notions of strong and uniform equivalence with their characterizations. Our results form an abstract account of properties of some recent extensions of logic programming with aggregates, especially the formalism of *smodels*.

Introduction

We study programs with *monotone* and *convex* constraints. These formalisms allow constraints to appear in the heads of program rules. That sets them apart from other recent proposals for integrating constraints with logic programs (Denecker *et al.* 2001; Pelov *et al.* 2004; Dell'Armi *et al.* 2003; Pelov 2004; Faber *et al.* 2004) and makes them suitable as an abstract basis for formalisms such as *smodels* programs (Simons *et al.* 2002). In this paper we develop a theory of programs with monotone and convex constraints and show that main concepts, techniques and results from normal logic programming extend to these abstract formalisms.

Normal logic programming with the semantics of stable models is an effective knowledge representation formalism, mostly due to its ability to express default assumptions (Baral 2003; Gelfond and Leone 2002). However, modeling numeric constraints on sets in normal logic programming is cumbersome, requires many auxiliary atoms and leads to large programs hard to process efficiently. Since such constraints, often called *aggregates*, are ubiquitous, researchers proposed extensions of normal logic programming with explicit means to express aggregates, and generalized the stable-model semantics to the extended settings.

Aggregates imposing bounds on weights of sets of atoms and literals, called *weight* or *pseudoboolean* constraints, are especially common in practical applications and are included in all recent extensions of logic programs with aggregates. Typically, these extensions do not allow aggregates to appear in the heads of rules. A notable exception is the formalism of programs with weight constraints (Simons *et al.* 2002), often referred to as *smodels*, which remains one of the most commonly used extensions of logic programming with weight constraints. Since rules in *smodels* programs may have weight constraints as their heads, the concept of one-step provability is nondeterministic and so, establishing direct parallels with normal logic programming is difficult.

An explicit connection emerged only recently, when (Marek *et al.* 2004; Marek and Truszczyński 2004) introduced *logic programs with monotone constraints*. These programs allow aggregates in the heads of rules and support nondeterministic computations. (Marek *et al.* 2004; Marek and Truszczyński 2004) proposed a generalization of the van Emden-Kowalski one-step provability operator to account for that nondeterminism, defined supported and stable models for programs with monotone constraints that mirror their normal logic programming counterparts, and showed encodings of *smodels* programs as programs with monotone constraints.

In this paper, we continue investigations of programs with monotone constraints. We adapt to programs with monotone constraints the notion of a *tight* program (Erdem and Lifschitz 2003) and generalize Fages Lemma (Fages 1994).

We introduce extensions of propositional logic with monotone constraints. We define the completion of a monotone-constraint program with respect to this logic, and generalize the notion of a loop formula. We then prove the loop-formula characterization of stable models of programs with monotone constraints, extending to the setting of monotone-constraint programs results obtained for normal logic programs in (Clark 1978; Lin and Zhao 2002).

We show that the notions of uniform and strong equivalence of programs (Lifschitz *et al.* 2001; Lin 2002; Turner 2003; Eiter and Fink 2003) extend to programs with monotone constraints, and that their characterizations (Turner 2003; Eiter and Fink 2003) generalize, as well.

Programs with monotone constraints make explicit references to the default negation operator. We show that by allowing a more general class of constraints, called *convex*, default negation can be eliminated from the language. We argue that all results in our paper extend to programs with convex constraints.

Our paper shows that programs with monotone and convex constraints have a rich theory that closely follows that of normal logic programming. It implies that programs with monotone and convex constraints form an abstract generalization of extensions of normal logic programming such as *smodels*. In particular, all results we obtain in this abstract setting specialize to *smodels* programs and, in most cases, yield results that are new.

Due to space limitations we omit proofs or provide only short sketches of arguments.

Preliminaries

We consider the propositional case only. It does not lead to loss of generality, as programs with variables typically are interpreted in terms of propositional ones. We also assume a fixed and countably infinite set At of propositional atoms. The definitions and results we present in this section come from (Marek and Truszczyński 2004). Some of them are more general as in the present paper we also consider programs with inconsistent constraints in the heads.

Constraints. A *constraint* is an expression $A = (X, C)$, where $X \subseteq At$ is finite, we call it the *domain* of the constraint, and $C \subseteq \mathcal{P}(X)$ ($\mathcal{P}(X)$ denotes the powerset of X). The assumption of the finiteness of constraint domains can be dropped. We adopt it here as it simplifies the presentation.

A constraint (X, C) describes a property of subsets of its domain, with C consisting precisely of these subsets of X that *satisfy* the constraint (have property) C.

In the paper, we represent truth assignments (interpretations) by the sets of atoms that are true in them. An interpretation $M \subseteq At$ *satisfies* a constraint $A = (X, C)$ ($M \models A$), if $M \cap X \in C$ or, informally, if the set of atoms from X that are true in M satisfies constraint (property) A. Otherwise, M does not satisfy A, ($M \not\models A$).

A constraint $A = (X, C)$ is *consistent* if there is M such that $M \models A$. It is so precisely when $C \neq \emptyset$.

Constraint programs. Constraints are building blocks of rules and programs. (Marek and Truszczyński 2004) defined *constraint programs* to consist of *constraint* rules

$$A \leftarrow A_1, \ldots, A_k, \mathbf{not}(A_{k+1}), \ldots, \mathbf{not}(A_m) \qquad (1)$$

where A, $A_1, \ldots, A_m$ are constraints and $\mathbf{not}$ denotes *default negation*. The constraint A is the *head* and the set $\{A_1, \ldots, A_k, \mathbf{not}(A_{k+1}), \ldots, \mathbf{not}(A_m)\}$ is the *body* of r. We denote them by $hd(r)$ and $bd(r)$, respectively.

We denote by $At(P)$ the set of atoms in the domains of constraints in a constraint program P. We denote by $hset(P)$, the *headset* of P, that is, the union of the domains of the heads of all rules in P.

Models. The concept of satisfiability extends in a standard way to negated literals and rules. An interpretation $M \subseteq At$ is a *model* of a constraint program P if it satisfies every rule in P.

***M*-applicable rules.** Let $M \subseteq At$ be an interpretation. A rule (1) is *M-applicable* if M satisfies every literal in $bd(r)$. We denote by $P(M)$ the set of all M-applicable rules in P.

Supported models. Supportedness is a property of models. Intuitively, every atom a in a supported model must have "reasons" for being "in". Such reasons are M-applicable rules whose heads contain a in their domains. Formally, let P be a constraint program and M a subset of $At(P)$. A model M of P is *supported* if $M \subseteq hset(P(M))$.

Nondeterministic one-step provability. Let P be a constraint program and M a set of atoms. A set M' is *nondeterministically one-step provable* from M by means of P, if $M' \subseteq hset(P(M))$ and $M' \models hd(r)$, for every rule r in $P(M)$.

The *nondeterministic one-step provability operator* T_P^{nd} for a program P is an operator on $\mathcal{P}(At)$ such that for every $M \subseteq At$, $T_P^{nd}(M)$ consists of all sets that are nondeterministically one-step provable from M by means of P.

The operator T_P^{nd} is *nondeterministic* as it assigns to each $M \subseteq At$ a *family* of subsets of At, each being a possible outcome of applying P to M. In general, T_P^{nd} is *partial*, since there may be sets M such that $T_P^{nd}(M) = \emptyset$ (no set can be derived from M by means of P).

Monotone constraints. A constraint (X, C) is *monotone* if C is closed under superset, that is, for every $W, Y \subseteq X$, if $W \in C$ and $W \subseteq Y$ then $Y \in C$.

Cardinality constraints in *smodels* are examples of monotone constraints. A cardinality constraint $A = k\{a_1, \ldots, a_n\}$ requires that the number of atoms in $\{a_1, \ldots, a_n\}$ that are true in an interpretation be at least k. If $M \models A$ and $M' \supseteq M$, then $M' \models A$ as well. Thus, cardinality constraints are indeed monotone. In *smodels*, cardinality constraints may also specify upper bounds. In such cases, we can split the constraint into the conjunction of a monotone constraint and the negation of a monotone constraint. *Smodels* weight constraints with all weights non-negative and without negative literals are also examples of monotone constraints.

Monotone-constraint programs. We call constraint programs built of monotone constraints *monotone-constraint programs* or *programs with monotone constraints*. From now we consider *only* monotone constraints and programs with monotone constraints.

Horn constraint programs. A rule (1) is *Horn* if constraints A, $A_1, \ldots, A_k$ are monotone and if $k = m$ (no occurrences of default negation). A constraint program is *Horn* if every rule in the program is Horn.

Bottom-up computation. Let P be a Horn constraint program. A *P-computation* is a sequence $\langle X_k \rangle$ (indexed with non-negative integers) such that $X_0 = \emptyset$ and for every k,

$$X_k \subseteq X_{k+1}, \text{ and } X_{k+1} \in T_P^{nd}(X_k).$$

The *result* of a P-computation $t = \langle X_k \rangle$ is the set $\bigcup_k X_k$. We denote it by R_t.

Proposition 1 *Let P be a Horn constraint program and t a P-computation. Then R_t is a supported model of P.*

Derivable models. We use computations to define *derivable* models of Horn constraint programs. A set M of atoms is a *derivable model* of a Horn constraint program P if there exists a P-computation t such that $M = R_t$. By Proposition 1, derivable models are indeed models.

Since inconsistent monotone constraints may appear in the heads of Horn rules, there are Horn programs P and sets $X \subseteq At$, such that $T_P^{nd}(X) = \emptyset$. Thus, some Horn constraint programs have no computations and no derivable models. However, if a Horn constraint program has models, the existence of computations and derivable models is guaranteed.

Let M be a model of a Horn constraint program P. We define $X_0^{P,M} = \emptyset$ and, for $k \geq 0$, we set

$$X_{k+1}^{P,M} = hset(P(X_k^{P,M})) \cap M.$$

The sequence $\langle X_k^{P,M} \rangle$ is well-defined. We denote it by $t^{P,M}$. We have the following result.

Proposition 2 *Let P be a Horn constraint program. For every model M of P, the sequence $t^{P,M}$ is a P-computation.*

We call the computation $t^{P,M}$ *canonical* and denote its result by $Can(P, M)$. We now gather properties of derivable models that extend properties of the least model of normal Horn logic programs.

Proposition 3 *Let P be a Horn constraint program. Then:*

1. *For every model M of P, $Can(P, M)$ is a greatest derivable model of P contained in M*
2. *A set of atoms M is a derivable model of P if and only if $M = Can(P, M)$*
3. *If M is a minimal model of P then M is a derivable model of P.*

The reduct. Let P be a monotone-constraint program and M a subset of $At(P)$. The reduct of P, denoted by P^M, is a program obtained from P by:

1. removing from P all rules whose body contains a literal $\mathbf{not}(B)$ such that $M \models B$;
2. removing literals $\mathbf{not}(B)$ for the bodies of the remaining rules.

Stable models. The reduct of a monotone-constraint program is Horn since it contains no occurrences of default negation. Therefore, the following definition is sound.

Let P be a program. A set of atoms M is a *stable* model of P if M is a derivable model of P^M.

The definitions of the reduct and stable models follow and generalize those proposed for normal logic programs, if we take into account that in the setting of Horn constraint programs, derivable models play the role of a least model.

As in normal logic programming and its standard extensions, stable models of monotone-constraint programs are supported models and, consequently, models.

Proposition 4 *Let P be a monotone-constraint program. If $M \subseteq At(P)$ is a stable model of P, then M is a supported model of P.*

If a normal logic program is Horn then its least model is its (only) stable model. Here we have an analogous situation.

Proposition 5 *Let P be a Horn monotone-constraint program. Then $M \subseteq At(P)$ is a derivable model of P if and only if M is a stable model of P.*

Finally, we note that (Marek *et al.* 2004; Marek and Truszczyński 2004) demonstrated that *smodels* programs can be encoded as programs with monotone constraints so that stable model semantics is preserved.

In the remainder of the paper we show that several fundamental results from normal logic programming extend to the class of monotone-constraint programs.

Fages Lemma

In general, supported models and stable models of a logic program (both the normal case and the monotone-constraint case) do not coincide. Fages Lemma (Fages 1994) (later extended in (Erdem and Lifschitz 2003)), establishes a sufficient condition under which a supported model of a normal logic program is stable. In this section, we show that Fages Lemma extends to programs with monotone constraints.

Definition 1 *A monotone-constraint program P is called* tight *on a set $M \subseteq At(P)$ of atoms, if there exists a mapping λ from M to ordinals such that for every rule $r = A \leftarrow A_1, \ldots, A_k, \mathbf{not}(A_{k+1}), \ldots, \mathbf{not}(A_m)$ in $P(M)$, if X is the domain of A and X_i the domain of A_i, $1 \leq i \leq k$, then for every $x \in M \cap X$ and for every $a \in M \cap \bigcup_{i=1}^{k} X_i$, $\lambda(a) < \lambda(x)$.*

We will now show that tightness provides a sufficient condition for supported model to be stable. In order to prove a general result, we first establish it in the Horn case.

Lemma 1 *Let P be a Horn monotone-constraint program and let M be a supported model of P. If P is tight on M, then M is a stable model of P.*

Given that lemma, the general result follows easily.

Theorem 1 *Let P be a monotone-constraint program and let M be a supported model of P. If P is tight on M, then M is a stable model of P.*

Proof: One can check that if M is a supported model of P, then it is a supported model of the reduct P^M. Since P is tight on M, the reduct P^M is tight on M, too. Thus, M is a stable model of P^M (by the lemma) and, consequently, a derivable model of P^M (by Proposition 5). It follows that M is a stable model of P. $\square$

Logic PL^{mc} and completion of monotone-constraint programs

The *completion* of a normal logic program (Clark 1978) is a propositional theory whose models are precisely supported models of the program. Thus, supported models of the program can be computed by means of SAT solvers. In those cases, when supported models are stable (e.g., when the assumptions of Fages Lemma hold) that allows us to use SAT solvers to compute stable models of normal programs, an idea proposed in (Babovich, Erdem, & Lifschitz 2000) and incorporated in *cmodels* (Babovich and Lifschitz 2002).

Our goal is to extend the concept of the completion to programs with monotone constraints. It will allow us, in the restricted setting of *smodels* programs, to use solvers of propositional weight (pseudoboolean) constraints to compute supported models. Unlike *cmodels* (Babovich and Lifschitz 2002), the resulting approach does not require that pseudoboolean constraints be compiled away prior to the application of SAT solvers, as it can use solvers of weight constraints developed by the SAT community (e.g., SATZOO (Eén and Sörensson 2003)).

To define the completion, we first introduce an extension of propositional logic with monotone constraints, a formalism that we denote by PL^{mc}. A *formula* in the logic PL^{mc}

is a boolean combinations of monotone constraints. The notion of a model of a constraint, as introduced earlier, extends in a standard way to the class of formulas in the logic PL^{mc}.

Let P be a monotone-constraint program. The *completion* of P, denoted by $Comp(P)$, is the collection of the following PL^{mc} formulas:

1. $A_1 \wedge \ldots \wedge A_k \wedge \neg A_{k+1} \wedge \ldots \wedge \neg A_m \rightarrow A$ for each rule $r \in P$;
2. $x \rightarrow bd(r_1) \vee \ldots \vee bd(r_l)$, for each atom $x \in At(P)$, where rules $r_1, \ldots, r_l$ are the rules in P containing x in the domains of their heads (we write the body of a rule as a conjunction of monotone constraints so that it follows the syntax of PL^{mc}).

The following theorem shows that the completion we defined generalizes the completion of normal logic programs.

Theorem 2 *Let P be a monotone-constraint program and S a subset of $At(P)$. S is a supported model of P if and only if S is a model of $Comp(P)$.*

Loops and loop formulas in monotone-constraint programs

The completion alone is not quite satisfactory as it relates *supported* not *stable* models of monotone-constraint programs with models of PL^{mc} theories. Loop formulas, proposed in (Lin and Zhao 2002), provide a way to eliminate non-stable models from among supported models of a normal logic program (models of its completion). Thus, they allow us to use SAT solvers to compute stable models of *arbitrary* normal logic programs and not only those for which supported and stable models coincide.

We will now extend this idea to monotone-constraint programs. Let P be a monotone-constraint program. The *positive dependency graph* of P is the directed graph $G_P = (V, E)$, where $V = At(P)$ and $\langle u, v\rangle$ is an edge in E if and only if there exists a rule $r \in P$ such that u occurs in the domain of the head constraint of r and v occurs in the domain of some non-negated monotone constraint in the body of r.

Let $G = (V, E)$ be a directed graph. A set $L \subseteq V$ is a *loop* in G if the subgraph of G induced by L is strongly connected. A maximal loop L in G (maximal loops are vertex sets of strongly connected components of G) is *terminating* if there is no path in G from L to any other maximal loop.

A rule r of the form (1) that belongs to a monotone-constraint program P *participates* in a loop L of G_P, written as $r \lhd L$, if $L \cap X \neq \emptyset$, where X is the domain of the head constraint A of r. Let us assume each A_i in rule r has the form (X_i, C_i). By $\beta_L(r)$, we mean the following PL^{mc} formula obtained from the body of r (of the form (1)):

$$A'_1 \wedge \ldots \wedge A'_k \wedge \neg A_{k+1} \wedge \ldots \wedge \neg A_m,$$

where $A'_i = (X_i \setminus L, C_i)$ for $i = 1, \ldots, k$.

If L is a loop of a monotone-constraint program P, then the *loop formula* of L, denoted by $LP(L)$, is the PL^{mc} formula

$$LP(L) = \bigvee L \rightarrow \bigvee \{\beta_L(r) \colon r \in P \;\&\; r \lhd L\}.$$

We have the following theorem:

Theorem 3 *Let P be a monotone-constraint program and M a subset of $At(P)$. Then M is a stable model of P if and only if M is a model of $Comp(P) \cup \{LP(L) \colon\ L$ is a loop in $G_P\}$.*

Let M be a supported model of P that is not stable. By Proposition 3, M contains a greatest derivable model M' of P^M. Let $M^- = M \setminus M'$. By $G_P[M^-]$ we denote the subgraph of G_P induced by atoms in M^-. The following result provides us with a way to filter out specific non-stable supported models from $Comp(P)$.

Theorem 4 *Let P be a monotone-constraint program and M a model of $Comp(P)$. If M^- is not empty, then M violates the loop formula of every terminating loop of $G_P[M^-]$.*

Strong and uniform equivalence of monotoneconstraint programs

Program equivalence (Lifschitz *et al.* 2001; Lin 2002; Turner 2003; Eiter and Fink 2003) is an important concept due to its potential uses in program rewriting and optimization. (Turner 2003) presented an elegant characterization of strong equivalence of *smodels* programs and (Eiter and Fink 2003) described a similar characterization of uniform equivalence of normal and disjunctive logic programs. We show that both characterizations can be adapted to the case of monotone-constraint programs.

***M*-maximal models.** A key role in our approach is played by models of Horn constraint programs satisfying a certain maximality condition. Let P be a Horn constraint program. Let M and N be models of P such that $N \subseteq M \subseteq At(P)$. Then N is an M-*maximal* model of P, written $N \models_M P$, if for every $r \in P(N)$, $M \cap X \subseteq N$, where X is the domain of $hd(r)$.

Intuitively, N is an M-maximal model of P if N satisfies each rule $r \in P(N)$ "maximally" with respect to M. That is, if N contains all atoms in M that belong to the domain of the constraint in the head of r.

Here is an example of an M-maximal model. Let $P = \{1\{p, q, r\} \leftarrow 1\{s, t\}\}$. It is evident that P is Horn. Let $M = \{p, q, s, t\}$ and $N = \{p, q, s\}$. We can verify that both M and N are models of P. Moreover, since the only rule in P is N-applicable, and $M \cap \{p, q, r\} \subseteq N$, N is an M-maximal model of P. With similar reasoning, we can see that $N' = \{p, s\}$ is not M-maximal even though N' is a model of P and it is contained in M.

M-maximal models have several interesting properties. Due to space limitations, we list here only one that plays an important role in our further considerations.

Proposition 6 *Let P be a Horn constraint program and let M be a model of P. Then $Can(P, M)$ is the least M-maximal model of P.*

Strong equivalence and SE-models. Monotone-constraint programs P and Q are *strongly equivalent*, denoted by $P \equiv_s Q$, if for every monotone-constraint program R, $P \cup R$ and $Q \cup R$ have the same set of stable models.

To study the strong equivalence of monotone-constraint programs, we generalize the concept of an *SE-model* from (Turner 2003).

Let P be a monotone-constraint program and let $X, Y \subseteq At(P)$. We say that (X, Y) is an *SE-model* of P if the following conditions hold: (1) $X \subseteq Y$; (2) $Y \models P$; and (3) $X \models_Y P^Y$. We denote by $SE(P)$ the set of all SE-models of P.

If M is a model of a monotone-constraint program P, it follows directly from the definition that both (M, M) and $(Can(P^M, M), M)$ are SE-models of P.

SE-models yield a simple characterization of strong equivalence of monotone-constraint programs.

Theorem 5 *Let P and Q be monotone-constraint programs. Then $P \equiv_s Q$ if and only if $SE(P) = SE(Q)$.*

Proof (sketch). ($\Leftarrow$) the following two properties are easy to check: (1) If $SE(P) = SE(Q)$, then P and Q have the same stable models; and (2) $SE(P \cup R) = SE(P) \cap SE(R)$ holds for any monotone-constraint programs P and R.

Let R be an arbitrary monotone-constraint program. Property (2) implies that $SE(P \cup R) = SE(P) \cap SE(R)$ and $SE(Q \cup R) = SE(Q) \cap SE(R)$. Since $SE(P) = SE(Q)$, we have that $SE(P \cup R) = SE(Q \cup R)$. By property (1), $P \cup R$ and $Q \cup R$ have the same stable models. Hence, $P \equiv_s Q$ holds.

($\Rightarrow$) One can show that if $P \equiv_s Q$ then the following two properties hold: (1) P and Q have the same set of models; and (2) for any X, Y such that $X \subseteq Y$ and $Y \models P \cup Q$, $X \models_Y P^Y$ if and only if $X \models_Y Q^Y$.

Let us assume that $P \equiv_s Q$ and let us consider an SE-model (N, M) of P. By the definition, $N \subseteq M$, $M \models P$, and $N \models_M P^M$. By property (1), we have $M \models Q$. By property (2), we have $N \models_M Q^M$. Hence, (N, M) is an SE-model of Q, as well. It follows that $SE(P) \subseteq SE(Q)$. The other inclusion follows by symmetry. $\square$

Uniform equivalence and UE-models. Monotone-constraint programs P and Q are *uniformly equivalent*, denoted by $P \equiv_u Q$, if for every set of *atoms* X, $P \cup X$ and $Q \cup X$ have the same stable models.

An SE-model (X, Y) of a monotone-constraint program P is a *UE-model* of P if for every SE-model (X', Y) of P with $X \subseteq X'$, either $X = X'$ or $X' = Y$ holds.

We write $UE(P)$ to denote the set of all UE-models of P.

Our notion of a UE-model is a generalization of the notion of a UE-model from (Eiter and Fink 2003) to the setting of monotone-constraint programs.

We have the following characterization of uniform equivalence in terms of UE-models.

Theorem 6 *Let P and Q be two monotone-constraint programs. Then $P \equiv_u Q$ if and only if $UE(P) = UE(Q)$.*

Proof (sketch). We outline only the proof of the implication ($\Leftarrow$). Let X be an arbitrary set of atoms and M be a stable model of $P \cup X$. Then M is a model of P and, consequently, $(M, M) \in UE(P)$. It follows that $(M, M) \in UE(Q)$ and so, M is a model of Q. Since M is a model of X, too, M is a model of $Q \cup X$ and thus, also of $(Q \cup X)^M$.

Let $N = Can((Q \cup X)^M, M)$. Then $X \subseteq N \subseteq M$ and, by Proposition 6, N is an M-maximal model of $(Q \cup X)^M$. Consequently, N is an M-maximal model of Q^M. One can now show that $(N, M) \in UE(Q)$. By the assumption, $(N, M) \in UE(P)$. It follows that $N \models_M P^M$ and since $X \subseteq N$, $N \models_M (P \cup X)^M$. Thus, N is an M-maximal model of $(P \cup X)^M$. Since M is a stable model of $P \cup X$, $M = Can((P \cup X)^M, M)$. By Proposition 6, M is the least M-maximal model of $(P \cup X)^M$. Thus, $M \subseteq N$.

It follows that $M = N = Can((Q \cup X)^M, M)$. Thus, M is a stable model of $Q \cup X$. By symmetry, every stable model of $Q \cup X$ is also a stable model of $P \cup X$. $\square$

Examples. Let $P = \{1\{p, q\} \leftarrow \mathbf{not}(2\{p, q\})\}$, and $Q = \{p \leftarrow \mathbf{not}(q).\ q \leftarrow \mathbf{not}(p)\}$. Then P and Q are strongly equivalent. We note that both programs have $\{p\}$, $\{q\}$, and $\{p, q\}$ as models. Furthermore, $(\{p\}, \{p\})$, $(\{q\}, \{q\})$, $(\{p\}, \{p, q\})$, $(\{q\}, \{p, q\})$, $(\{p, q\}, \{p, q\})$ and $(\emptyset, \{p, q\})$ are all SE-models of the two programs. Thus, by Theorem 5, P and Q are strongly equivalent.

We also observe that the first five SE-models are precisely UE-models of P and Q. Therefore, by Theorem 6, P and Q are also uniformly equivalent.

It is possible for two monotone-constraint programs to be uniformly but not strongly equivalent. If we add rule $p \leftarrow$ to P, and rule $p \leftarrow q$ to Q, then the two resulting programs, say P' and Q', are uniformly equivalent. However, they are not strongly equivalent. The programs $P' \cup \{q \leftarrow p\}$ and $Q' \cup \{q \leftarrow p\}$ have different stable models. Another way to show it is by observing that $(\emptyset, \{p, q\})$ is an SE-model of Q' but not an SE-model of P'.

Programs with convex constraints

We will now discuss a syntactic variant of programs with monotone constraints.

A constraint (X, C) is *antimonotone* if it is closed under subset, that is, for every $W, Y \subseteq X$, if $Y \in C$ and $W \subseteq Y$ then $W \in C$. A constraint (X, C) is *convex*, if for every $W, Y, Z \subseteq X$ such that $W \subseteq Y \subseteq Z$ and $W, Z \in C$, we have $Y \in C$.

The *upward* and *downward closures* of a constraint $A = (X, C)$ are constraints $A^+ = (X, C^+)$ and $A^- = (X, C^-)$, respectively, where

$C^+ = \{Y \subseteq X \colon \text{for some } W \in C, W \subseteq Y\}$, and
$C^- = \{Y \subseteq X \colon \text{for some } W \in C, Y \subseteq W\}$.

Proposition 7 *A constraint (X, C) is convex if and only if $C = C^+ \cap C^-$.*

Programs with monotone constraints make use of the default negation operator. Allowing convex constraints in rules leads to an *equivalent* formalism, without the need for default negation in the language. A *convex-constraint rule* is an expression r of the form

$$A \leftarrow A_1, \ldots, A_k \qquad (2)$$

where A, $A_1, \ldots, A_k$ are convex constraints. A *convex-constraint program* is a set of convex-constraint rules. Programs with convex constraints are of interest, as their syntax is more directly aligned with that of *smodels* programs.

Programs with monotone and convex constraints are closely related. If A is a monotone constraint, a literal $\mathbf{not}(A)$ behaves as an antimonotone constraint: if $M \models$

$\mathbf{not}(A)$ and $M' \subseteq M$, then $M' \models \mathbf{not}(A)$. Thus, programs with monotone constraints can be regarded directly as programs with convex constraints.

The converse embedding also exists. It relies on Proposition 7, which allows us to split a convex constraint into the conjunction of two constraints, one monotone and the other one — antimonotone. To construct it, we note that if $A = (X, C)$ is antimonotone then $\overline{A} = (X, \overline{C})$, where $\overline{C} = \mathcal{P}(X) \setminus C$, is monotone. Let $r = A \leftarrow A_1, \ldots, A_k$ be a rule built of convex constraints. We encode r with the following two rules r' and r'' built of monotone constraints:

$$A^+ \leftarrow A_1^+, \mathbf{not}\ \overline{A_1^-}, \ldots, A_k^+, \mathbf{not}\ \overline{A_n^-}$$
$$I \leftarrow \overline{A^-}, A_1^+, \mathbf{not}\ \overline{A_1^-}, \ldots, A_n^+, \mathbf{not}\ \overline{A_k^-}$$

where I is a fixed inconsistent constraint, say $(\emptyset, \emptyset)$.

We can formally show that under the embeddings we outlined above, the formalisms of programs with monotone and convex constraints are equivalent with respect to all semantics we consider in the paper and all results we stated here in terms of monotone-constraint programs can be restated in terms of convex-constraint programs. In particular, it follows from our results that default negation can be expressed in terms of antimonotone constraints.

Conclusions

Our work shows that concepts, techniques and results from normal logic programming generalize to the abstract setting of programs with monotone and convex constraints allowing constraints in the heads of program rules.

Several results we obtained in the paper for these general classes of programs specialize to *new* results about *smodels* programs. While characterizations of strong equivalence of *smodels* programs were obtained in (Turner 2003), the characterization of uniform equivalence of *smodels* programs implied by Theorem 6 is new.

Specializations to the case of *smodels* of Theorems 1, 2, 3 and 4, concerning Fages Lemma, the completion of a program and loop formulas, are also new. Given these results we are currently working on an implementation of a new software for computing stable models of *smodels* programs. The idea is to follow the approach developed in *assat* (Lin and Zhao 2002), and use the results we mentioned above to reduce the problem to that of finding models of theories consisting of pseudoboolean constraints, for which several fast solvers exist (Eén and Sörensson 2003).

Acknowledgments

We acknowledge the support of NSF grants IIS-0097278 and IIS-0325063.

References

Y. Babovich and V. Lifschitz. *Cmodels package*, 2002. `http://www.cs.utexas.edu/users/tag/cmodels.html`.

Babovich, Y.; Erdem, E.; and Lifschitz, V. 2000. Fages' theorem and answer set programming. In: *Proceedings of NMR-2000*. `http://xxx.lanl.gov/html/cs.AI/0003073`.

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

K. Clark 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and data bases*. New York-London: Plenum Press.

T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in DLV. In *Proc. of IJCAI-2003*, pages 847–852. Morgan Kaufmann, 2003.

M. Denecker, N. Pelov, and M. Bruynooghe. Ultimate well-founded and stable semantics for logic programs with aggregates. In *Proc. of ICLP-2001*, vol. 2237 of *LNCS*, pages 212–226. Springer, 2001.

N. Eén and N. Sörensson". An extensible SAT solver. In *Proc. of SAT-2003*, vol. 2919 of *LNCS*, pages 502–518, Springer, 2003.

T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In *Proc. of ICLP-2003*, vol. 2916 of *LNCS*, pages 224–238. Springer, 2003.

E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4-5):499–518, 2003.

W. Faber, N. Leone and G. Pfeifer. Recursive aggregates in disjunctive logic programs: semantics and complexity. Proceedings of JELIA-2004, vol. 3229 of *LNAI*, pages 200–212, Springer, 2004.

F. Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

M. Gelfond and N. Leone. Logic programming and knowledge representation – the A-prolog perspective. *Artificial Intelligence*, 138:3–38, 2002.

M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In *Proc. of ICLP-1988*, pages 1070–1080. MIT Press, 1988.

V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Trans. on Computational Logic*, 2(4):526–541, 2001.

F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of AAAI-2002*, pages 112–117. AAAI Press, 2002.

F. Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proc. of KR-2002*, Morgan Kaufmann, 2002.

V.W. Marek and M. Truszczyński. Logic programs with abstract constraint atoms. In *Proc. of AAAI-04*, pages 86–91. AAAI Press, 2004.

V.W. Marek, I. Niemelä, and M. Truszczyński. Characterizing stable models of logic programs with cardinality constraints. In *Proc. of LPNMR-2004*, vol. 2923 of *LNAI*, pages 154–166. Springer, 2004.

N. Pelov. Semantics of logic programs with aggregates. *Ph.D. Thesis*. Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2004.

N. Pelov, M. Denecker, and M. Bruynooghe. Partial stable models for logic programs with aggregates. In *Proc. of LPNMR-2004*, vol. 2923 of *LNAI*, pages 207–219. Springer, 2004.

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

H. Turner. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3, (4&5):609–622, 2003.