A Default Approach to Semantics of Logic Programs with Constraint Atoms

Yi-Dong Shen[1] and Jia-Huai You[2]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
ydshen@ios.ac.cn
[2] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2E8
you@cs.ualberta.ca

Abstract. We define the semantics of logic programs with (abstract) constraint atoms in a way closely tied to default logic. Like default logic, formulas in rules are evaluated using the classical entailment relation, so a constraint atom can be represented by an equivalent propositional formula. Therefore, answer sets are defined in a way closely related to default extensions. The semantics defined this way enjoys two properties generally considered desirable for answer set programming – *minimality* and *derivability.* The derivability property is very important because it guarantees free of self-supporting loops in answer sets. We show that when restricted to basic logic programs, this semantics agrees with the conditional-satisfaction based semantics. Furthermore, answer sets by the minimal-model based semantics can be recast in our approach. Consequently, the default approach gives a unifying account of the major existing semantics for logic programs with constraint atoms. This also makes it possible to characterize, in terms of the minimality and derivability properties, the precise relationship between them and contrast with others.

1 Introduction

In knowledge representation and reasoning, it is often desirable to embed into a general inference process/structure special methods for solving/querying a predefined relation. The goal is to make representation easier and reasoning more effective. The technique has been explored, under the name of *global constraints*, in constraint programming [1], and *aggregates* in logic programming [2].

More recently, a great deal of attention has been paid to incorporating constraint atoms (*c-atoms* for short, sometimes under the name of *aggregates*) into answer set programming (ASP) [3,4,5,6,7,8,9,10,11,12,13,14,15]. The intensive study on the subject is largely due to the unsettling question on the semantics for these programs. All of the major semantics proposed so far agree on logic programs with monotone c-atoms; when arbitrary c-atoms are allowed, disagreements exist. Among the recent proposals beyond monotone constraints, two approaches have attracted the most attention – the minimal-model based semantics [4] (with its extension [6]) and the conditional-satisfaction based semantics [15] (with the equivalent one [12]). Although example programs have been used to show their differences, since the ways in which they are defined are quite different, to the best of our knowledge, no precise relationship has been established.

In this paper, we propose a different approach to the semantics of logic programs with c-atoms, with a close tie to default logic [16]. We consider programs consisting of rules of the form $H \leftarrow G$, where H is a propositional atom and G an arbitrary formula built from atoms, c-atoms and standard connectives. Like default logic, formulas in rules are evaluated using the classical entailment relation, so a constraint atom can be represented by an equivalent propositional formula. Therefore, answer sets are defined in a way closely related to default extensions. The semantics defined this way enjoys two properties generally considered desirable for answer set programming − *minimality* and *derivability.* The derivability property is very important because it guarantees free of self-supporting loops in answer sets. Several major existing semantics, such as [4,6,9], lack this property, thus their answer sets may incur self-supporting loops.

We show that for basic logic programs, this semantics agrees with the conditional-satisfaction based semantics. This reveals that the latter can be viewed as a form of default reasoning without resorting to conditional satisfaction. Furthermore, answer sets by the minimal-model based semantics can be recast in our approach. This results in a generalization of the existing semantics to arbitrary propositional formulas. This generalization differs from the one to nested expressions [6] in their different underlying logics and the ways in which c-atoms are encoded by formulas. Our approach is like default logic where formulas are evaluated as in classical logic, while answer sets of nested expressions are minimal total models in the logic here-and-there [17].

Our approach provides a unifying framework for evaluating/comparing the major existing semantics. In particular, we can characterize their differences in terms of the minimality and derivability properties. We show that an answer set by the conditional-satisfaction based semantics is an answer set by the minimal-model based semantics, with the additional condition that it is *derivable* in the sense of default logic.

2 Preliminaries

We restrict attention to a propositional language, as we consider only Herbrand interpretations so that the first-order case reduces to a propositional one via grounding. We assume a propositional language determined by a fixed countable set Σ of propositional *atoms.*

Any subset I of Σ is an *interpretation.* We call $I^+ = I$ the *positive literals* of I, and $I^- = \{\neg a \mid a \in \Sigma \setminus I\}$ the *negative literals.*

A *c-atom* A is a pair (D, C), where D is a finite set of atoms in Σ and C is a collection of sets of atoms in D, i.e., $C \subseteq 2^D$ with 2^D being the powerset of D. For convenience, we use A_d and A_c to refer to the components D and C

of A, respectively. As a general abstract framework, a c-atom A can be used to represent any constraint with a finite set A_c of admissible solutions over a finite domain A_d, including various aggregates [10,11]. Therefore, in the sequel we use the aggregate notation and c-atoms exchangeably to express abstract constraints.

The *complement* of a c-atom A is the c-atom A' with $A'_d = A_d$ and $A'_c = 2^{A_d} \setminus A_c$.

Formulas are built from atoms and c-atoms using connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), and $\supset$ (implication), as in propositional logic. A *literal* is an atom/c-atom or a negated atom/c-atom. A *theory* is a set of formulas. When a theory mentions no c-atoms, it is called an *ordinary theory.* An ordinary theory may be simply called a theory if it is clear from the context.

A (deductive) *rule* r is of the form $H \leftarrow B$, where H is an atom, B is a formula, and $\leftarrow$ is an *if-then* operator. The meaning of r is that "if the logic property B holds then H holds." We use $head(r)$ and $body(r)$ to refer to the head H and body B of r, respectively.

A *general logic program* (or *program*) P is a set of rules. P is called a *basic program* if the body of each rule is a conjunction of literals. A *normal program* is a basic program without c-atoms. A *positive program* is a normal program without negative literals.

Let $I \subseteq \Sigma$ be an interpretation. I *satisfies* an atom A if $A \in I$; $\neg A$ if $A \notin I$. I satisfies a c-atom A if $A_d \cap I \in A_c$; $\neg A$ if $A_d \cap I \notin A_c$. Therefore, it follows that I satisfies $\neg A$ if and only if I satisfies the complement of A.

The satisfaction of a formula F by an interpretation I is defined as in propositional logic. I satisfies a rule r if it satisfies $head(r)$ or it does not satisfy $body(r)$. I is a *model* of a theory W if it satisfies all formulas of W. I is a *model* of a program P if it satisfies all rules of P. A *minimal* model is a model none of whose proper subset is also a model. A model I of P is *supported* if for each $a \in I$, there is a rule in P of the form $a \leftarrow body(r)$ such that $body(r)$ is satisfied by I. For any expression E, we say E is *true* (resp. *false*) in I if and only if I satisfies (resp. does not satisfy) E.

For an (ordinary) theory W, we say that W is *consistent* (or *satisfiable*) if W has a model. We use $Cn(W)$ to denote the *deductive closure* of W as in propositional logic.

The *entailment relation* $\models$ is defined as a minor extension to the one in propositional logic. For any two formulas F and G (possibly containing c-atoms), F *entails* G, denoted $F \models G$, if G is true in all models of F. We write $F \equiv G$ if $F \models G$ and $G \models F$, in which case F and G are said to be *logically equivalent.* Note that when F and G involve no c-atoms, the entailment relation is exactly the classical one. The extension makes it convenient to talk about formulas involving c-atoms being logically equivalent.

For a set S of atoms, when S appears in a formula, it expresses a conjunction $\bigwedge_{a \in S} a$. Similarly, $\neg S$ expresses a conjunction $\bigwedge_{a \in S} \neg a$.

In this paper, the *standard ASP semantics* refers to the stable model semantics defined in [18].

3 Answer Sets for Programs without C-Atoms

We start by recalling the *rationality principle* − the spirit of the standard ASP semantics for normal programs, which says that *one shall not believe anything one is not forced to believe* [19,18]. The rationality principle instructs us that in the construction of an answer set I from a normal program P, we should make as many negative beliefs as possible, provided that no contradiction is derived. On the one hand, this means that the answer set I shall be a *minimal* model of P. On the other hand, this means that given the set $I^- = \{\neg a_1, ..., \neg a_m\}$ of negative beliefs, all positive beliefs I^+ in I shall be *derivable* by applying the deductive rules $H \leftarrow B$ in P in the way that if B holds, then derive H.

We apply the rationality principle to general logic programs. The derivability property is formally defined by means of deductive sets.

Definition 1. Let P be a program without c-atoms and W an (ordinary) theory. The *deductive set* $Th(P, W)$ of P and W is the smallest set of formulas satisfying the following two conditions: (1) $W \subseteq Th(P, W)$; (2) for each rule r in P, if $Th(P, W) \models body(r)$ then $head(r) \in Th(P, W)$.

We can alternatively use a fixpoint approach to define the deductive set. We first introduce the following one-step provability operator:

$$T_P(W) = \{H \mid P \text{ has a rule } H \leftarrow B \text{ such that } W \models B\}$$

Since the entailment relation $\models$ is defined as in propositional logic, $W \models B$ implies $V \models B$ if $W \subseteq V$. Then, T_P is monotone, i.e., for any theories W_1, W_2 such that $W_1 \subseteq W_2$, $T_P(W_1) \subseteq T_P(W_2)$. Therefore, for any (ordinary) theory W, the sequence $T_P^i(W)$, where $T_P^0(W) = W$ and $T_P^{i+1}(W) = W \cup T_P(T_P^i(W))$, is monotonically increasing and has a fixpoint $T_P^\delta(W)$ (i.e., there exists the first ordinal δ such that $T_P^\delta(W) = T_P^{\delta+1}(W)$). We then have this fixpoint theory $T_P^\delta(W)$ as the deductive set of P and W, i.e. $Th(P, W) = T_P^\delta(W)$.[1]

The above fixpoint definition shows that all positive literals in a deductive set are derived stratum by stratum via the fixpoint sequence. This means that the derivability property guarantees free of self-supporting loops.

Our definition of answer sets for general logic programs adheres to the rationality principle and builds directly on the derivability property.

Definition 2. Let P be a program without c-atoms and I an interpretation. I is an *answer set* of P if (1) $Th(P, I^-)$ is consistent; and (2) for each $a \in I$, $Th(P, I^-) \models a$.

Since the head of each rule is an atom, $Th(P, I^-)$ consists of I^- plus a set of atoms. In particular, we have

Theorem 1. *Let P be a program without c-atoms and I an interpretation. I is an answer set of P if and only if $Th(P, I^-) = I \cup I^-$.*

[1] When P is a normal program and W is a set of negative literals, $T_P^\delta(W)$ coincides with the fixpoint $T_{P'}^\delta(\emptyset)$ introduced in [20], where $P' = P \cup W$.

Example 1. Consider the following program:

$$\begin{array}{ll} P_1: & p(a) \leftarrow (p(a) \wedge \neg p(b)) \vee (p(b) \wedge \neg p(a)) \vee (p(a) \wedge p(b)). \\ & p(b) \leftarrow \neg q. \qquad\qquad q \leftarrow \neg p(b). \end{array}$$

Let $\Sigma = \{p(a), p(b), q\}$ and $I = \{p(a), p(b)\}$. Then $I^- = \{\neg q\}$. $T^0_{P_1}(I^-) = I^- = \{\neg q\}$, $T^1_{P_1}(I^-) = I^- \cup T_{P_1}(T^0_{P_1}(I^-)) = \{\neg q, p(b)\}$, $T^2_{P_1}(I^-) = I^- \cup T_{P_1}(T^1_{P_1}(I^-)) = \{\neg q, p(b), p(a)\}$, and $T^3_{P_1}(I^-) = T^2_{P_1}(I^-)$. Note that the body of the second rule is entailed by $\{\neg q\}$, and the body of the first rule is entailed by $\{\neg q, p(b)\}$. Therefore, $Th(P_1, I^-) = T^2_{P_1}(I^-) = \{\neg q, p(b), p(a)\}$. Since $Th(P_1, I^-) = I \cup I^-$, $I = \{p(a), p(b)\}$ is an answer set of P_1.

It is easy to check $I = \{q\}$ is also an answer set of P_1. No other interpretations are answer sets of P_1. For instance, for $I = \{p(b)\}$ we have the deductive set $Th(P_1, I^-) = \{\neg q, \neg p(a), p(b), p(a)\}$, which is inconsistent. □

Example 2. Consider another program:

$$\begin{array}{ll} P_2: & c \leftarrow \neg((a \wedge \neg b) \vee (b \wedge \neg a)). \\ & a \leftarrow c. \qquad\qquad b \leftarrow a. \end{array}$$

Let $\Sigma = \{a, b, c\}$ and $I = \{a, b, c\}$. Then $I^- = \emptyset$. $T^0_{P_2}(I^-) = \emptyset$ and $T^1_{P_2}(I^-) = T^0_{P_2}(I^-)$, where no rule body is entailed by $\emptyset$. $Th(P_2, I^-) = \emptyset$ and thus I is not an answer set of P_2. No other interpretations are answer sets of P_2. Therefore, P_2 has no answer set. □

The derivability property of the answer sets as defined by Definition 2 immediately follows from the definition. It is also easy to show that such answer sets satisfy the minimality property.

Theorem 2. *Answer sets by Definition 2 are supported, minimal models.*

A supported, minimal model is not necessarily an answer set. For example, let P consist of two rules: $d \leftarrow \neg a$ and $a \leftarrow a$. $I = \{a\}$ is a supported, minimal model of P, but it is not an answer set.

For normal programs, answer sets defined above coincide with answer sets under the standard ASP semantics.

Theorem 3. *For a normal program P, an interpretation I is an answer set by Definition 2 if and only if I is an answer set under the standard ASP semantics.*

Our definition of answer sets is closely tied to Reiter's default logic [16]. Recall that a *default theory* is a pair $\triangle = (D, W)$, where W is a theory, D is a set of *defaults* of the form $\frac{\alpha\,:\,\beta_1,...,\beta_m}{\gamma}$, and $\alpha, \beta_1, ..., \beta_m, \gamma$ are formulas. For a theory E, let $\Gamma_\triangle(E)$ be the smallest deductively closed set of formulas satisfying the following two conditions: (1) $W \subseteq \Gamma_\triangle(E)$; (2) for each default r in D, if $\Gamma_\triangle(E) \models \alpha$ and $\neg\beta_1, ..., \neg\beta_m \notin E$, then $\gamma \in \Gamma_\triangle(E)$. E is called an *extension* of $\triangle$ if $E = \Gamma_\triangle(E)$.

Theorem 4. *Let P be a program without c-atoms and I an interpretation. Let $D_1 = \{\frac{body(r):}{a} \mid a \leftarrow body(r) \in P\}$, $D_2 = \{\frac{:\neg a}{\neg a} \mid a \in \Sigma\}$, and $\triangle = (D_1 \cup D_2, \emptyset)$. $Cn(Th(P, I^-)) = \Gamma_\triangle(I \cup I^-)$.*

By Theorem 1, I is an answer set of P if and only if $I \cup I^- = Th(P, I^-)$; then by Theorem 4, if and only if $Cn(I \cup I^-) = \Gamma_\triangle(I \cup I^-)$. Therefore, I is an answer set of P if and only if $Cn(I \cup I^-)$ is an extension of $\triangle$.

4 Answer Sets for Programs with C-Atoms

Since our semantics (Definition 2) is defined by deductive sets using the classical entailment relation, it has an advantage that replacing the body of a rule by a logically equivalent formula preserves answer sets. Therefore, to extend the semantics to programs with c-atoms, it suffices to represent each c-atom as an equivalent formula.

Recall that the satisfaction of a c-atom A is defined by means of propositional interpretations [11]: for any interpretation I, I satisfies A if $A_d \cap I \in A_c$, and I satisfies $\neg A$ if $A_d \cap I \notin A_c$. We want to have a suitable formula F with the property that for any interpretation I, I satisfies A if and only if I satisfies F. Such a formula has been introduced in [13], where it was used to justify an abstract representation of c-atoms.

Definition 3 ([13]). Let $A = (A_d, A_c)$ be a c-atom with $A_c = \{S_1, ..., S_m\}$. The *DNF* formula $C_1 \vee ... \vee C_m$ for A is defined as: each C_i is a conjunction $a_1 \wedge ... \wedge a_k \wedge \neg b_1 \wedge ... \wedge \neg b_l$ built from S_i such that $S_i = \{a_1, ..., a_k\}$ and $(A_d \setminus S_i) = \{b_1, ..., b_l\}$.

Proposition 1. *Let A be a c-atom and I an interpretation. Let $C_1 \vee ... \vee C_m$ be the DNF formula for A. I satisfies A if and only if I satisfies $C_1 \vee ... \vee C_m$. I satisfies $\neg A$ if and only if I satisfies $\neg(C_1 \vee ... \vee C_m)$.*

Let *body*(r) be the body of a rule r, and *body*(r') be *body*(r) with all c-atoms replaced by their DNF formulas. By Proposition 1, an interpretation I satisfies *body*(r) if and only if I satisfies *body*(r'). We then have the following ASP semantics for general logic programs.

Definition 4. Let P be a program and I an interpretation. I is an *answer set* of P if I is an *answer set* of P' as defined in Definition 2, where P' is P with all c-atoms replaced by their DNF formulas.

Since answer sets of P' satisfy both the minimality and the derivability property, by Proposition 1 answer sets of P satisfy the two properties as well.

Example 3. Consider the following program (borrowed from [21]):

$$\begin{array}{lll} P_3: & p(a) \leftarrow COUNT(\{X \mid p(X)\}) > 0. & \\ & p(b) \leftarrow \neg q. & q \leftarrow \neg p(b). \end{array}$$

The aggregate notation $COUNT(\{X \mid p(X)\}) > 0$ corresponds to a c-atom A with $A_d = \{p(a), p(b)\}$ and $A_c = \{\{p(a)\}, \{p(b)\}, \{p(a), p(b)\}\}$. The DNF formula for A is $(p(a) \wedge \neg p(b)) \vee (p(b) \wedge \neg p(a)) \vee (p(a) \wedge p(b))$. By replacing A with this DNF formula, we obtain P_3' which is the same as P_1 in Example 1. Since $P_3' = P_1$ contains no c-atom, its answer sets can be computed applying Definition 2. P_1 has two answer sets: $I_1 = \{p(a), p(b)\}$ and $I_2 = \{q\}$. Therefore, by Definition 4 P_3 has the two answer sets. □

Example 4. Consider the following program with a negated constraint:

$$\begin{array}{lll} P_4: & c \leftarrow \neg 1\{a,b\}1. & \\ & a \leftarrow c. & b \leftarrow a. \end{array}$$

The cardinality constraint $1\{a,b\}1$ corresponds to a c-atom A with $A_d = \{a,b\}$ and $A_c = \{\{a\},\{b\}\}$. The DNF formula for A is $(a \wedge \neg b) \vee (b \wedge \neg a)$. By replacing A with this DNF formula, we obtain P'_4 which is the same as P_2 in Example 2. P_2 has no answer set, thus P_4 has no answer set. □

5 Relating to Other Approaches

Our approach provides a unifying framework for evaluating the major existing proposals for arbitrary c-atoms, including the conditional-satisfaction based semantics [15], the minimal-model based semantics [4,5], and the computation-based semantics [9]. In this section, we characterize the differences of these semantics in terms of the minimality and derivability properties for basic programs.

5.1 Conditional-Satisfaction Based Semantics

We show that when restricted to basic programs, Definition 4 defines the same answer sets as the ones by conditional-satisfaction [15].

To introduce the notion of conditional satisfaction, we need a simple program transformation. Any atom a can be expressed as an *elementary* c-atom $A = (\{a\},\{\{a\}\})$; any negated atom $\neg a$ can be expressed as a c-atom $A = (\{a\},\{\emptyset\})$; and any negated c-atom $\neg A$ can be replaced by the complement of A. Due to this, in this section we assume all basic programs have been rewritten so that their rules are of the form $H \leftarrow A_1 \wedge ... \wedge A_m$, where each A_i is a c-atom.

Definition 5. Let R and S be two sets of atoms and A a c-atom. We say R *conditionally satisfies* A w.r.t. S, denoted $R \models_S A$, if R satisfies A and for every S' such that $R \cap A_d \subseteq S'$ and $S' \subseteq S \cap A_d$, we have $S' \in A_c$. For a rule r in a basic program, $R \models_S body(r)$ if $R \models_S A_i$ for each A_i in $body(r)$.

An immediate consequence operator $\mathcal{T}_P(R,S)$ is defined in terms of the conditional satisfaction.

Definition 6. Let P be a basic program and R and S be two sets of atoms. Define

$$\mathcal{T}_P(R,S) = \{a \mid a \leftarrow body(r) \in P \text{ and } R \models_S body(r)\}$$

It is proved that when the second argument is a model of P, $\mathcal{T}_P$ is monotone w.r.t. its first argument [15]. Therefore, for any model M the monotone sequence $\mathcal{T}_P^i(\emptyset,M)$, where $\mathcal{T}_P^0(\emptyset,M) = \emptyset$ and $\mathcal{T}_P^{i+1}(\emptyset,M) = \mathcal{T}_P(\mathcal{T}_P^i(\emptyset,M),M)$, converges to a fixpoint $\mathcal{T}_P^\alpha(\emptyset,M)$.

The *conditional-satisfaction based ASP semantics* says that a model M is an answer set of P if M coincides with the fixpoint $\mathcal{T}_P^\alpha(\emptyset,M)$ [15]. Denecker et al. [22] and Shen and You [13] define the same semantics using different definitions.

Example 5. Assume P_4 of Example 4 has been rewritten to

$$\begin{aligned} P_4: \quad & c \leftarrow (\{a,b\},\{\emptyset,\{a,b\}\}). \\ & a \leftarrow (\{c\},\{\{c\}\}). \\ & b \leftarrow (\{a\},\{\{a\}\}). \end{aligned}$$

Note that the c-atom in the body of the first rule is the complement of the original c-atom. This program has no answer set under the conditional-satisfaction based semantics. Take $M = \{a,b,c\}$ as an example, which is a model of P_4. Let $\mathcal{T}^0_{P_4}(\emptyset, M) = \emptyset$. Although the c-atom in the body of the first rule is satisfied by $\emptyset$, it is not conditionally satisfied by $\emptyset$ w.r.t. M. As a result, we reach the fixpoint $\mathcal{T}^1_{P_4}(\emptyset, M) = \mathcal{T}_{P_4}(\mathcal{T}^0_{P_4}(\emptyset, M), M) = \emptyset$. M does not agree with the fixpoint $\emptyset$, thus is not an answer set of P_4. □

The conditional-satisfaction based fixpoint and the deductive set are closely related.

Theorem 5. *Let P be a basic program, P' be P with all c-atoms replaced by their DNF formulas, and M be a model of P. We have $\mathcal{T}^\alpha_P(\emptyset, M) = Th(P', M^-) - M^-$.*

The following corollary follows immediately, showing that the ASP semantics of Definition 4 and the conditional-satisfaction based semantics agree on basic programs.

Corollary 1. *Let P be a basic program and M a model of P. M is an answer set of P under the conditional-satisfaction based semantics if and only if M is an answer set defined by Definition 4.*

Since answer sets by Definition 4 respect the rationality principle, the conditional-satisfaction based semantics also satisfies the two properties for basic programs. Similar results have been established earlier [15], which shows that any answer set under the conditional-satisfaction based semantics is a minimal model for which a *level mapping* exists. The definition of a level mapping relies on identifying structural properties of a c-atom w.r.t. a given interpretation, in order to capture conditional satisfaction. Thus, the derivability property can be seen as an independent way to capture level mapping, without conditional satisfaction.

However, conditional satisfaction is not defined over disjunction of c-atoms. In fact, a direct application could cause unintuitive behaviors.

Example 6 (borrowed from [23]). Let A be the aggregate $SUM(\{X \mid p(X)\}) \neq -1$. In standard mathematics, this is equivalent to $SUM(\{X \mid p(X)\}) > -1$ or $SUM(\{X \mid p(X)\}) < -1$, because $A \equiv B \vee C$, where B and C denote the latter two aggregates, respectively.

Now suppose $S = \{p(1)\}$ and $M = \{p(1), p(2), p(-3)\}$. It can be verified that while $S \models_M A$, it is not the case that $S \models_M B$ or $S \models_M C$. □

In our approach, c-atoms are represented by propositional formulas, which are then evaluated as in propositional logic. As a result, logic equivalence guarantees the preservation of answer sets. In fact, it is not difficult to verify that, for any program P, replacing a formula in the body of a rule in P by a logically equivalent one results in a program strongly equivalent to P. This is an advantage of our approach, inherited from that of default logic.

5.2 Minimal-Model Based Semantics

Let P be a basic program and I an interpretation. To check if I is an answer set, this semantics first removes all rules whose body contains a literal that is not satisfied by I, then defines I to be an answer set if I is a minimal model of the simplified program P^I [4,5]. Ferraris [6] defines an ASP semantics in a different way, which agrees with the minimal-model based one on basic programs.

We now recast the minimal-model based semantics in our framework for general logic programs, where c-atoms are represented by their DNF formulas. We define the *reduct* P^I of P w.r.t. I as the result of removing all rules r from P such that $body(r)$ is not satisfied by I.

Definition 7. Let P be a general logic program and P' be P with all c-atoms replaced by their DNF formulas. An interpretation I is said to be a *weak answer set* of P if I is a minimal model of P'^I.

Theorem 6. *Weak answer sets of a program P are minimal models of P.*

For basic programs, by Proposition 1 weak answer sets coincide with answer sets under the minimal-model based semantics. Furthermore, we have

Theorem 7. *Let P be a general logic program and P' be P with all c-atoms replaced by their DNF formulas. An interpretation I is an answer set of P by Definition 4 if and only if I is a weak answer set of P such that for any $a \in I$, $Th(P', I^-) \models a$.*

It follows immediately that for basic programs, answer sets under the conditional-satisfaction based semantics are such answer sets under the minimal-model based semantics that are derivable via the deductive set.

The minimal-model based semantics does not satisfy the derivability property, even for basic programs.

Example 7. Consider the following basic program:

$$\begin{array}{ll} P_5: & p(1). \\ & p(2) \leftarrow p(-1). \\ & p(-1) \leftarrow SUM(\{X \mid p(X)\}) \geq 1. \end{array}$$

The aggregate $SUM(\{X \mid p(X)\}) \geq 1$ corresponds to a c-atom A, where $A_d = \{p(-1), p(1), p(2)\}$ and $A_c = \{\{p(1)\}, \{p(2)\}, \{p(-1), p(2)\}, \{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\}$. The DNF formula for A is

$$\begin{array}{l} (p(1) \wedge \neg p(-1) \wedge \neg p(2)) \vee (\neg p(1) \wedge \neg p(-1) \wedge p(2)) \vee (\neg p(1) \wedge p(-1) \wedge p(2)) \\ \vee (p(1) \wedge \neg p(-1) \wedge p(2)) \vee (p(1) \wedge p(-1) \wedge p(2)) \end{array}$$

which as in propositional logic, can be simplified to $(p(1) \wedge \neg p(-1)) \vee p(2)$.

Let $\Sigma = \{p(1), p(-1), p(2)\}$. P_5 has only one model, $\{p(1), p(-1), p(2)\}$. By Definition 4 or equivalently under the conditional-satisfaction based semantics, P_5 has no answer set, since the model is not derivable via the deductive set. However, $I = \{p(1), p(-1), p(2)\}$ is an answer set of P_5 under the minimal-model based semantics. □

It should be pointed out that due to lack of derivability, the minimal-model based semantics ([4] and its extension [6]) may incur undesirable self-supporting loops in its answer sets. For instance, in the above example $I = \{p(1), p(-1), p(2)\}$ is the answer set of P_5, where $p(2)$ and $p(-1)$ can only be deduced via one of the following self-supporting loops:

$$p(2) \rightarrow SUM(\{X \mid p(X)\}) \geq 1 \rightarrow p(-1) \rightarrow p(2).$$
$$p(-1) \rightarrow p(2) \rightarrow SUM(\{X \mid p(X)\}) \geq 1 \rightarrow p(-1).$$

That is, in order to derive $p(2)$ and $p(-1)$ from P_5 we must assume either $p(2)$ or $p(-1)$ is true in advance.

5.3 Computation-Based Semantics

Liu et al. [9] define an answer set I for a basic program P to be the fixpoint of a *computation* $\langle I_i\rangle_{i=0}^{\infty}$ with $I_0 = \emptyset$ and $I_{i+1} = T_{Q_i}^{nd}(I_i)$, where Q_i is a subset of the rules in P whose bodies are satisfied by I_i and $T_{Q_i}^{nd}(I_i)$ consists of all heads of rules in Q_i, such that for each $i \geq 0$, $I_i \subseteq I_{i+1}$ and $Q_i \subseteq Q_{i+1}$.

Consider the following basic program:

$$\begin{array}{ll} P_6: & p(1). \\ & p(-1) \leftarrow p(2). \\ & p(2) \leftarrow SUM(\{X \mid p(X)\}) \geq 1. \end{array}$$

As described in Example 7, the DNF formula for $SUM(\{X \mid p(X)\}) \geq 1$ is $(p(1) \wedge \neg p(-1)) \vee p(2)$. P_6 has two models, $\{p(1), p(-1)\}$ and $\{p(1), p(-1), p(2)\}$.

Let r_i refer to the i-th rule. We have a computation for P_6, where $I_0 = \emptyset$, $I_1 = \{p(1)\}$ with $Q_0 = \{r_1\}$, $I_2 = \{p(1), p(2)\}$ with $Q_1 = \{r_1, r_3\}$, and $I_3 = \{p(1), p(-1), p(2)\}$ with $Q_2 = \{r_1, r_2, r_3\}$. I_3 is the fixpoint of the computation, where $I_0 \subseteq ... \subseteq I_3$ and $Q_0 \subseteq Q_1 \subseteq Q_2$. Thus I_3 is an answer set of P_6 under the computation-based semantics.

Observe that I_3 is not a minimal model of P_6; neither is it derivable via the deductive set $Th(P_6, I_3^-)$. This shows that the computation-based semantics satisfies neither the minimality nor the derivability property.

Just like the minimal-model based semantics, due to lack of derivability the computation-based semantics may incur undesirable self-supporting loops in its answer sets. For instance, $I_3 = \{p(1), p(-1), p(2)\}$ is an answer set of P_6, where $p(2)$ can only be deduced via a self-supporting loop

$$p(2) \rightarrow SUM(\{X \mid p(X)\}) \geq 1 \rightarrow p(2).$$

$\{p(1), p(-1), p(2)\}$ is an answer set of P_6 under the computation-based semantics, but it is not under the minimal-model based one. On the contrary, $\{p(1), p(-1), p(2)\}$ is an answer set of P_5 under the minimal-model based semantics, but it is not under the computation-based one. This means that answer sets under the minimal-model based semantics are not necessarily answer sets under the computation-based semantics, and vice versa.

To get more restricted computations (answer sets), Liu et al. [9] propose to use a *sub-satisfiability* relation $\triangleright$ to replace the standard satisfiability with the property that for any interpretation I and c-atom A, $I \triangleright A$ implies I satisfies A. Then, different computations and answer sets can be obtained by embedding different sub-satisfiability relations into the definition of computation.

6 Summary and Discussion

Incorporating constraints into a general knowledge representation and reasoning (KR) system has proven to be a crucial step in gaining representation power and reasoning efficiency. When logic programs with constraint atoms are taken as the underlying KR language, the question on the semantics has raised great interest, with competing views and definitions of answer sets.

For normal logic programs, the stable model semantics has roots in different nonmonotonic formalisms, thus, as summarized by Lifschitz [24], leading to current twelve different definitions. As pointed out by Lifschitz, "there are reasons why each of them is valuable and interesting. A new characterization of stable models can suggest an alternative picture of the intuitive meaning of logic programs; $\cdots$ or it can be interesting simply because it demonstrates a relationship between seemingly unrelated ideas."

In this paper, we developed an alternative new approach to the semantics of logic programs with c-atoms, where formulas in rules are evaluated using the classical entailment relation and c-atoms are represented by equivalent propositional formulas. The resulting framework can be seen as one with c-atoms embedded into a fragment of default logic. It turns out that both the conditional-satisfaction based semantics and the minimal-model based semantics have a root in the default framework. The former can be viewed as a form of default reasoning without conditional satisfaction, while under exactly the same representation of c-atoms, the latter can be recast in the same framework. As a result, we are able to identify the precise relationship between the two major existing semantics and contrast with others such as the computation-based semantics.

The semantics defined by the default approach has two important properties $-$ minimality and derivability. The derivability property is very useful; it guarantees free of self-supporting loops in answer sets. Our examples show that several major existing semantics, such as [4,6,9], lack this property, thus their answer sets may incur self-supporting loops.

Another advantage of the default approach is that replacing a constraint by a logically equivalent one preserves strong equivalence. This is an important feature, since constraint atoms are supposed to represent pre-defined/built-in/global constraints in constraint solving, and special constraint propagation rules for such a built-in constraint may need to be implemented or updated. In this case, one only needs to verify the preservation of satisfaction. This feature also provides a methodology of representing a constraint by some (logically equivalent) combination of constraints. For example, an aggregate $SUM(..) \neq k$ can be substituted by $SUM(..) > k \vee SUM(..) < k$, according to the standard mathematics while preserving strong equivalence. Since constraints are viewed as propositional formulas, this feature is applicable in a more general context: Given a program P, replacing a formula (possibly including constraint atoms) of the body of a rule in P by a logically equivalent one results in a program P' which is strongly equivalent to P.

As remarked earlier, our approach is different from that of [6], due to the different underlying logics and methods of representing c-atoms. In [6] rule bodies and c-atoms are nested expressions in the logic of here-and-there, while in our approach they are classical propositional formulas. We would like to argue that an intimate integration of classical formulas into logic programs, as done in our approach which is inherited from default logic, facilitates knowledge representation. The same point has been argued for a more general context by [25] and recent work on integrating ASP with ontologies and description logics for the Semantic Web (where classical formulas in rule bodies are interpreted as queries to a description logic knowledge base [26]).

Our language can be extended to accommodate c-atoms in rule heads like $A \leftarrow G$, where A is a c-atom and G an arbitrary formula. It can also be extended to logic programs whose rule heads may be a disjunction of atoms. As a concrete application, we are applying the default approach to characterizing the semantics of *description logic programs* for the Semantic Web [26].

Acknowledgments

We would like to thank the anonymous referees for their constructive comments. Yi-Dong Shen is supported in part by NSFC grants 60673103, 60721061 and 60833001, and by the National High-tech R&D Program (863 Program). The work by Jia-Huai You is supported in part by the Natural Sciences and Engineering Research Council of Canada.

References

1. Marriott, K., Stuckey, P.: Programming with Constraints. MIT Press, Cambridge (1998)
2. Kemp, D., Stuckey, P.: Semantics of logic programs with aggregates. In: Proc. Int'l Symposium on Logic Programming, pp. 387–401. MIT Press, Cambridge (1991)
3. Calimeri, F., Faber, W., Leone, N., Perri, S.: Declarative and computational properties of logic programs with aggregates. In: IJCAI 2005, pp. 406–411 (2005)
4. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
5. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and implementation of aggregate functions in the dlv system. TPLP 8(5) (2008)
6. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 119–131. Springer, Heidelberg (2005)
7. Lee, J., Lifschitz, V., Palla, R.: A reductive semantics for counting and choice in answer set programming. In: AAAI 2008, pp. 472–479 (2008)

8. Liu, L., Truszczynski, M.: Properties and applications of programs with monotone and convex constraints. JAIR 7, 299–334 (2006)
9. Liu, L., Pontelli, E., Son, T., Truszczynski, M.: Logic programs with abstract constraint atoms: the role of computations. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 286–301. Springer, Heidelberg (2007)
10. Marek, V.W., Remmel, J.B.: Set constraints in logic programming. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 167–179. Springer, Heidelberg (2004)
11. Marek, V.W., Truszczynski, M.: Logic programs with abstract constraint atoms. In: AAAI 2004, pp. 86–91 (2004)
12. Pelov, W., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. TPLP 7(3), 301–353 (2007)
13. Shen, Y.D., You, J.H.: A generalized Gelfond-Lifschitz transformation for logic programs with abstract constraints. In: AAAI 2007, pp. 483–488 (2007)
14. Simons, P., Niemela, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138(1-2), 181–234 (2002)
15. Son, T.C., Pontelli, E., Tu, P.H.: Answer sets for logic programs with arbitrary abstract constraint atoms. JAIR 29, 353–389 (2007)
16. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
17. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2(4), 526–541 (2001)
18. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP 1988, pp. 1070–1080 (1988)
19. Gelfond, M.: Answer sets. Handbook of Knowledge Representation. Elsevier, Amsterdam (2008)
20. van Gelder, A.: The alternating fixpoint of logic programs with negation. In: PODS, pp. 1–10 (1989)
21. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. TPLP 7(3), 355–375 (2007)
22. Denecker, M., Pelov, N., Bruynooghe, M.: Ultimate well-founded and stable semantics for logic programs with aggregates. In: Codognet, P. (ed.) ICLP 2001. LNCS, vol. 2237, pp. 212–226. Springer, Heidelberg (2001)
23. Liu, G.H., You, J.H.: Lparse programs revisited: Semantics and representation of aggregates. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 347–361. Springer, Heidelberg (2008)
24. Lifschitz, V.: Twelve definitions of a stable model. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 37–51. Springer, Heidelberg (2008)
25. Denecker, M., Vennekens, J.: Building a knowledge base sysem for an integration of logic programmng and classical logic. In: ICLP 2009, pp. 71–76 (2009)
26. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. 172(12-13), 1495–1539 (2008)