Logic Programs with Abstract Constraints: Representation, Disjunction and Complexities[1]

Jia-Huai You, Li Yan Yuan, Guohua Liu
Department of Computing Science
University of Alberta, Edmonton, Alberta, Canada

Yi-Dong Shen
Lab of Computer Science, Institute of Software
Chinese Academy of Sciences, Beijing, China

Abstract. We study logic programs with arbitrary abstract constraint atoms, called *c-atoms*. As a theoretical means to analyze program properties, we investigate the possibility of unfolding these programs to logic programs composed of ordinary atoms. This approach reveals some structural properties of a program with c-atoms, and enables characterization of these properties based on the known properties of the transformed program. Furthermore, this approach leads to a straightforward definition of answer sets for disjunctive programs with c-atoms, where a c-atom may appear in the head of a rule as well as in the body. We also study the complexities for various classes of logic programs with c-atoms.

1 Introduction

Logic programs with abstract constraints were introduced as a general framework for representing, and reasoning with, sets of atoms [12, 13]. This is in contrast with traditional logic programs, which are used primarily to reason with individuals.

An abstract constraint atom, which is also called a *c-atom* following [21], is of the form (D, C), where D is a domain and C is a collection of subsets from the power set of D, which are intended to represent admissible solutions. By allowing c-atoms to appear anywhere in a rule, the framework of logic programs with c-atoms has become a highly expressive knowledge representation language. For example, many problems can be conveniently represented with constraints over sets of atoms, such as *weight* and *cardinality constraints* and *aggregates* (see, e.g. [2–5, 7, 15–19]).

In the study of logic programs with abstract constraints, a crucial assumption was made: an abstract constraint be monotone [12, 13]. A constraint is *monotone* if it holds that whenever it is satisfied by a set of atoms I, it must be satisfied by any extension of I. It is observed that under this assumption, much of the basic concepts and techniques for characterizing the semantics of normal logic programs can be generalized to the new context. In addition, some important constraints, such as pseudo-boolean constraints and cardinality constraints, can be represented by abstract monotone atoms [11, 12].

The assumption on monotone atoms, however, also limits the scope of applications that the framework supports. For example, many constraints involving aggregate functions are not monotone. More generally, when a c-atom is allowed to appear in the head

[1] Revised from the version that appeared in the Proc. of LPNMR '07.

of a rule, it is fully capable of expressing a constraint in the sense of *Constraint Satisfaction Problem* (CSP). Hence, a CSP can be represented by a collection of condition-free rules. In this way, the framework of logic programs with c-atoms can express complex constraint satisfaction problems, such as those involving conditional constraints [14][2], which are useful in modeling configuration and design problems.

Recently, Son et al. propose to define answer sets for programs with arbitrary c-atoms with a notion of conditional satisfaction [21]. They show that answer sets defined this way generalize the notion of *well-supported* models for normal logic programs [6], and the resulting semantics coincides with the previously introduced semantics for logic programs with monotone c-atoms. In addition, they point out the different behaviors between answer sets defined *by reduct* and those defined *by complement*.

In this paper, we extend the work of [21] in three directions. First, we study the possibility of representing logic programs with c-atoms by logic programs composed of ordinary atoms. This can be seen as an extension of the unfolding approach presented in [20], but we also consider unfolding c-atoms in the head of a rule as well as negative c-atoms in the body. The unfolding approach leads to a characterization of the existence of answer sets based on an extended *call-consistent* condition [6]. Second, the unfolding approach leads to an answer set semantics for disjunctive programs with c-atoms. Disjunctive programs with aggregates have previously been studied in [5, 17], where aggregates do not appear in the heads of program rules. In our case, an arbitrary c-atom can appear in a disjunctive head as well as positively or negatively in a rule body. Since the unfolding approach reveals structural properties, it is possible to formulate the notion of *head cycle* for the new context. We know that if a disjunctive program is head cycle-free, it can be reduced to a normal program by *shifting* [1]. A similar result holds for disjunctive programs with c-atoms. Finally, we provide complexity results for classes of programs with c-atoms.

The next section provides background and defines the notations, followed by an example to show a usage of arbitrary c-atoms to represent CSPs in Section 3. Section 4 shows unfolding of (non-disjunctive) logic programs with c-atoms to normal programs while preserving the underlying answer set semantics. By further unfolding disjunctions in the rule heads, in Section 5 we define a semantics for disjunctive programs with c-atoms. Section 6 gives some complexity results. Section 7 is about related work, with Section 8 containing final remarks.

2 Background and Notation

We follow the notation used in [10, 21]. The key definitions are taken from [21].

We assume a propositional language $\mathcal{L}$ with a countable set $\mathcal{A}$ of *propositional atoms*. A propositional atom is also called an *ordinary atom* or just an $atom$. An *abstract constraint atom*, or *c-atom* for abbreviation, is a pair (D, C) such that $D \subseteq \mathcal{A}$ and $C \subseteq 2^D$. Given a c-atom $A = (D, C)$, we use A_d and A_c to denote D and C, respectively.

A c-atom is called *elementary* if it is of the form $(\{a\}, \{\{a\}\})$, where a is an atom. In expressing such a c-atom, we may simply write the atom instead.

[2] which were called *dynamic CSPs*.

A disjunctive program (with c-atoms) is a collection of rules of the form

$$A_1 \vee ... \vee A_k \leftarrow B_1, ..., B_m, \texttt{not } C_1, ..., \texttt{not } C_n. \quad (1)$$

where $k \geq 1$, $m, n \geq 0$, and A_i's, B_i's, and C_i's are all c-atoms. $\texttt{not } C_i$ is called a *negation-as-failure c-atom*, or simply a *negative c-atom*. A *literal* refers to a c-atom or a negative c-atom. For a rule r of the form (1), the left hand side of $\leftarrow$ is called the *head* and the right hand side the *body*. We use $head(r)$, $pos(r)$ and $neg(r)$ to denote $\{A_1, ..., A_k\}$, $\{B_1, ..., B_n\}$, and $\{\texttt{not } C_1, ..., \texttt{not } C_n\}$, respectively, and let $body(r) = pos(r) \cup neg(r)$. By abuse of notation, we may denote a rule r by $head(r) \leftarrow pos(r), neg(r)$. Note that by doing so we have assumed that when we write a set of literals in the body we mean a conjunction, and in the head we mean a disjunction.

C-atoms of the form $(D, \emptyset)$ are always *false* (to be defined shortly). When a c-atom of this type appears in the head of a rule, we will write $\bot$ instead, and call it a *constraint*.

A program P is called a *general program* (or a *non-disjunctive program*), if for every rule $r \in P$, $head(r)$ is singleton. A general program P is called *basic* if for any $r \in P$, either r is a constraint or the head of r is elementary. A *condition-free program* is a general program where all the rules have an empty body. A *positive program* is a general program without negative c-atoms. A *positive disjunctive program* is a disjunctive program without negative c-atoms.

Given a program P, $At(P)$ denotes the set of (ordinary) atoms appearing in P.

Sometimes we say *a model M restricted to the atoms appearing in a program P*. By this we mean $M \cap At(P)$, and denote it by $M|_{At(P)}$.

Let $I \subseteq \mathcal{A}$ and A be a c-atom. We say I satisfies A, denoted $I \models A$, if $I \cap A_d \in A_c$; we say I satisfies $\texttt{not } A$, denoted $I \models \texttt{not } A$, if $I \cap A_d \notin A_c$. I satisfies the body of a rule r if $I \models l$ for each $l \in body(r)$; I satisfies the disjunctive head of a rule r if $I \models A$ for some atom $A \in head(r)$. A set of atoms S is a *model* of a program P if for each rule $r \in P$, $S \models head(r)$ whenever $S \models body(r)$.

Answer sets for basic positive programs are defined with a notion of conditional satisfaction.

Definition 1. *Let R and S be two sets of atoms. The set R* conditionally satisfies *a c-atom A w.r.t. S, denoted by $R \models_S A$, if $R \models A$ and, for every I such that $R \cap A_d \subseteq I$ and $I \subseteq S \cap A_d$, we have that $I \in A_c$.*

Conditional satisfaction is used to define a generalized version of one-step provability operator. Given a basic positive program P, and sets of atoms R and S, define

$$T_P(R, S) = \{a \mid \exists r \in P,\ R \models_S body(r)\ \&\ head(r) = (\{a\}, \{\{a\}\})\}$$

Definition 2. *Let M be a model of a basic positive program P. M is an* answer set *for P iff $M = T_P^\infty(\emptyset, M)$, where $T_P^0(\emptyset, M) = \emptyset$ and $T_P^{i+1}(\emptyset, M) = T_P(T_P^i(\emptyset, M), M)$, for all $i \geq 0$.*

Answer sets for general programs are defined in two steps. In the first step, answer sets for basic programs are defined, and in the second, a general program is represented by a collection of basic programs, and the answer sets for the former are defined in terms of the answer sets for the latter.

There have been two different interpretations of negation-as-failure. The first treats a negative c-atom $\texttt{not}\ A$ by its complement, and replaces it with a positive c-atom A' where $A'_d = A_d$ and $A'_c = 2^{A_d} \backslash A_c$. Given a program P, the resulting program is called the *complement of* P. The second adopts the well-known technique of reduct, and defines the reduct of a program P w.r.t. a set of atoms M as

$$P^M = \{head(r) \leftarrow pos(r) \mid r \in P,\ \forall \texttt{not}\ C \in neg(r), M \not\models C\}$$

Definition 3. *Let P be a basic program and M a set of atoms. (i) M is an* answer set by complement *for P iff M is an answer set for its complement. (ii) M is an* answer set by reduct *for P iff M is an answer set for P^M.*

Next, a general program is represented by its instances in the form of a basic program, and the answer sets of the former are defined in terms of the answer sets of the latter.

Let P be a general program and $r \in P$. For each $\pi \in head(r)_c$ (when $|head(r)| = 1$, we write $head(r)$ to denote the only c-atom in it), the *instance* of r w.r.t. π is the set of rules consisting of

1. $b \leftarrow body(r)$, for each $b \in \pi$, and
2. $\bot \leftarrow d, body(r)$, for each $d \in head(r)_d \backslash \pi$.

An *instance of* P is a program obtained by replacing each rule of P with one of its instances. Note that an instance of P is a basic program.

Definition 4. *Let P be a general program and M a set of atoms. M is an answer set by reduct (by complement, resp.) for P iff M is an answer set by reduct (by complement, resp.) for one of its instances.*

For convenience, from now on, the term *answer set* may refer to either answer set by reduct or answer set by complement. Distinction will be made when necessary.

3 Representing Constraint Satisfaction Problem

C-atoms can be used to represent CSPs in a straightforward way. A CSP, denoted $A(\mathcal{X}, \mathcal{D}, \mathcal{C})$, consists of a finite set of variables $\mathcal{X}$, a collection of finite domains $\mathcal{D}$, where variable x's domain is denoted $D(x)$, and a collection of constraints $\mathcal{C}$. A constraint $R_{x_1 \ldots x_j} \in \mathcal{C}$ is a subset of the Cartesian product $D(x_1) \times ... \times D(x_j)$; the set of variables $\{x_1, ..., x_j\}$ is called the *scope* of the constraint. A *solution* to a CSP is an assignment where each variable is assigned a value from its domain such that all of the constraints are satisfied. A constraint $R_{x_1 \ldots x_j}$ is satisfied if and only if the assignment to variables in its scope yields a tuple in the relation $R_{x_1 \ldots x_j}$.

In the following, we will use an atom $x(v)$ to represent that the CSP variable x takes the value $v \in D(x)$. Given a CSP $A(\mathcal{X}, \mathcal{D}, \mathcal{C})$, we define the corresponding program P_A as the one that consists of the following condition-free rules:

- for each constraint $R_{x_1 \ldots x_j} \in \mathcal{C}$, there is a condition-free rule $(D, C) \leftarrow$ in P_A, where
$$D = \{x(v) \mid x \in \{x_1, ..., x_j\},\ v \in D(x)\}$$
$$C = \{\{x_1(v_1), ..., x_j(v_j)\} \mid (v_1, ..., v_j) \in R_{x_1 \ldots x_j}\}$$

– if a variable $x \in \mathcal{X}$ doesn't appear in any constraint, we have a condition-free rule $(D, C) \leftarrow$ in P_A, where
$$D = \{x(v) \mid v \in D(x)\} \quad \text{and} \quad C = \{\{x(v)\} \mid v \in D(x)\}.$$

Let $A(\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP. We can show that an assignment of the variables in $\mathcal{X}$, denoted $\{x_1 \leftarrow v_1, ..., x_n \leftarrow v_n\}$, is a solution to the CSP if and only if $\{x_1(v_1), ..., x_n(v_n)\}$ is an answer set for P_A.

Thus, the class of general programs can be used to represent conditional CSPs, where the satisfaction of a constraint may be conditioned upon the satisfaction of some other constraints. The expressiveness can be further enhanced by allowing a disjunction of constraints in the head of a rule.

4 Unfolding General Programs

We will describe the process of unfolding in three steps: unfolding the heads of rules, unfolding positive c-atoms in rule bodies, and unfolding negative c-atoms. It turns out that these transformations are independent of each other. As such, the order of their applications is unimportant.

4.1 Unfolding the Head

Let P be a general program. Unfolding the rule heads in P will yield a basic program, which is denoted $U_{hd}(P)$.

Let A be a c-atom in the head of a rule. In our unfolding, each subset $\pi \in A_c$ will be named explicitly by a new atom, say ξ_π. For each rule $r \in P$, of the form $A \leftarrow body(r)$, where $A_c = \{\pi_1, ..., \pi_k\}$, the resulting rules in $U_{hd}(P)$ are:

1. $1\{\xi_{\pi_1}, ..., \xi_{\pi_k}\}1 \leftarrow body(r)$.
2. $h \leftarrow \xi_\pi$. for each $\pi \in A_c$, and for each $h \in \pi$, and
3. $\bot \leftarrow \xi_\pi, d, body(r)$. for each $\pi \in A_c$, and for each $d \in A_d \setminus \pi$.

Note that the choice rule used above is a convenient representation of a collection of normal rules; that is,

$$\xi_{\pi_i} \leftarrow body(r), \texttt{not } \xi_{\pi_1}, ..., \texttt{not } \xi_{\pi_{i-1}}, \texttt{not } \xi_{\pi_{i+1}}, ..., \texttt{not } \xi_{\pi_k}. \quad \text{for each } 1 \leq i \leq k$$

Example 1. Suppose we have a rule: $(\{a, b, c\}, \{\emptyset, \{a\}, \{a, b\}\}) \leftarrow$. The unfolded program consists of the rules:

$$1\{\xi_\emptyset, \xi_{\{a\}}, \xi_{\{a,b\}}\}1 \leftarrow .$$

$\bot \leftarrow \xi_\emptyset, a.$	$\bot \leftarrow \xi_\emptyset, b.$	$\bot \leftarrow \xi_\emptyset, c.$
$a \leftarrow \xi_{\{a\}}.$	$\bot \leftarrow \xi_{\{a\}}, b.$	$\bot \leftarrow \xi_{\{a\}}, c.$
$a \leftarrow \xi_{\{a,b\}}.$	$b \leftarrow \xi_{\{a,b\}}.$	$\bot \leftarrow \xi_{\{a,b\}}, c.$

Unfolding the head resembles the definition of answer sets for a general program in terms of its instances in the form of basic program. However, the use of new symbols is essential in representing all the instances by a program in a compact way.

In the following, and in the rest of this paper, given a program P, we will denote by $AS(P)$ the set of all answer sets of P, and by $AS_{|N}(P)$ the set of all answer sets of P restricted to the atoms in N.

Proposition 1. *For any general program P, the transformation from P to $U_{hd}(P)$ takes polynomial time (in the size of P[3]), and satisfies $AS(P) = AS_{|At(P)}(U_{hd}(P))$.*

A condition-free program may not have an answer set. An advantage of unfolding is to understand this behavior in terms of its unfolded program.

Example 2. The following condition-free program has no answer set.

$$(\{a\}, \{\emptyset\}) \leftarrow . \qquad (\{a\}, \{\{a\}\}) \leftarrow .$$

The unfolded program is

$$\xi_\emptyset \leftarrow . \quad \bot \leftarrow \xi_\emptyset, a. \quad \xi_{\{a\}} \leftarrow . \quad a \leftarrow \xi_{\{a\}}.$$

where the conflict in the original program can be demonstrated by a derivation of $\bot$.

4.2 Unfolding Positive C-atoms in Rule Body

Unfolding positive c-atoms in rule bodies can be applied to any program. Given a program P, the unfolded program will be denoted $U_{pos}(P)$.

The unfolding is carried out in three steps. In the first step, an approximation denoted P' will be generated, from which a *weakening* process is performed, and finally subsumed rules are removed to obtain $U_{pos}(P)$.

Step One: For each rule $r \in P$ of the form

$$head(r) \leftarrow (D_1, C_1), ..., (D_n, C_n), neg(r). \tag{2}$$

where $n \geq 0$, we will have a rule

$$head(r) \leftarrow \phi_1, ..., \phi_n, neg(r).$$

in P', where ϕ_i's are new symbols, plus the following rules: for each $1 \leq j \leq n$,

$$\phi_j \leftarrow \pi, \texttt{not}\ d_1, ..., \texttt{not}\ d_k. \quad \text{for each } \pi \in C_j \text{ where } \{d_1, ..., d_k\} = D_j \setminus \pi.$$

For the next two steps, we need the following notations. Let

$$r_1: \quad \phi \leftarrow body(r_1). \qquad r_2: \quad \phi \leftarrow body(r_2).$$

be two rules in P', where ϕ is a new symbol introduced in the last step. r_1 is said to be a *cover rule* of r_2 if there exists some atom p such that $body(r_1) = \Pi \cup \{p\}$ and

[3] The size of P is measured by the number of occurrences of ordinary atoms in P. Note that the size of P could be exponential in the number of distinct atoms in P.

$body(r_2) = \Pi \cup \{\texttt{not}\ p\}$, for some Π. In the second notation, we say that r_1 subsumes r_2 if $body(r_1) \subseteq body(r_2)$.

Step Two: Let $r:\ \phi \leftarrow \Psi$ be a rule in P' and Δ_r be the set of its cover rules. Define

$$\Delta'_r = \{\phi \leftarrow (\Psi - \{\texttt{not}\ p\}) \mid \phi \leftarrow \Phi \cup \{p\} \in \Delta_r\}$$

In this step, P' is weakened by repeatedly performing the following operation until a fixpoint is reached for all the rules in P': for any rule $r \in P'$, if Δ'_r is non-empty, then replace r by the rules in Δ'_r, i.e., after each operation, the new P' is $(P' - \{r\}) \cup \Delta'_r$.

Step Three: Remove all subsumed rules in P'. The resulting program is $U_{pos}(P)$.

Example 3. Consider the following program P from [21]:

$$p(1). \qquad p(-1) \leftarrow p(2).$$
$$p(2) \leftarrow \text{SUM}(\{X \mid p(X)\}) \geq 1.$$

$\text{SUM}(\{X \mid p(X)\}) \geq 1$ above denotes a constraint, say A, where $A_d = \{p(1), p(2), p(-1)\}$ and $A_c = \{\{p(1)\}, \{p(2)\}, \{p(1), p(2)\}, \{p(2), p(-1)\}, \{p(1), p(2), p(-1)\}\}$. As shown in [21], P has no answer set. In our unfolding, the approximation P' consists of

$$p(1). \quad p(-1) \leftarrow p(2). \quad p(2) \leftarrow \phi.$$
$$r_1:\ \phi \leftarrow p(1), p(2), p(-1).$$
$$r_2:\ \phi \leftarrow p(1), p(2), \texttt{not}\ p(-1).$$
$$r_3:\ \phi \leftarrow p(-1), p(2), \texttt{not}\ p(1).$$
$$r_4:\ \phi \leftarrow p(1), \texttt{not}\ p(2), \texttt{not}\ p(-1).$$
$$r_5:\ \phi \leftarrow p(2), \texttt{not}\ p(1), \texttt{not}\ p(-1).$$

where we have labeled the rules with ϕ as the head.

For example, $\Delta_{r_5} = \{r_2, r_3\}$. After weakening, we obtain

$$p(1). \quad p(-1) \leftarrow p(2). \quad p(2) \leftarrow \phi.$$
$$\phi \leftarrow p(1), p(2), p(-1).$$
$$\phi \leftarrow p(1), p(2).$$
$$\phi \leftarrow p(-1), p(2).$$
$$\phi \leftarrow p(1), \texttt{not}\ p(-1).$$
$$\phi \leftarrow p(2).$$

where apparently the last rule subsumes the first three with ϕ as the head. As the result, $U_{pos}(P) = \{p(1).\ p(-1) \leftarrow p(2).\ p(2) \leftarrow \phi.\ \phi \leftarrow p(1), \texttt{not}\ p(-1).\ \phi \leftarrow p(2)\}$.

For this example, the program before and after weakening are equivalent in the sense that either of them has any answer set. However, the following example shows that unfolding without weakening is too strong, because a unfolded normal program before weakening may lose answer sets.

Example 4. Let $P = \{a \leftarrow (\{a\}, \{\emptyset, \{a\}\}).\}$. P has a unique answer set $\{a\}$. while $U_{pos}(P)$ consists of a single rule $\{a \leftarrow .\}$, P' before weakening is $\{a \leftarrow \phi.\ \phi \leftarrow \texttt{not}\ a.\ \phi \leftarrow a\}$, which has no answer set.

Proposition 2. *For any general program P and the transformation from P to $U_{pos}(P)$, we have $AS(P) = AS_{|At(P)}(U_{pos}(P))$, where $U_{pos}(P)$ is polynomial in the size of P.*

Although the resulting program takes a polynomial size in the size of P, the transformation itself could be very expensive. Let A be a c-atom, $d = |A_d|$ and $k = |A_c|$. Step One generates k normal rules to approximate A. In Step Two, each operation adds (at most) d more rules for each rule whose body consists of d elements. Thus, the entire weakening process on the rules resulting from a c-atom A is bounded by $k(d!)$.

Call-consistent condition: We can formulate a *call-consistent* condition [6] and describe sufficient conditions for the class of basic positive programs to possess an answer set and multiple answer sets, respectively. This is an an important class, since it includes basic programs under the by-complement semantics.

Let P be a basic positive program consisting of rules of the form

$$A \leftarrow (D_1, C_1), ..., (D_n, C_n). \tag{3}$$

To be consistent with the original definition of stable model [8], we assume that a constraint $\bot \leftarrow body$ in a program is already replaced by a rule with an elementary head: $(\{f\}, \{\{f\}\}) \leftarrow body, (\{f\}, \{\emptyset\})$, where f is a new symbol.

Let A be a c-atom and $\pi \in A_c$. Let $\Sigma \subseteq A_c$ such that $\pi \in \Sigma$, every element in Σ is a superset of π, and Σ has a maximum element (under set inclusion). Denote by ξ_Σ the maximum element in Σ. Σ is said to be a *complete set* for π if $\{\rho - \pi \mid \rho \in \Sigma\}$ is the power set of $\xi_\Sigma - \pi$.

For example, suppose $\pi = \{a\}$ and $\Sigma = \{\{a\}, \{a, b\}, \{a, c\}, \{a, c, b\}\}$. ξ_Σ is $\{a, b, c\}$, and Σ is a complete set for π, since after removing a from each set in Σ, the resulting set is the power set of $\{b, c\}$.

We are interested in *maximal* complete sets for π. A complete set Σ for π is maximal if there is no complete set Σ' for π such that $\Sigma \subset \Sigma'$. Note that a maximal complete set may not be unique. For example, with c-atom $A = (\{a, b, c\}, \{\{a\}, \{a, b\}, \{a, c\}\})$ and $\pi = \{a\}$, $\{\{a\}, \{a, b\}\}$ and $\{\{a\}, \{a, c\}\}$ are both maximal complete sets for π.

We now construct a *dependency graph* G_P as follows: for each rule of the form (3) in P, there is a positive edge from atom A to each atom $b \in \pi$, for all $\pi \in C_i$, $1 \leq i \leq n$; and a negative edge from A to each $d \in D_i$, $1 \leq i \leq n$, such that $\exists \pi \in C_i$ such that $d \notin \xi_\Sigma$, for some maximal complete set Σ for π.

We say that P has an *odd loop* if there is a path in G_P from an atom to itself via an odd number of negative edges, and P has an *even loop* if there is a path in G_P from an atom to itself via an even number of negative edges. P is said to be *call-consistent* if P has no odd loops.

Example 5. For the program P in Example 4, there are no negative edges in G_P. Suppose $P' = \{p \leftarrow (D, C)\}$, where $D = \{a, b, c\}$ and $C = \{\{a\}, \{a, b\}, \{a, c\}\}$. Then, G'_P contains a negative edge from p to b and another one from p to c, along with the obvious positive edges.

Theorem 1. *Let P be a basic positive program. (i) P has an answer set if P is call-consistent. (ii) P has more than one answer set only if P has an even loop.*

Example 6. To illustrate the point (ii) above, consider the following program.

$$p \leftarrow . \quad a \leftarrow (\{p, b\}, \{\{p\}\}). \quad b \leftarrow (\{p, a\}, \{\{p\}\}).$$

The program has two answer sets $\{p, a\}$ and $\{p, b\}$. Then, according to the theorem, there must exist an even loop in its dependency graph. Indeed, the path from a to b negatively, and then from b to a negatively, is an even loop.

4.3 Unfolding Negative C-atoms in Rule Bodies

Unfolding negative c-atoms is formulated for answer sets by reduct. Given a program P, the unfolded program will be denoted by $U_{neg}(P)$.

For each rule $r \in P$ of the form

$$head(r) \leftarrow pos(r), \texttt{not } (D_1, C_1), ..., \texttt{not } (D_n, C_n). \tag{4}$$

where $n \geq 0$, we will have a rule in $U_{neg}(P)$, which is obtained by replacing each negative c-atom $\texttt{not } (D_j, C_j)$ in (4), where $C_j = \{\pi_{j_1}, ..., \pi_{j_k}\}$, by a conjunction $\texttt{not } \psi_{j_1}, ..., \texttt{not } \psi_{j_k}$, where ψ_{j_i}'s are new symbols; in addition, for each $1 \leq j \leq n$, we will have the following rules in $U_{neg}(P)$:

$$\psi_{j_i} \leftarrow body_{j_i}. \qquad \text{for each } j_1 \leq j_i \leq j_k$$

where $body_{j_i} = \pi_{j_i} \cup \{\texttt{not } a \mid a \in D_j \setminus \pi_{j_i}\}$.

Example 7. Consider the following program P from [21]:

$$c \leftarrow \texttt{not } 1\{a, b\}1. \qquad a \leftarrow c. \qquad b \leftarrow a.$$

Note that $1\{a, b\}1$ represents the constraint $(\{a, b\}, \{\{a\}, \{b\}\})$. P has $\{a, b, c\}$ as its unique answer set by reduct. The unfolded program is:

$$c \leftarrow \texttt{not } \psi_1, \texttt{not } \psi_2. \quad \psi_1 \leftarrow a, \texttt{not } b. \quad \psi_2 \leftarrow b, \texttt{not } a. \quad a \leftarrow c. \quad b \leftarrow a.$$

Note that the unique answer set above is not an answer set by complement for P. In fact, the program has no answer sets by complement. As noticed in [21], an answer set by reduct may not be well-supported. The unfolded program provides a technical explanation to this phenomenon.

As noted in [21], for the program P above, $\{a, b, c\}$ is not a minimal model, since $\{b\}$ is a model. It is also known that every answer set by complement is an answer set by reduct, but the reverse does not hold. Thus, answer sets by complement can be computed by first computing answer sets by reduct, and then checking the minimality. Such an extra level of minimality checking sometimes may push the complexity to a higher level (e.g., this is the case from normal programs to disjunctive programs). An interesting question arises: Is the complexity of the semantics by complement higher than that of the semantics by reduct? We will see in Section 6 that the answer is NO.

Proposition 3. *For any general program P, the transformation from P to $U_{neg}(P)$ takes polynomial time and, under the by-reduct semantics, $AS(P) = AS_{|At(P)}(U_{neg}(P))$.*

5 Unfolding Disjunction

When a c-atom A appears in the head of a rule, there may be multiple specified solutions in A_c, and thus an answer set may not be a minimal model. Therefore, the traditional method of defining answer sets for disjunctive programs, namely requiring a set of atoms M to be a minimal model of P^M, is not applicable. On the other hand, when the head of a rule is a disjunction of c-atoms, minimal truth in these c-atoms is desirable, and when every c-atom in a disjunctive program is elementary, the semantics should reduce to the stable model semantics [9].

In this section, we describe unfolding of disjunction in the head, based on which we define answer sets for disjunctive programs. Given a disjunctive program P, the resulting program will be denoted $U_{dis}(P)$.

Consider any rule r in P of the form

$$(D_1, C_1) \vee ... \vee (D_k, C_k) \leftarrow body(r). \tag{5}$$

where $k \geq 1$. Let $C_i = \{\pi_{i_1}, ..., \pi_{i_m}\}$, where $1 \leq i \leq k$. The rule r can be unfolded to yield the following rules in $U_{dis}(P)$ (below, i ranges over $[1..k]$ if not quantified).

$\Phi_1 \vee ... \vee \Phi_k \leftarrow body(r).$
$1\{\xi_{\pi_{i_1}}, ..., \xi_{\pi_{i_m}}\}1 \leftarrow \Phi_i.$ for each $i \in [1..k]$,
$\Phi_i \leftarrow \xi_{\pi_{i_j}}.$ for each $i \in [1..k]$ and each $\pi_{i_j} \in C_i$,
$h \leftarrow \xi_{\pi_{i_j}}.$ for each $\pi_{i_j} \in C_i$, and each $h \in \pi_{i_j}$,
$\bot \leftarrow \xi_{\pi_{i_j}}, d, \Phi_i.$ for each $\pi_{i_j} \in C_i$, and each $d \in D_i \setminus \pi_{i_j}$, and
$\xi_{\pi_{i_j}} \leftarrow h_1, ..., h_v, \texttt{not}\ d_1, ..., \texttt{not}\ d_u.$
for each $\pi_{i_j} \in C_i$ where $h_p \in \pi_{i_j}, d_q \in D_i \setminus \pi_{i_j}$

where Φ_i and $\xi_{\pi_{i_j}}$ are all new symbols.

In the unfolding of disjunction, we first express a disjunction in the head by a disjunction of named c-atoms; a c-atom in the head is satisfied if and only if exactly one admissible set in it is satisfied (the if part takes a weaker form that the satisfaction of any admissible set in it leads to the satisfaction of the c-atom); then, the meaning of an admissible set is encoded by an iff definition of the new symbol representing an admissible set.

It's clear that unfolding of disjunction is a polynomial time process in the size of P.

Example 8. Consider the following rule where $D = \{sunny, windy, raining\}$.

$$(D, \{\{sunny\}\}) \vee (D, \{\{windy\}, \{raining\}, \{windy, raining\}\}) \leftarrow .$$

This rule can be unfolded to

$pleasant \vee annoying \leftarrow .$ $\quad 1\{nice\}1 \leftarrow pleasant.$ $\quad pleasant \leftarrow nice.$
$sunny \leftarrow nice.$ $\quad nice \leftarrow sunny, \texttt{not}\ windy, \texttt{not}\ rainning$
$\bot \leftarrow nice, windy.$ $\quad \bot \leftarrow nice, raining.$
.....

where the variables $pleasant$, $annoying$, and $nice$ are all new symbols.

We now define answer sets for disjunctive programs with c-atoms.

Definition 5. *Let P be a disjunctive program and M be a set of atoms.*

Answer set by reduct: *Let $P' = U_{neg} \circ U_{pos} \circ U_{dis}(P)$. M is an* answer set by reduct *for P iff there is an answer set M' for P' such that $M = M'|_{At(P)}$.*

Answer set by complement: *Let P' be the complement of P, and $P'' = U_{pos} \circ U_{dis}(P')$. M is an* answer set by complement *for P iff there is an answer set M' for P'' such that $M = M'|_{At(P)}$.*

Theorem 2. *(i) If P is a general program, then the answer sets defined in Def. 5 coincide with those defined in Def. 4.*

(ii) If P is a disjunctive program, where any c-atom appearing in any rule head is elementary, then any answer set by complement for P is a minimal model.

(iii) If P is a disjunctive program where every c-atom is elementary, then the answer sets by reduct for P coincide with those by complement for P, and coincide with the stable models of P as defined in [9].

The result in (i) states that our semantics for disjunctive programs is a faithful extension of the semantics for non-disjunctive programs. The result stated in (ii) gives a condition for an answer set by complement to be minimal. In (iii), our semantics reduces to the standard stable model semantics for (conventional) disjunctive programs.

Head cycle-free condition: We can formulate a *head cycle-free* condition for a disjunctive program P. Let P consist of rules r of the form

$$(A_1, B_1) \vee ... \vee (A_k, B_k) \leftarrow (D_1, C_1), ..., (D_n, C_n), neg(r). \tag{6}$$

We construct a *positive dependency graph* G_P as follows: there is a positive edge from an atom p to an atom q if there is a rule $r \in P$ of the form (6) such that $p \in \xi$ for some $\xi \in B_i$, $1 \leq i \leq k$, and $q \in \pi$ for some $\pi \in C_j$, $1 \leq j \leq n$.

Two atoms a_1 and a_2 are said to be *head sharing* if there is a rule of the form 6 such that $a_1 \in B_i$ and $a_2 \in B_j$, for some i and j such that $1 \leq i, j \leq k$ and $i \neq j$. A *head cycle* refers to a path in G_P that goes through two *head-sharing* atoms. G_P is said to be *head cycle-free* if there is no head cycle in it.

Below, by *shifting*, we mean that, given program P, any disjunctive rule $r \in P$, of the form $A_1 \vee ... \vee A_k \leftarrow body(r)$ is replaced by the collection of rules

$$A_i \leftarrow body(r), \texttt{not } A_1, ..., \texttt{not } A_{i-1}, \texttt{not } A_{i+1}, ..., \texttt{not } A_k.$$

where $1 \leq i \leq k$. Let us denote the resulting program by P_{shift}.

Theorem 3. *Let P be a disjunctive program. If G_P is head cycle-free, then P can be reduced to a general program by shifting such that $AS(P) = AS(P_{shift})$.*

6 Complexity

Theorem 4. *For the following classes of programs with c-atoms, the problem of deciding whether an answer set exists for a program P in that class is NP-complete.*

(i) P *is a condition-free program.*
(ii) P *is a positive program.*
(iii) P *is a general program under the by-reduct semantics.*
(iv) P *is a general program under the by-complement semantics.*
(v) P *is a disjunctive program such that* G_P *is head cycle-free.*

Proof.
Hardness: There is a polynomial time reduction from the NP-complete problem CSP to a condition-free program, and the class of condition-free programs is a subclass of all the other classes.

Membership in NP: It suffices to prove (iv) for all the cases from (i) to (iv). For (v), the conclusion follows from Theorem 3 and the result on shifting [1].

To prove (iv), let's guess a set of atoms I for which we are to verify deterministically in polynomial time whether it is an answer set by complement for P. We can do this by checking whether $I = T_P^\infty(\emptyset, I)$, with a straightforward extension of conditional satisfaction to negative c-atoms: for any set of atoms S, $S \models_I \texttt{not}\ A$ iff $S \not\models_I A'$, where A' is the complement of A.

In computing $T_P^\infty(\emptyset, I)$, the whole process terminates after at most n iterations, where $n = |At(P)|$. Thus, we only need to show that each iteration in computing $T_P^n(\emptyset, I)$ takes polynomial time in the size of P. Clearly, this holds if checking the conditional satisfaction of a c-atom A as well as a negative c-atom $\texttt{not}\ A$ takes polynomial time in the size of A.

Let S be a set of atoms and A a c-atom. We check $S \models_I A$ as follows. By definition, $S \models_I A$ if and only if any set between (inclusive) $S \cap A_d$ and $I \cap A_d$ is in A_c. We know that the number of the subsets between $S \cap A_d$ and $I \cap A_d$ is 2^N, where $N = |I \cap A_d - S \cap A_d|$. (Note that $S \cap A_d$ may not be a subset of $I \cap A_d$, in which case, $N = 0$ and $S \cap A_d$ is the only set that need to be considered.) So, if the size of the set $\{S \cap A_d \mid S \cap A_d \in A_c\} \cup \{\pi \mid \pi \in A_c, I \cap \pi \supseteq S \cap A_d\}$ is indeed 2^N, then $S \models_I A$, otherwise $S \not\models_I A$. Clearly, this can be checked in polynomial time in the size of A.

Now consider $\texttt{not}\ A$, with A' being the complement of A. By definition, the following are equivalent to $S \models_I \texttt{not}\ A$

(i) $S \models_I A'$;
(ii) $S \not\models A'$, and for any S' such that $S \cap A_d \subseteq S' \subseteq I \cap A_d$, $S' \cap A_d \in A_c'$;
(iii) $S \cap A_d \notin A_c$, and for each $\pi \in A_c$, it is not the case that $S \cap A_d \subseteq \pi \subseteq I \cap A_d$.

Clearly, the last statement can be checked in polynomial time in the size of A. □

We can also show

Theorem 5. *For the following classes of programs with c-atoms, the problem of deciding whether an answer set exists for a program P in that class is Σ_2^P-complete.*

(i) P *is a disjunctive program under the by-reduct semantics.*
(ii) P *is a disjunctive program under the by-complement semantics.*

7 Relation to Previous Work

Logic programming with abstract constraints was proposed in [13], where the semantics were studied for the programs with monotone c-atoms. Son et al. [21] treated arbitrary c-atoms. In this paper, we further extended this line of work by treating disjunction. Our treatment is based on unfolding, which was used earlier in [20] to unfold positive c-atoms in rule bodies. We extend this approach by also unfolding the head, including the disjunctive head, and negative c-atoms in rule bodies (for the by-reduct semantics).

Disjunction has been considered in [5, 17], where the head of a rule is a disjunction of elementary c-atoms. We have shown in Theorem 2 that when the head contains only elementary c-atoms, we will get the expected behavior, namely all answer sets by complement are minimal models.

Logic programs with aggregates have gained quite a bit attention lately. Usually, it is assumed that an aggregate only occurs positively in a rule body. By allowing c-atoms to appear anywhere in a disjunctive rule, we are able to represent conditional constraints and disjunctive constraints.

Our complexity results are derived for the language of logic programs with c-atoms, where, given a c-atom A, the size of A_c could be exponential in the size of A_d. It is important to note that this language differs from the language of logic programs with aggregates, where aggregates are expressed by pre-defined aggregate functions over some relational operators. The complexity for the latter has been reported in [19]. To the best of our knowledge, the complexity for programs with aggregates/c-atoms in the head has not been studied before.

8 Final Remarks

In this paper, we have studied the class of disjunctive programs with c-atoms, and various subclasses, where a c-atom can appear anywhere in a rule. We have also provided some complexity results for these classes of programs. We have shown that programs in these classes all have an intuitive, rather direct interpretation in terms of the familiar programs composed of ordinary atoms, which can be obtained by unfolding. We have shown that the insights into the semantical issues for logic programs with c-atoms can often be revealed by their unfolded counterparts.

An interesting question is how to compute answer sets for classes of programs with arbitrary c-atoms. The unfolding approach studied here is intended as a means for theoretical analysis. In practical systems, c-atoms are typically expressed in terms of some pre-defined functions and predicates, such as weight and cardinality constraints, and those involving aggregates. In this case, an implementation supported by special constraint propagation algorithms for effective space pruning is highly desirable, which we are studying at the present time.

Acknowledgment: The work by Jia-Huai You, Li Yan Yuan and Guohua Liu were partially supported by NSERC, and the first two were also assisted from Grant NSFC 60573009. Yi-Dong Shen is supported by the NSFC grants 60673103 and 60421001.

References

1. R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Math. and Artifi cial Intelligence*, 14:53–87, 1994.
2. F. Calimeri, W. Faber, N. Leone, and S. Perri. Declarative and computational properties of logic programs with aggregates. In *Proc. IJCAI'05*, pages 406–411, 2005.
3. T. Dell'Armi, W. Faber, G. Lelpa, and N. Leone. Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in DLV. In *Proc. IJCAI'03*, pages 847–852, 2003.
4. M. Denecker, N. Pelov, and M. Bruynooghe. Ultimate well-founded and stable semantic for logic programs with aggregates. In *Proc. ICLP '01*, pages 212–226, 2001.
5. W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs. In *Proc. JELIA'04*, pages 200–212, 2004.
6. F. Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
7. M. Gelfond. *Representing knowledge in A-Prolog*, chapter 413-451. Springer, 2002.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. 5th ICLP*, pages 1070–1080. MIT Press, 1988.
9. M. Gelfond and V. Lifschitz. Classic negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
10. L. Liu and M. Truszczyński. Properties of programs with monotone and convex constraints. In *Proc. AAAI '05*, pages 701–706, 2005.
11. L. Liu and M. Truszczyński. Properties and applications of programs with monotone and convex constraints. *Journal of Artifi cial Intelligence Research*, 7:299–334, 2006.
12. V. Marek, I. Niemelä, and M. Truszczyński. Logic programs with monotone abstract constraint atoms. *Theory and Practice of Logic Programming*. To appear.
13. V. Marek and M. Truszczyński. Logic programs with abstract constraint atoms. In *Proc. AAAI '04*, pages 86–91, 2004.
14. S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proc. AAAI'90*, pages 25–32, 1990.
15. N. Pelov. *Semantics of Logic Programs with Aggregates*. PhD thesis, Ketholieke Universiteit Leuven, 2004.
16. N. Pelov, M. Denecker, and M. Bruynooghe. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming*. To appear.
17. N. Pelov and M. Truszczyński. Semantics of disjunctive programs with monotone aggregates an operator-based approach. In *Proc. NMR '04*, pages 327–334, 2004.
18. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artifi cial Intelligence*, 138(1-2):181–234, 2002.
19. T.C. Son and E. Pontelli. A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming*. To appear.
20. T.C. Son, E. Pontelli, and I. Elkabani. A translational semantics for aggregates in logic programs. Technical Report CS-2005-006, New Mexico State University, 2005.
21. T.C. Son, E. Pontelli, and P. Tu. Answer sets for logic programs with arbitrary abstract constraint atoms. In *Proc. AAAI'06*, pages 129–134, 2006.