Answer Sets for Logic Programs with Arbitrary Abstract Constraint Atoms *

Tran Cao Son
Computer Science Department
New Mexico State University
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

Enrico Pontelli
Computer Science Department
New Mexico State University
Las Cruces, NM 88003, USA
epontell@cs.nmsu.edu

Phan Huy Tu
Computer Science Department
New Mexico State University
Las Cruces, NM 88003, USA
tphan@cs.nmsu.edu

Abstract

We present two equivalent approaches for defining answer sets for logic programs with *arbitrary* abstract constraint atoms (c-atoms). The first approach uses an immediate consequence operator for answer set checking, whose definition relies on the notion of *conditional satisfaction* of c-atoms w.r.t. a pair of interpretations. The second approach generalizes the notion of *well-supported* models of normal logic programs to programs with c-atoms. We prove that the newly defined semantics coincides with previously introduced semantics for logic programs with monotone c-atoms and extends the original answer set semantics for normal logic programs. We discuss different possibilities for treating negation-as-failure c-atoms and characterize situations in which they yield the same answer sets. We study some properties of answer sets of programs with c-atoms and relate our definition to several semantics for logic programs with aggregates.

Introduction and Motivation

Logic programming under the answer set semantics has recently been presented as an attractive and suitable knowledge representation language for AI research (Baral 2005), since it features several desirable properties for this purpose. Among other things, the language is declarative and has a fairly simple syntax; it is nonmonotonic and is expressive enough for representing several classes of problems in the complexity hierarchy; it has solid theoretical foundations with a large body of building block results (e.g., equivalence between programs, systematic program development, relationships to other non-monotonic formalisms); it also has a large number of efficient computational tools. Please, see, e.g., (Baral 2003; Gelfond & Leone 2002) for more on this.

In recent years, a large number of extensions aimed at improving the usability of logic programming in knowledge representation and reasoning have been proposed or revisited. For example:

- *weight and cardinality constraints* have been introduced to make the representation of several types of constraints easier (Simons, Niemelä, & Soininen 2002);
- *aggregates*, first studied in the context of logic programming in a variety of proposals (e.g., (Kemp & Stuckey 1991; Mumick, Pirahesh, & Ramakrishnan 1990; Gelder 1992)), and further developed in recent years (e.g., (Dell'Armi *et al.* 2003; Denecker, Pelov, & Bruynooghe 2001; Elkabani, Pontelli, & Son 2004; Faber, Leone, & Pfeifer 2004; Gelfond 2002; Pelov 2004; Son & Pontelli 2005)), have been introduced to support representation and reasoning with set operators.

The semantics of these extensions has been defined either indirectly, by translating programs with these extensions to normal logic programs, or directly, by providing new definitions of answer sets for programs with these extensions. Weight and cardinality constraints have been implemented in *Smodels* (Simons, Niemelä, & Soininen 2002). **dlv** and *Smodels* have been extended to deal with aggregates in (Dell'Armi *et al.* 2003) and (Elkabani, Pontelli, & Son 2005) respectively.

Logic programs with *abstract constraint atoms* (or *c-atoms*) have been introduced in (Marek & Truszczyński 2004) as a generalized theoretical framework for different extensions of logic programming, among them weight constraints and aggregates. Intuitively, a c-atom A represents a constraint on models (or answer sets) of the program containing A. Under this view, most of the extensions of logic programming can be viewed as instances of logic programs with c-atoms. The proposal has been further extended to disjunctive logic programs (Pelov & Truszczyński 2004). In these papers, the focus has been on *monotone* c-atoms, where a c-atom A is monotone if, for each pair of interpretations I and I' with $I \subseteq I'$, we have that, if I satisfies A then I' satisfies A as well. In another paper (Liu & Truszczyński 2005), properties of programs with monotone and convex c-atoms have been studied. It is shown that many well-known properties of logic programming under answer set semantics are maintained in the case of programs with c-atoms.

The main advantage of focusing on *monotone* c-atoms lies in that it provides a simple way for defining answer sets of logic programs with c-atoms. However, this restriction does not allow several well-known types of constraints to be directly considered. For example, the aggregate atom $\text{MIN}(\{X \mid p(X)\}) \neq 2$ and the choice atom $1\ \{a, b\}\ 1$ cannot be represented as monotone c-atoms.

In this paper, we overcome these limitations, by considering programs with *arbitrary* (e.g., non-monotone) c-atoms, including c-atoms in the head of the rules. This advances

*Supported by NSF grants 0454066, 0420407, and 0220590.

the state-of-the-art w.r.t. most proposals on aggregations (by allowing, as (Marek & Truszczyński 2004), c-atoms in the heads) and w.r.t. to (Marek & Truszczyński 2004; Liu & Truszczyński 2005), by allowing arbitrary c-atoms in the programs. We present two approaches to define answer sets for logic programs with arbitrary c-atoms. In the first approach, we generalize the notion of *conditional satisfaction* of (Son & Pontelli 2005) to program with c-atoms, as a mean to deal with c-atoms. Using this notion, we define a T_P-like operator for answer set checking. In the second approach, we generalize the notion of *well-supported model* of (Fages 1994) to programs with c-atoms. We prove that the two approaches are equivalent. We show that the newly defined semantics coincides with the previously introduced semantics for monotone c-atoms and extends the original stable model semantics for normal logic programs. We discuss different approaches for treating negation-as-failure c-atoms and characterize situations in which they are equivalent and relate our definitions to several semantics for logic programs with aggregates, as c-atom can encode arbitrary aggregates.

Preliminaries: Syntax, Model, and Satisfaction

We follow the syntax used in (Liu & Truszczyński 2005) to define programs with abstract constraint atoms. Throughout the paper, we assume a fixed propositional language $\mathcal{L}$ with a countable set $\mathcal{A}$ of propositional atoms. An abstract constraint atom (or *c-atom*) is an expression of the form (D, C), where $D \subseteq \mathcal{A}$ is a set of atoms and C is a collection of sets of atoms belonging to D, i.e., $C \subseteq 2^D$. Intuitively, a c-atom (D, C) is a constraint on the set of atoms D and C are its admissible solutions. Given a c-atom $A = (D, C)$, we use A_d and A_c to denote D and C, respectively.

A c-atom of the form $(\{a\}, \{\{a\}\})$ is called an *elementary* c-atom and will be simply written as a. A c-atom of the form $(\mathcal{A}, \emptyset)$, representing the fact that nothing is acceptable, will be denoted by $\bot$. A c-atom A is said to be *monotone* if for every $X \subseteq Y \subseteq A_d$, $X \in A_c$ implies that $Y \in A_c$.

It has been shown in (Marek & Truszczyński 2004) that c-atoms can be used to conveniently represent weight and cardinality constraints (Simons, Niemelä, & Soininen 2002) and various types of aggregates. For example, the c-atom $(\{p(1), p(-2)\}, \{\emptyset, \{p(1)\}, \{p(-2), p(1)\}\})$ represents the aggregate atom $\text{SUM}(\{X|p(X)\}) \geq -1$ (assuming that $p(1)$ and $p(-2)$ are the only two atoms of the form $p(X)$ in $\mathcal{L}$), and the c-atom $(\{p(1), p(-1)\}, \{\{p(1)\}, \{p(-1)\}\})$ can be viewed as the *Smodels* choice atom $1\ \{p(1), p(-1)\}\ 1$. For this reason, we will often use aggregate atoms and/or weight constraints instead of c-atoms in our examples, whenever no confusion is possible.

A *rule* is of the form

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

where A, A_j's are c-atoms. The literals $not\ A_j$ ($k < j \leq n$) are called *negation-as-failure c-atoms* (or *naf-atoms*). For a rule r of the form (1), $head(r)$, $pos(r)$, and $neg(r)$ denote A, $\{A_1, \ldots, A_k\}$, and $\{A_{k+1}, \ldots, A_n\}$, respectively. $body(r)$ denotes the right hand side (w.r.t. $\leftarrow$) of r. A rule r is (*i*) *positive*, if $neg(r) = \emptyset$; (*ii*) *basic*, if $head(r)$ is an elementary c-atom; and (*iii*) a *constraint*, if $head(r) = \bot$.

A *logic program with c-atoms* (or *logic program*, for simplicity)[1] is a set of rules. A program P is called a *basic* program if each rule $r \in P$ is a basic or a constraint rule. P is said to be *positive* if every rule in P is positive. P is *monotone* (resp. *semi-monotone*) if each c-atom occurring in P (resp. as a naf-atom in P) is monotone. Clearly, a monotone program is also semi-monotone.

A set of atoms $S \subseteq \mathcal{A}$ satisfies a c-atom A, denoted by $S \models A$, if $A_d \cap S \in A_c$. S satisfies $not\ A$, denoted by $S \models not\ A$, if $A_d \cap S \notin A_c$.

A set of atoms S satisfies the body of a rule r of the form (1), denoted by $S \models body(r)$, if $S \models A_i$ for $i = 1, \ldots, k$ and $S \models not\ A_j$ for $j = k+1, \ldots, n$. S satisfies a rule r if it satisfies $head(r)$ or it does not satisfy its body.

A set of atoms S is a *model* of a program P if S satisfies every rule of P. M is a minimal model of P if it is a model of P and there is no proper subset of M which is also a model of P. In particular, programs may have more than one *minimal* model (see Example 3).

Given a program P, a set of atoms S is said to support an atom $a \in \mathcal{A}$ if there exists some rule r in P such that $S \models body(r)$, $S \cap head(r)_d \in head(r)_c$ and $a \in S \cap head(r)_d$.

Example 1 Let P_1 be the program

$$p(a).\quad p(b).\quad p(c) \leftarrow q.\quad q \leftarrow \text{COUNT}(\{X \mid p(X)\}) > 2.$$

$\text{COUNT}(\{X|p(X)\})>2$ represents the c-atom $(D, \{D\})$ where $D = \{p(a), p(b), p(c)\}$. P_1 has two models:

$M_1 = \{p(a), p(b)\}$ and $M_2 = \{p(a), p(b), p(c), q\}$.

M_1 is a minimal model while M_2 is not. □

Example 2 Let P_2 be the program

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

where $\text{SUM}(\{X|p(X)\})\geq 1$ represents the c-atom (D, C) where $D = \{p(1), p(2), p(-1)\}$ and $C = \{\{p(1)\}, \{p(2)\}, \{p(1), p(2)\}, \{p(2), p(-1)\}, \{p(1), p(2), p(-1)\}\}$. Because of the first rule, any model of P_2 will need to contain $p(1)$. Note that $\{p(1), p(-1)\}$ and $\{p(1), p(2), p(-1)\}$ are models of P_2 but $\{p(1), p(2)\}$ is not a model of P_2. □

Example 3 Let P_3 be the program

$$p \leftarrow (\{q\}, \{\emptyset\}). \qquad q \leftarrow (\{p\}, \{\emptyset\}).$$

P_3 has three models $\{p\}$, $\{q\}$, and $\{p, q\}$, of which $\{p\}$ and $\{q\}$ are minimal. □

Answer Sets for Basic Programs: Fix-Point Based Approach

In this approach, we follow the standard way to define answer sets, i.e., *(i)* we start with positive programs (Def. 2); and *(ii)* extend it to deal with naf-atoms (Defs. 5-6).

Answer Sets for Basic Positive Programs

Example 3 shows that a basic positive program might have more than one minimal model. This leads us to define a T_P-like operator for answer set checking, whose construction is based on the following observation.

[1] Whenever we want to refer to traditional logic programs (without c-atoms), we will explicitly talk about *normal logic programs*.

Observation 1.[2]: Let P be a normal logic program (without c-atoms) and R, S be two sets of atoms. Let

$$T_P(R,S) = \left\{ a \;\middle|\; \begin{array}{l} \exists r \in P \,:\, a = head(r), \\ pos(r) \subseteq R, neg(r) \cap S = \emptyset \end{array} \right\}$$

Then, M is an answer set of P w.r.t. (Gelfond & Lifschitz 1988) iff $M = I^\omega$, where I^ω is the limit of the monotone sequence of sets of atoms $\langle I_j \rangle_{j=0}^{\omega}$ defined as follows: $I_0 = \emptyset$, and $I_{j+1} = T_P(I_j, M)$ for $j \geq 0$.

As it can be seen in the above observation, the (modified) consequence operator T_P takes two sets of atoms, R and S, as its arguments and generates one set of atoms which could be viewed as the consequences of P given that R is true and S is assumed to be an answer set of P. It is easy to see that T_P is monotone w.r.t. its first argument, i.e., if $R \subseteq V$, then $T_P(R,S) \subseteq T_P(V,S)$. Thus, the sequence $\langle I_j \rangle_{j=0}^{\omega}$ is monotone and converges to I^ω for a given S. We will next show how T_P can be generalized to programs with c-atoms.

Observe that the definition of T_P requires that $pos(r) \subseteq R$ or, equivalently, $R \models pos(r)$. For normal logic programs, this is sufficient to guarantee the monotonicity of $T_P(\cdot, S)$. If this definition is naively generalized to the case of programs with c-atoms, the monotonicity of $T_P(., S)$ is guaranteed only when c-atoms in $pos(r)$ are monotone. To deal with arbitrary c-atoms, we need to introduce the notion of *conditional satisfaction* of a c-atom as follows.

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 $I \in A_c$.*

We say that R conditionally satisfies a set of c-atoms V w.r.t. S, denoted by $R \models_S V$, if $R \models_S A$ for every $A \in V$. This allows us to generalize the operator T_P defined in Observation 1 as follows. For two sets of atoms S and R and a positive basic program P, let

$$T_P(R,S) = \left\{ a \;\middle|\; \begin{array}{l} \exists r \in P \,:\, R \models_S pos(r), \\ head(r) = (\{a\}, \{\{a\}\}) \end{array} \right\}$$

The following proposition holds.

Proposition 1 *Let M be a model of P, and let $S \subseteq U \subseteq M$. Then $T_P(S,M) \subseteq T_P(U,M) \subseteq M$.*

The above proposition states that T_P is monotone for subsets of M w.r.t. its first argument (given that the second argument is fixed). Hence, the sequence $T_P^i(\emptyset, M)$ where $T_P^0(\emptyset, M) = \emptyset$ and $T_P^{i+1}(\emptyset, M) = T_P(T_P^i(\emptyset, M), M)$, converges to a fixpoint. We denote this fixpoint with $T_P^\infty(\emptyset, M)$ and define:

Definition 2 *Let M be a model of a basic positive program P. M is an* answer set *of P if $M = T_P^\infty(\emptyset, M)$.*

[2]For a normal logic program rule r,

$$a \leftarrow a_1, \ldots, a_n,\ not\ b_1, \ldots,\ not\ b_m$$

$head(r)$, $pos(r)$, and $neg(r)$ denote a, $\{a_1, \ldots, a_n\}$, and $\{b_1, \ldots, b_m\}$, respectively.

Observe that the constraints present in P do not contribute to the construction performed by T_P; nevertheless, the requirement that M should be a model of P implies that all the constraints will have to be satisfied by each answer set.

We illustrate Definition 2 in the next examples.

Example 4 Let P_1 be the program in Example 1.

- $M_1 = \{p(a), p(b)\}$ is an answer set of P_1 since: $T_{P_1}^0(\emptyset, M_1) = \emptyset$, $T_{P_1}^1(\emptyset, M_1) = \{p(a), p(b)\} = M_1$, and $T_{P_1}^2(\emptyset, M_1) = M_1$.
- $M_2 = \{p(a), p(b), p(c), q\}$ is not an answer set of P_1, since: $T_{P_1}^0(\emptyset, M_2) = \emptyset$, $T_{P_1}^1(\emptyset, M_2) = \{p(a), p(b)\} = M_1$, and $T_{P_1}^2(\emptyset, M_2) = M_1$. □

The next example shows that not every positive program has an answer set.

Example 5 Consider P_2 (Example 2). Since answer sets of positive programs are *minimal* models (Proposition 5) and $M = \{p(1), p(-1)\}$ is the only minimal model of P_2, we have that M is the only potential answer set of P_2. Because $T_{P_2}^0(\emptyset, M) = \emptyset$, $T_{P_2}^1(\emptyset, M) = \{p(1)\}$, and $T_{P_2}^2(\emptyset, M) = \{p(1)\}$ (since $\{p(1)\} \not\models_M \textsc{Sum}(\{X \mid p(X)\}) \geq 1$) we can conclude that M is not an answer set of P_2, i.e., P_2 does not have any answer sets. □

Answer Sets for Basic Programs

We will now define answer sets for basic programs (with negation). It is interesting to note that various extensions of logic programming (e.g., weight constraints, aggregates) support negation-as-failure atoms by replacing each $not\ A$ with an A', where A' is obtained from A by replacing the predicate relation of A with its "negation". For example, following this approach, $not\ 1\ \{a, b\}\ 1$ is replaced by $(\{a, b\}, \{\emptyset, \{a, b\}\})$ and $not\ \textsc{Sum}(\{X \mid p(X)\}) \neq 5$ is replaced by $\textsc{Sum}(\{X \mid p(X)\}) = 5$. On the other hand, in (Marek & Truszczyński 2004), naf-atoms are dealt with by using a form of *program reduct* (in the same spirit as (Gelfond & Lifschitz 1988)).

Following these perspectives, we study two different approaches for dealing with naf-atoms, described in the next two subsections. It is worth mentioning that both approaches coincide for monotone programs (Proposition 3).

Negation-as-Failure by Complement To define this notion for programs with c-atoms, we first define the notion of *complement* of a c-atom as follows.

Definition 3 *The complement of a c-atom A is the c-atom $(A_d, 2^{A_d} \setminus A_c)$.*

We next define the complementary program of P.

Definition 4 *Given a basic program P, we define $\mathcal{C}(P)$ to be the program obtained from P by replacing every occurrence of $not\ A$ in P with the complement of A.*

Obviously, $\mathcal{C}(P)$ is a basic positive program, whose answer sets have been defined in Def. 2. This allows us to define the answer sets of basic programs as follows.

Definition 5 *A set $M \subseteq \mathcal{A}$ is an* answer set by complement *of a basic program P iff it is an answer set of $\mathcal{C}(P)$.*

Example 6 Let P_4 be the program

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

$\mathcal{C}(P_4)$ is the program

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

This program does not have an answer set (w.r.t. Definition 2); thus P_4 does not have an answer set by complement. □

Negation-as-Failure by Reduct Another approach for dealing with naf-atoms is to adapt the Gelfond-Lifschitz reduction of normal logic programs (Gelfond & Lifschitz 1988) to programs with c-atoms—this approach has been considered in (Marek & Truszczyński 2004). We can generalize this to programs with arbitrary c-atoms as follows. For a basic program P and a set of atoms M, the *reduct of P w.r.t. M* (P^M) is the set of rules obtained by

1. removing all rules containing $not\ A$ s.t. $M \models A$; and
2. removing all naf-atoms from the remaining rules.

Obviously, the program P^M is a positive program. Thus, we can define answer sets for P as follows.

Definition 6 *A set of atoms M is an* answer set by reduct *of P iff M is an answer set of P^M (w.r.t. Definition 2).*

The next example shows that this approach might lead to different answer sets than the case of negation by complement (for non-monotone programs).

Example 7 Consider the program P_4 from Example 6. Let $M = \{a,b,c\}$. The reduct of P_4 w.r.t. M is the program

$$c. \qquad a \leftarrow c. \qquad b \leftarrow a.$$

which has M as its answer set, i.e., M is an answer set by reduct of P_4. □

One drawback of the negation by reduct approach is the fact that it might lead to non-minimal answer sets in the presence of non-monotone atoms. For instance, if we replace the atom $\text{COUNT}(\{X \mid p(X)\}) > 2$ in P_1 with $not\ \text{COUNT}(\{X \mid p(X)\}) \leq 2$, the new program will admit $\{p(a), p(b), p(c), q\}$ as an answer set by reduct. Nevertheless, this indicates that, for programs with c-atoms, there might be different ways to treat naf-atoms. This problem has been mentioned in (Ferraris 2005). Investigating other methodologies for dealing with naf-atoms is an interesting topic of research, that we plan to pursue in the future.

Answer Sets for Basic Programs: Level Mapping Based Approach

The definitions of answer sets in the previous section can be viewed as a generalization of the answer set semantics for normal logic programs, in the sense that they rely on a fixpoint operator (defined for positive programs). In this subsection, we discuss another approach for defining answer sets for programs with c-atoms, which is based on the notion of *well-supported models*.

The notion of well-supported models for normal logic programs has been introduced in (Fages 1994). It provides an interesting characterization for answer sets as defined in (Gelfond & Lifschitz 1988). Intuitively, a model M of a program P is a well-supported model iff there exists a level mapping, from atoms in M to the set of positive integers, such that each atom $a \in M$ is supported by a rule r whose body is satisfied by M and the level of each positive atom in $body(r)$ is strictly smaller than the level of a.[3] Fages proved that answer sets are well-supported models and vice versa (Fages 1994). The notion of well-supportedness has been considered for dynamic logic programs (Banti *et al.* 2005). Level mapping has been used as an effective tool to analyze different semantics of logic programs in a uniform way (Hitzler & Wendt 2005).

We will show that the notion of well-supported models can also be ported to programs with c-atoms. Key to the formulation of this notion is the answer to the question "what will be the level of a c-atom A given a set of atoms M and a level mapping L of M?" On one hand, one might argue that the level mapping of A should be defined independently from the mapping of the other atoms. On the other hand, it is reasonable to assume that the level of A depends on the levels of the atoms in A_c, since the satisfaction of A (w.r.t. a given interpretation) depends on the satisfaction of the elements in A_d. The fact that every existing semantics of programs with c-atoms (or other extensions) evaluates the truth value of a c-atom A based on the truth value assigned to elements of A_d convinced us to adopt the second view. It is worth to mention that this view also allows us to avoid circular justifications of elements of a well-supported model[4].

Let M be a set of atoms, l be a mapping from M to positive integers, and let A be a c-atom. We define

$$L(A,M) = \texttt{min}(\{H(X) \mid X \in A^c, X \subseteq M, X \models_M A\})$$

where $H(X) = \texttt{max}(\{l(a) \mid a \in X\})$. We assume that $\texttt{max}(\emptyset) = 0$, while $\texttt{min}(\emptyset)$ is undefined.

Definition 7 (Well-supported) *Let P be a basic program. A model M of P is said to be* well-supported *iff there exists a level mapping l s.t. for each $a \in M$, P contains a rule r with $head(r) = (\{a\},\{\{a\}\})$, $M \models body(r)$, and for each $A \in pos(r)$, $L(A,M)$ is defined and $l(a) > L(A,M)$.*

The next proposition generalizes Fages's result to answer sets by reduct for programs with c-atoms.

Proposition 2 *A set M of atoms is an answer set by reduct of a basic program P iff it is a well-supported model of P.*

As we have seen in the previous section, different ways to deal with naf-atoms result in different semantics for programs with c-atoms. This indicates that certain adjustments have to be made to apply Proposition 2 for answer sets by complement. Indeed, we can show that for a basic program P, each answer set by complement of P is a well-supported model of $\mathcal{C}(P)$ and vice versa.

Properties of Answer Sets of Basic Programs

We will now show that the notion of answer sets for basic programs with c-atoms is a natural generalization of the no-

[3] This implicitly means that $pos(r) \subseteq M$ and $neg(r) \cap M = \emptyset$, i.e., naf-atoms are dealt with by reduct.

[4] In the first view, the program $\{a \leftarrow b.\ \ b \leftarrow a.\ \ a \leftarrow A.\}$ where $A = (\{a,b\},\{\emptyset,\{a,b\}\})$ has $M = \{a,b\}$ as a well-supported model with $l(a)=1$, $l(b)=2$, $l(A)=0$; a is true because A is true, which is true because a and b are both true. In our opinion, $\{a,b\}$ should not be viewed as an answer set of this program.

tions of answer sets for normal logic programs. We prove that answer sets of basic positive programs are minimal and supported models and characterize situations in which these properties hold for basic programs. We begin with a result stating that, for the class of semi-monotone programs, the two approaches for dealing with naf-atoms coincide.

Proposition 3 *For every basic program P, each answer set by complement of P is an answer set by reduct of P. Furthermore, if P is semi-monotone, then each answer set by reduct of P is also an answer set by complement of P.*

This proposition implies that, in general, the negation-as-failure by complement approach is more 'skeptical' than the negation-as-failure by reduct approach, in that it may accept fewer answer sets. Furthermore, Examples 6 and 7 show that a *minimal* (w.r.t. set inclusion) answer set by reduct is not necessarily an answer sets by complement of a program.

Let P be a normal logic program and *c-atom*(P) be the program obtained by replacing each occurrence of an atom a in P with $(\{a\}, \{\{a\}\})$. Since $(\{a\}, \{\{a\}\})$ is a monotone c-atom, *c-atom*(P) is a semi-monotone program. Proposition 3 implies that answer sets by reduct of *c-atom*(P) are answer sets by complement and vice versa. In the next proposition, we prove that the notion of answer sets for programs with c-atoms preserves the notion of answer set for normal logic programs, in the following sense[5].

Proposition 4 (Preserving Answer Sets) *For a normal logic program P, M is an answer set (by complement or by reduct) of c-atom(P) iff M is an answer set of P (w.r.t. Definition in (Gelfond & Lifschitz 1988)).*

In the next proposition, we study the minimality and supportedness properties of answer sets of basic programs.

Proposition 5 (Minimality of Answer Sets)

1. *Every answer set by complement of a basic program P is a minimal model of P.*
2. *Every answer set by reduct of a basic, semi-monotone program P is a minimal model of P.*
3. *Every answer set (by complement/reduct) of a basic program P supports each of its members.*

Answer Sets for General Programs

In this section, we define answer sets for general programs (i.e., programs where the rule heads are arbitrary c-atoms). Our approach is to convert a program with c-atoms in the head, P, into a collection of basic programs, whose answer sets are defined as answer sets of P. To simplify the presentation, we will use the phrase "an answer set of a basic program" to refer to either an answer set by complement or an answer set by reduct of the program. The distinction will be stated clearly whenever it is needed.

Let P be a program and $r \in P$. For each $V \in head(r)_c$, the *instance* of r w.r.t. V, is the set of rules consisting of

1. a rule $b \leftarrow body(r)$, for each $b \in V$, and
2. a constraint $\bot \leftarrow d, body(r)$, for each $d \in head(r)_d \setminus V$.

An *instance of P* is a program obtained by replacing each rule of P with one of its instances. It is easy to see that an instance of P is a basic program. This allows us to define answer sets of general programs as follows.

Definition 8 *Let P be a general program. M is an answer set of P iff M is an answer set of one of its instances.*

Observe that if P is a basic program then P is its unique instance. As such, the notion of answer sets for general programs is a generalization of the notion of answer sets for basic programs. It can be shown that Proposition 3 also holds for general programs.

Related Works and Discussions

In this section, we relate our work to some recently proposed extensions of logic programming. *Logic programs with c-atoms*, as defined in this paper, have been introduced in (Marek & Truszczyński 2004). One of the main differences between our work and the work of (Marek & Truszczyński 2004) is that we consider *arbitrary c-atoms* while they only deal with *monotone c-atoms*. On the other hand, we did not consider disjunctive programs with c-atoms, as done in (Pelov & Truszczyński 2004).

The proposed operator T_P differs from the nondeterministic one-step provability operator T_P^{nd} of (Marek & Truszczyński 2004), in that it is deterministic and is applied only to basic positive programs. We have not investigated the portability of several properties of answer sets for normal logic programs to answer sets for programs with c-atoms as presented in (Liu & Truszczyński 2005). As we will see later, Proposition 6 implies that the results proved in (Liu & Truszczyński 2005) will be valid for the class of monotone programs (w.r.t. our answer set definition). We do, however, focus on the use of well-supported models and level mapping in studying answer sets for programs with c-atoms.

We will next present a result that shows that our approach to define answer sets for monotone programs coincides with that of (Marek & Truszczyński 2004).

Proposition 6 *For every monotone program P, a set of atoms M is an answer set of P w.r.t. Definition 8 iff M is an answer set of P w.r.t. (Marek & Truszczyński 2004).*

As discussed earlier, c-atoms can be used to represent several extensions of logic programs, among them weight constraints and aggregates. Intuitively, an aggregate atom α (e.g., see (Elkabani, Pontelli, & Son 2004; Faber, Leone, & Pfeifer 2004)) can be represented by a c-atom (D, C) where D consists of all atoms occurring in the set expression of α and $C \subseteq 2^D$ such that every $X \in C$ satisfies α (see Examples 1-2). As indicated in (Marek & Truszczyński 2004), many proposals do not allow aggregates in the head of rules. Our general programs allow c-atoms in the head.

With regards to naf-atoms, some proposals (e.g., (Elkabani, Pontelli, & Son 2004)) do not allow aggregates to occur in naf-atoms. The proposal in (Faber, Leone, & Pfeifer 2004) treats naf-atoms by complement, although a reduction is used in defining the semantics, while (Ferraris 2005) argues that, under a different logic, naf-atoms might require a different treatment.

[5]Together with Proposition 3, this proposition implies that answer sets of P are answer sets of $\mathcal{C}$(*c-atom*(P)). Thus, normal logic programs could be represented by positive basic programs.

We will now present some propositions which relate our work to the recent works on aggregates. We can prove [6]:

Proposition 7 *For a program with aggregates P, if M is an answer set by complement of P then it is an answer set of P w.r.t. (Faber, Leone, & Pfeifer 2004) and (Ferraris 2005).*

The proposal presented in (Pelov 2004; Denecker, Pelov, & Bruynooghe 2001) deals with aggregates by using approximation theory and three-valued logic, building the semantics on the three-valued immediate consequence operator Φ_P^{aggr}, which maps three-valued interpretations into three-valued interpretations of the program. This operator can be viewed as an operator which maps pairs of set of atoms (R, S) where $R \subseteq S$ into pairs of set of atoms (R', S') with $R' \subseteq S'$. The authors show that the ultimate approximate aggregates provide the most precise semantics for logic programs with aggregates. Let $\Phi_P^{aggr}(R, M) = (\Phi^1(R, M), \Phi^2(R, M))$. We next relate T_P to Φ_P^{aggr}.

Proposition 8 *Let P be a positive program with aggregates and R and M be two set of atoms such that $R \subseteq M$. Then, $T_P(R, M) = \Phi^1(R, M)$.*

The above proposition, together with the fact that the evaluation of the truth value of aggregate formulas in (Denecker, Pelov, & Bruynooghe 2001) treats naf-atoms by complement, implies that, for a program with aggregates P, answer sets by complement of P (w.r.t. Definition 2) are ultimate stable models of P (Denecker, Pelov, & Bruynooghe 2001) and vice versa. Together with the results in (Son & Pontelli 2005), it also implies that T_P is a generalization of the immediate consequence operator for programs with aggregates in (Son & Pontelli 2005).

Conclusions and Future Work

The goal of this paper is to explore a general logic programming framework based on the use of arbitrary constraint atoms. We provide two characterizations of answer set semantics for programs with arbitrary constraint atoms, which are equivalent for the class of semi-monotone programs. The first approach is based on a generalization of the immediate consequence operator for programs with aggregates of (Son & Pontelli 2005) and the second is built on a generalization of the notion of well-supported models of (Fages 1994). We discuss two methodologies for treating naf-atoms and identify the class of semi-monotone programs, on which the two approaches for dealing with naf-atoms coincide. We prove that the newly proposed semantics coincides with the semantics proposed in (Marek & Truszczyński 2004) for monotone programs and relate our work to various works on logic programs with aggregates.

As future work, we propose to further investigate the relationships between different approaches for dealing with naf-atoms; e.g., we would like to investigate the existence of a program reduction which is equivalent to the complement and the generalization of the notion of well-supported model to general programs; we will also explore implementation methodologies based on modifications of *Smodels*.

[6] Abusing the notation, we use a single symbol to denote a program in different notations.

References

Baral, C. 2003. *Knowledge Representation, reasoning, and declarative problem solving with Answer sets*. Camb. Uni. Press.

Baral, C. 2005. From Knowledge to Intelligence: Building Blocks and Applications. Invited Talk, AAAI, `www.public.asu.edu/~cbaral/aaai05-invited-talk.ppt`.

Banti, F.; Alferes, J.; Brogi, A.; and Hitzler, P. 2005 The Well Supported Semantics for Multidimensional Dynamic Logic Programs. In *LPNMR*, 356–368.

Dell'Armi, T.; Faber, W.; Ielpa, G.; Leone, N.; and Pfeifer, G. 2003. Aggregate functions in disj. logic programming: semantics, complexity, and implementation in DLV. In *IJCAI*, 847–852.

Denecker, M.; Pelov, N.; and Bruynooghe, M. 2001. Ultimate well-founded and stable semantics for logic programs with aggregates. In *ICLP*, 212–226.

Elkabani, I.; Pontelli, E.; and Son, T. C. 2004. Smodels with CLP and its applications: A simple and effective approach to aggregates in ASP. In *ICLP*, 73–89.

Elkabani, I.; Pontelli, E.; and Son, T. C. 2005. Smodels^{A} - A System for Computing Answer Sets of Logic Programs with Aggregates. In *LPNMR*, 427–431.

Faber, W.; Leone, N.; and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *JELIA*, 200–212.

Fages, F. 1994. Consistency of Clark's completion and existence of stable models. *MLCS* (1):51–60.

Ferraris, P. 2005. Answer sets for propositional theories. In *LPNMR*, 119–131.

Gelder, A. V. 1992. The well-founded semantics of aggregation. In *PODS*, 127–138. ACM Press.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation—A-Prolog perspective. *AIJ* 138(1-2).

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP*, 1070–1080.

Gelfond, M. 2002. Representing Knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond*. Springer Verlag. 413–451.

Hitzler, P., and Wendt, M. 2005. A uniform approach to logic programming semantics. *TPLP* 5(1-2):123–159.

Kemp, D. B., and Stuckey, P. J. 1991. Semantics of logic programs with aggregates. In *ISLP*, 387–401.

Liu, L., and Truszczyński, M. 2005. Properties of programs with monotone and convex constraints. In *AAAI*, 701–706.

Marek, V. W., and Truszczyński, M. 2004. Logic programs with abstract constraint atoms. In *AAAI*.

Mumick, I. S.; Pirahesh, H.; and Ramakrishnan, R. 1990. The magic of duplicates and aggregates. VLDB, 264–277.

Pelov, N., and Truszczyński, M. 2004. Semantics of disjunctive programs with monotone aggregates — an operator-based approach. In *NMR*, 327–334.

Pelov, N. 2004. *Semantic of Logic Programs with Aggregates*. Ph.D. Dissertation, Katholieke Universiteit Leuven.

Simons, P.; Niemelä, N.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *AIJ* 138:181–234.

Son, T., and Pontelli, E. 2005. A Constructive Semantic Characterization of Aggregates in Answer Set Programming. *CoRR*. cs.AI/0601051.