Computing Loops with at Most One External Support Rule for Basic Logic Programs with Arbitrary Constraint Atoms

Jianmin Ji
University of Science and Technology of China, Hefei, China
jianmin@ustc.edu.cn

Fangzhen Lin
Hong Kong University of Science and Technology, Hong Kong
flin@cs.ust.hk

Jia-Huai You
University of Alberta, Edmonton, Canada
you@cs.ualberta.ca

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

The well-founded semantics of logic programs is not only an important semantics but also serves as an essential tool for program simplification in answer set computations. Recently, it has been shown that for normal and disjunctive programs, the well-founded models can be computed by unit propagation on program completion and loop formulas of loops with no external support. An attractive feature of this approach is that when loop formulas of loops with exactly one external support are added, consequences beyond the well-founded model can be computed, which sometimes can significantly speed up answer set computation. In this paper, we extend this approach to basic logic programs with abstract constraint atoms. We define program completion and loop formulas and show how to capture the well-founded semantics that approximate answer sets of basic logic programs. We show that by adding the loop formulas of loops with one external support, consequences beyond well-founded models can be computed. Our experiments show that for certain logic programs with constraints accepted by lparse, the consequences computed by our algorithms can speed up current ASP solvers smodels and clasp.

KEYWORDS: logic programs with abstract constraint atoms, loop formulas, well-founded semantics

1 Introduction

The well-founded semantics (Van Gelder et al. 1991) and answer set semantics (Gelfond and Lifschitz 1988) are two most prominent semantics in logic programming. To date, the well-founded semantics has been extended to disjunctive logic programs (Wang and Zhou 2005), logic programs with aggregates (or, aggregate programs) (Pelov et al. 2007), dl-programs (Eiter et al. 2011; You et al. 2012), and basic logic programs with abstract constraint atoms (c-atoms) (Wang et al. 2012). In the literature, most definitions of the well-founded semantics are in terms of unfounded sets or as the least fixpoint of a 3-valued immediate consequence operator. In recent papers, (Chen et al. 2008; Chen et al. 2009; Chen et al. 2013) showed that the well-founded model

of a normal logic program or a disjunctive logic program according to (Wang and Zhou 2005) can be computed from its completion and loop formulas of loops with no external support rules by unit propagation. In this paper, our focus is on basic logic programs with c-atoms.

In (Son et al. 2007), two different types of answer sets for logic programs with c-atoms are proposed, called *answer set by reduct* (or *r-answer set*) and *answer set by complement* (or *c-answer set*) (Son et al. 2007). These semantics differ from each other in the treatment of negation-as-failure literals. In the literature, different well-founded semantics have been formulated that approximate r-answer sets (Wang et al. 2012) and c-answer sets (Pelov et al. 2007), respectively. In this paper, we show that the well-founded semantics proposed in (Wang et al. 2012) can be computed by unit propagation on program completion and loop formulas of loops with no external support rules. Since every c-answer set is also an r-answer set, this result can be applied to the computation of c-answer sets. In fact, we show that our definitions naturally lead to a well-founded semantics that approximates c-answer sets directly.

In (Chen et al. 2013), the authors also showed that, for normal and disjunctive logic programs, loop formulas of loops with only one external support rule are equivalent to sets of binary clauses, thus can be used effectively by unit propagation. However, this is not the case for basic logic programs with c-atoms. In this paper, we identify a certain kind of loops with only one external support rule whose loop formulas are equivalent to sets of binary clauses. Again, we show that, using the loop formulas of loops with at most one external support rule, along with completion, unit propagation can compute more consequences than well-founded semantics and these extra consequences can help ASP solvers. This is supported experimentally. For certain logic programs, the computed consequences can speed up clasp (Gebser et al. 2007) and smodels (Simons et al. 2002). Thus we believe that this work opens the window for more efficient computations of answer sets.

In the rest of this paper, we first review the results on which our work is based. We then show the main result that the well-founded semantics of (Wang et al. 2012) can be computed by unit propagation on completion and loop formulas of loops with no external support. We identify a certain kind of loops whose loop formulas are equivalent to sets of binary clauses, and formulate an algorithm to compute these loops, which is followed by some experiments. The paper is concluded with final remarks.

2 Preliminaries

We assume an underlying propositional language with a set $\mathscr{A}$ of atoms. A *literal* is either an atom or an expression of the form $\neg a$, where a is an atom. *Lit* is the set of all possible literals in the language. Given a set of literals I, we use I^+ to denote the set of atoms in I, and $I^- = \{a \mid \neg a \in I\}$; I is said to be *consistent* if $I^+ \cap I^- = \emptyset$. A *partial interpretation* I is a consistent set of literals; it is *total* if $I^+ \cup I^- = \mathscr{A}$. Let M be a set of atoms. We denote by $C(M)$ the total interpretation $M \cup \{\neg a \mid a \notin M\}$, and denote by $\neg.M$ the set $\{\neg a \mid a \in M\}$. For a set of literals I, we denote by $\bar{I}$ the set $\{a \mid \neg a \in I\} \cup \{\neg a \mid a \in I\}$. For convenience, here we identify a set I of literals with the propositional formula $\bigwedge_{l \in I} l$. Let I be a set of literals and F a propositional formula. That I *satisfies* F, denoted $I \models F$, is defined as usual. A set of atoms M is a *model* of F if $C(M) \models F$.

An *abstract constraint atom (c-atom)* is an expression of the form (D,C), where D is a finite set of atoms and $C \subseteq 2^D$. In the following, given a c-atom $A = (D,C)$, we use A_d and A_c to refer to its first and second components, respectively. The *complement* of a c-atom A, written $\widehat{A}$, is then

a c-atom such that $\widehat{A}_d = A_d$ and $\widehat{A}_c = 2^{A_d} \setminus A_c$. A c-atom is said to be *elementary* if it is of the form $(\{a\}, \{\{a\}\})$, where a is an atom. In the following, an elementary c-atom of the above form will be identified with ordinary atom a.

A *logic program with c-atoms* (*logic program* or *program*) is a finite set of *rules* of the form

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

where A and A_i's are c-atoms. Given a rule r of the form (1), we define $head(r) = A$, $body(r) = pos(r) \cup not\, neg(r)$ where $pos(r) = \{A_1, \ldots, A_k\}$, $neg(r) = \{A_{k+1}, \ldots, A_n\}$ and $not\, S = \{not\, A \mid A \in S\}$ for a set S of c-atoms. A rule of the form (1) is *basic* (resp. *positive*) if A is elementary (resp. $k = n$). A logic program P is *basic* (resp. *positive*) if every rule in P is basic (resp. positive).

A set of atoms $M \subseteq \mathscr{A}$ satisfies a c-atom A, denoted $M \models A$, if $A_d \cap M \in A_c$. M satisfies $not\, A$, denoted $M \models not\, A$, if $A_d \cap M \notin A_c$. M satisfies the body of a rule r of the form (1), denoted $M \models body(r)$, if $M \models A_i$ $(1 \leq i \leq k)$ and $M \models not\, A_j$ $(k+1 \leq j \leq n)$. M satisfies a rule r if it satisfies $head(r)$ or if it does not satisfy $body(r)$. M is a *model* of a program P if M satisfies every rule of P. A rule is *normal* if every c-atom in it is elementary. A logic program P is *normal* if every rule in it is normal.

Here we briefly review the definitions of answer sets for basic logic programs (Son et al. 2007).

Let M and S be two sets of atoms. The set S *conditionally satisfies* a c-atom A w.r.t. M, denoted $S \models_M A$, iff $S \models A$ and I belongs to A_c for every I with $S \cap A_d \subseteq I$ and $I \subseteq M \cap A_d$.

Let P be a positive basic program and M a model of P. We define the operator $T_{(P,M)}$ as follows:

$$T_{(P,M)}(S) = \{head(r) \mid r \in P \text{ such that } S \models_M body(r)\}.$$

The operator $T_{(P,M)}$ is monotonic in that $S_1 \subseteq S_2 \subseteq M$ implies $T_{(P,M)}(S_1) \subseteq T_{(P,M)}(S_2) \subseteq M$. Thus for any model M of P, the least fixpoint of $T_{(P,M)}$ exists, written $lfp(T_{(P,M)})$, which can be iteratively evaluated as follows: $T^0_{(P,M)} = \emptyset$, and $T^{i+1}_{(P,M)} = T_{(P,M)}(T^i_{(P,M)})$, where $i \geq 0$. A model M of a positive basic program P is an *answer set* of P if M is the least fixpoint of $T_{(P,M)}$, i.e., $M = lfp(T_{(P,M)})$. There are two different answer set semantics for basic logic programs. One is defined by reduct, and the other is by complement. Let P be a basic logic program and $M \subseteq \mathscr{A}$, the *reduct* of P w.r.t. M, written P^M, is the positive program obtained from P by

- eliminating each rule r if $M \models B$ for some $B \in neg(r)$;
- eliminating any negative literal $not\, B$ from the remaining rules.

The *complement* of a basic logic program P, written $\widehat{P}$, is the positive program obtained from P by replacing each $not\, B$ with its complement $\widehat{B}$. It is clear that both P^M and $\widehat{P}$ are positive and basic. For a basic logic program P and a set M of atoms, we say that M is a *c-answer set* (resp. *r-answer set*) of P iff M is an answer set of $\widehat{P}$ (resp. P^M). It is known that every c-answer set of a basic program is an r-answer set, but not vice versa (Son et al. 2007), and both semantics generalize the stable model semantics for normal logic programs.

Unit propagation is an inference rule on clauses. It can be defined through a procedure that simplifies a set of clauses. The procedure is based on unit clauses, i.e. clauses that are composed of a single literal. If a set of clauses contains the unit clause l, the other clauses are simplified by the application of the two following rules:

1. every clause (other than the unit clause itself) containing l is removed;
2. in every clause that contains l this literal is deleted.

The procedure continues until neither of these two rules can be applied. It leads to a new set of

clauses that is equivalent to the old one. Let Γ be a set of clauses, below, we use $UP(\Gamma)$ to denote the set of literals (unit clauses) contained in the result of applying the procedure on Γ.

3 Well-Founded Semantics and Loop Formulas

We review the well-founded semantics of basic programs (Wang et al. 2012), followed by definitions of completion and loop formulas extended from (Liu and Truszczynski 2006) and (You and Liu 2008). Next, we show that the well-founded semantics in (Wang et al. 2012) can be computed by unit propagation on completion and loop formulas. This result lays the foundation for computing more consequences using loop formulas of loops with at most one external support. We then extend it to define a well-founded semantics that approximate c-answer sets directly.

3.1 Well-Founded Semantics by Reduct and Loop Formulas for Basic Programs

The well-founded semantics for basic logic programs in (Wang et al. 2012) relies on a compact representation of c-atoms proposed in (Shen et al. 2009).

Let S and J be two disjoint sets of atoms, by $S \uplus J$ we denote the set $\{S' \mid S \subseteq S' \text{ and } S' \subseteq S \cup J\}$ called the S-prefixed powerset of J. Let A be a c-atom, S and J two subsets of A_d. The S-prefixed powerset of J is *maximal* in A if $S \uplus J \subseteq A_c$ and there is no other sets S' and J' s.t. $S' \uplus J' \subseteq A_c$ and $S \uplus J \subset S' \uplus J'$. By A_c^* we denote the set of all maximal S-prefixed powerset of J in A for any two sets S and J. From Theorem 3.2 in (Shen et al. 2009), A_c^* is unique. The *abstract representation* of a c-atom A is denoted by $A^* = (A_d, A_c^*)$, which relates to a propositional formula.

Theorem 3.1 (*Theorem 4.2 (Shen et al. 2009)*)
Let A be a c-atom and M be a set of atoms. $M \models A$ if and only if $C(M)$ satisfies

$$\bigvee_{S \uplus J \in A_c^*} S \wedge \neg(A_d \setminus (S \cup J)). \tag{2}$$

Let A be a c-atom and I a partial interpretation. We say that I *satisfies* A, denoted $I \models_{Lit} A$, iff for some $S \uplus J \in A_c^*$, $S \subseteq I^+$ and $A_d \setminus (S \cup J) \subseteq I^-$; I *falsifies* A, denoted $I \Vdash_{Lit} A$, iff $S \cap I^- \neq \emptyset$ or $(A_d \setminus (S \cup J)) \cap I^+ \neq \emptyset$ for any $S \uplus J \in A_c^*$. I satisfies (resp. falsifies) *not* A iff I falsifies (resp. satisfies) A. Given a set L of c-atoms or *not* A where A is a c-atom, I satisfies (resp. falsifies) L, denoted $I \models_{Lit} L$ (resp. $I \Vdash_{Lit} L$), if I satisfies (resp. falsifies) each formula in L.

Let P be a basic program and I a partial interpretation. A set of atoms U is an *unfounded set* of P w.r.t. I iff for any $a \in U$ and $r \in P$ with $head(r) = a$, one of the following conditions is true:

- $I \Vdash_{Lit} not\, neg(r)$;
- there exists $A \in pos(r)$ s.t. for any $S \uplus J \in A_c^*$ either $U \cap S \neq \emptyset$ or $I \models \neg(S \wedge \neg(A_d \setminus (S \cup J))$.

We define the operators T_P, U_P and W_P as follows.

$T_P(I) = \{head(r) \mid r \in P \text{ and } I \models_{Lit} body(r)\}$;
$U_P(I) =$ the greatest unfounded set of P w.r.t. I;
$W_P(I) = T_P(I) \cup \neg.U_P(I)$.

As T_P, U_P, and W_P are all monotonic, the least fixpoint of W_P, denoted by $lfp(W_P)$, always exists and is called the *well-founded model* of P *by reduct*, written $WFS_r(P)$.

(You and Liu 2008) provided the notions of loops and loop formulas for positive basic programs which can be easily extended to general basic programs.

Let P be a basic program, the *dependency graph* of P, denoted by $G_P = (V,E)$, is a directed graph, where $V = \mathscr{A}$ and (u,v) is a directed edge from u to v in E if there is a rule $r \in P$ such that $u = head(r)$ and $v \in S$, for some $S \uplus J \in A_c^*$ and $A \in body(r)$. A set $L \subseteq \mathscr{A}$ is a *loop* in G_P if the subgraph of G_P induced by L is strongly connected. We also say the L is a loop of P. Note that, an atom $a \in \mathscr{A}$ is always a loop of P, we call it a *singleton*.

Let A be a c-atom and $L \subseteq \mathscr{A}$. The *restriction* of A to L, denoted by $A_{|L}$, is $(A_d, A^*_{c|L})$, where $A^*_{c|L} = \{S \uplus J \in A_c^* \mid L \cap S = \emptyset\}$. We use σ_A to denote the formula $\bigvee_{S \uplus J \in A_c^*} S \wedge \neg(A_d \setminus (S \cup J))$, and $\pi_A(L)$ to denote the formula $\bigvee_{S \uplus J \in A^*_{c|L}} S \wedge \neg(A_d \setminus (S \cup J))$. If there is a c-atom A such that $A_c^* = \emptyset$ (resp. $A^*_{c|L} = \emptyset$), then $\sigma_A = \bot$ (resp. $\pi_A(L) = \bot$).

Let r be a basic rule of the form (1). We define the formula $\theta_L(r) = \pi_{A_1}(L) \wedge \cdots \wedge \pi_{A_k}(L) \wedge \neg\sigma_{A_{k+1}} \wedge \cdots \wedge \neg\sigma_{A_n}$.

Let P be a basic program and L a loop of P. A rule $r \in P$ is an *external support of* L if $head(r) \in L$ and $\theta_L(r) \not\equiv \bot$. Below, let $R^-(L)$ be the set of external support rules of L. The *loop formula* for L of P, denoted $LF_P(L)$, is defined as

$$\bigvee_{a \in L} a \supset \bigvee_{r \in R^-(L)} \theta_L(r).$$

Note that if $R^-(L) = \emptyset$, then its loop formula $LF_P(L)$ is equivalent to $\neg.L$.

The *completion* of a basic program P, denoted by $Comp(P)$, consists of the following formulas:

- $\bigwedge_{A \in pos(r)} \sigma_A \wedge \bigwedge_{A \in neg(r)} \neg\sigma_A \supset head(r)$, for each $r \in P$;
- $a \supset \bigvee_{r \in P, head(r) = a} \left(\bigwedge_{A \in pos(r)} \sigma_A \wedge \bigwedge_{A \in neg(r)} \neg\sigma_A \right)$, for each $a \in \mathscr{A}$.

Let P be a basic logic program. We define $LComp(P) = Comp(P) \cup \{LF_P(L) \mid L \text{ is a loop of } P\}$.

Theorem 3.2
Let P be a basic program. A set $M \subseteq \mathscr{A}$ is an r-answer set of P iff M is a model of $LComp(P)$.

3.2 Loops without External Support Rules

Now we define the notion of loops with no external support under a set of literals. Let P be a basic program, L a loop of P, and X a set of literals. We say that a rule $r \in R^-(L)$ is an *external support of L under X* if $X \wedge \theta_L(r) \not\equiv \bot$. In the following, we denote by $R^-(L,X)$ the set of external support rules of L under X. Similarly, if a loop L has no external support rules under X, then under X, $LF_P(L)$ is equivalent to $\neg.L$. Let

$$loop_0(P,X) = \{\neg a \mid a \in L \text{ for a loop } L \text{ of } P \text{ such that } R^-(L,X) = \emptyset\}.$$

It is equivalent to the set of loop formulas of the loops with no external support under X.

As in (Chen et al. 2008; Chen et al. 2013), we can compute $loop_0(P,X)$ in polynomial time in the size of P.

Function $ML_0(P,X,S)$

$ML := \emptyset$; $G :=$ the S induced subgraph of G_P;
For each strongly connected component L of G:
  if $R^-(L,X) = \emptyset$ **then** add L to ML
  else append
  $ML_0(P,X,L \setminus \{head(r) \mid r \in R^-(L,X)\})$ to ML.
return ML.

Proposition 3.1
Let P be a basic program, $X \subseteq Lit$, and $S \subseteq \mathscr{A}$. The function $ML_0(P,X,S)$ returns the following set of loops in $O(n^2)$ time, where n is the size of P:

$$\{L \mid L \subseteq S \text{ is a loop of } P \text{ s.t. } R^-(L,X) = \emptyset \text{ and there does not exist any such loop } L' \text{ s.t. } L \subset L'\}$$

Proposition 3.2
Let P be a basic program and $X \subseteq Lit$.

$$loop_0(P,X) = \bigcup_{L \in ML_0(P,X,\mathscr{A})} \neg.L.$$

3.3 Computing Consequences of a Basic Program

To apply unit propagation, we convert the completion to a set of clauses as follows. Let P be a basic program, and $comp(P)$ denote the following set of clauses:

1. for each $a \in \mathscr{A}$, if there is no rule in P with a as its head, then add $\neg a$;
2. for each c-atom A occurring in P such that $A_c^* = \{S_1 \uplus J_1, \ldots, S_m \uplus J_m\}$, introduce a new variable α_A and m new variables $\beta_1, \ldots, \beta_m$, and add the following clauses:
   - $\neg\alpha_A \vee \beta_1 \vee \cdots \vee \beta_m$,
   - $\alpha_A \vee \neg\beta_i$, for each $1 \leq i \leq m$,
   - $\beta_i \vee \bigvee_{a \in S_i} \neg a \vee \bigvee_{b \in A_d \setminus (S_i \cup J_i)} b$, for each $1 \leq i \leq m$,
   - $\neg\beta_i \vee l$, for each $l \in S_i \cup \{\neg b \mid b \in A_d \setminus (S_i \cup J_i)\}$ and $1 \leq i \leq m$.
3. for each basic rule $r \in P$ in the form of (1), add the following clause:
$$head(r) \vee \neg\alpha_{A_1} \vee \cdots \vee \neg\alpha_{A_k} \vee \alpha_{A_{k+1}} \vee \cdots \vee \alpha_{A_n}.$$
4. if a is an atom and $r_1, \ldots, r_t$, $t > 0$, are all the rules in P with a as their head, then introduce t new variables $v_1, \ldots, v_t$, and add the following clauses:
   - $\neg a \vee v_1 \vee \cdots \vee v_t$,
   - $v_i \vee \bigvee_{A \in pos(r_i)} \neg\alpha_A \vee \bigvee_{A \in neg(r_i)} \alpha_A$, for each $1 \leq i \leq t$,
   - $\neg v_i \vee l$, for each $l \in \{\alpha_A \mid A \in pos(r_i)\} \cup \{\neg\alpha_A \mid A \in neg(r_i)\}$ and $1 \leq i \leq t$.

Note that, if the size of A_c^* for each c-atom A is less than a constant and the size of $S \wedge \neg(A_d \setminus (S \cup J))$ for each $S \uplus J \in A_c^*$ is also less than a constant, then the number of clauses in $comp(P)$ is polynomial in the size of P.

Using $comp(P)$, $loop_0(P,X)$, and $UP(C)$, we can provide a procedure as follows:

$Y := comp(P) \cup loop_0(P,\emptyset)$; $X := \emptyset$;
while $X \neq UP(Y)$ **do**
  $X := UP(Y)$; $Y := Y \cup loop_0(P,X)$;
return $X \cap Lit$.

Note that, the procedure terminates in polynomial time w.r.t. the size of P.

Formally, the above procedure computes $U(P)$, the least fixpoint of the following operator:

$$U^P(X) = UP(comp(P) \cup loop_0(P,X) \cup X) \cap Lit.$$

From Theorem 3.2, $U(P)$ is true in every r-answer set of P.

3.4 Main Results

Let P be a basic program and $I \subseteq Lit$. The function $M(P,I)$ is the least fixpoint of the operator M_P^I defined as:

$$loop_0^I(P,X) = \{a \mid \text{there is a loop } L \text{ of } P \text{ s.t. } a \in L \text{ and for each } r \in R^-(L,I), \text{ there exists } A \in pos(r) \text{ s.t. for each } S \uplus J \in A_c^*, X \cap S \neq \emptyset \text{ or } I \models \neg(S \wedge \neg(A_d \setminus (S \cup J)))\},$$

$$M_P^I(X) = X \cup loop_0^I(P,X).$$

Note that, for any $X \subseteq \mathscr{A}$, $loop_0(P,I) \subseteq \neg.loop_0^I(P,X) \subseteq loop_0(P, I \cup \overline{X})$.

Let us extend Theorem 3.1 to partial interpretations.

Corollary 3.1
Let A be a c-atom and I a partial interpretation. $I \models_{Lit} A$ iff $I \models \sigma_A$ and $I \Vdash_{Lit} A$ iff $I \models \neg\sigma_A$.

Lemma 3.1
Let P be a basic program, I a partial interpretation, X an unfounded set of P, and L a loop of P. If for each $r \in R^-(L,I)$, there exists $A \in pos(r)$ such that for each $S \uplus J \in A_c^*$, $X \cap S \neq \emptyset$ or $I \models \neg(S \wedge \neg(A_d \setminus (S \cup J)))$, then $L \cup X$ is an unfounded set of P w.r.t. I.

The following theorem shows that, the greatest unfounded set of a basic program P w.r.t. I can be computed from the iteration of the operator M_P^I.

Theorem 3.3
Let P be a basic program and I a partial interpretation. $U_P(I) = M(P,I)$.

Now we come to the main theorem - the well-founded semantics of a basic program P by reduct can be computed by unit propagation on completion and loop formulas of loops with no external support.

Theorem 3.4
Let P be a basic program. $WFS_r(P) = U(P)$.

3.5 Well-Founded Semantics by Complement

Given a basic program P, we can convert P to its complement $\widehat{P}$, and then define the well-founded semantics of P based on $\widehat{P}$, which results in a well-founded semantics of basic programs by complement. That is, we define the *well-founded model* of a basic program P *by complement* to be $WFS_r(\widehat{P})$, written by $WFS_c(P)$.

Example 3.1
Consider the following aggregate program P: $p(0) \leftarrow not\ \text{COUNT}(\{\langle 0 : p(0)\rangle, \langle 1 : p(1)\rangle\}) \neq 1$.

We can use a c-atom A to represent the aggregate above, i.e., $A = (A_d, \{\emptyset, \{p(0), p(1)\}\})$ where $A_d = \{p(0), p(1)\}$, and its complement is $\widehat{A} = (A_d, \{\{p(0)\}, \{p(1)\}\})$. One can verify that P has two r-answer sets, $\emptyset$ and $\{p(0)\}$, and $WFS_r(P) = \{\neg p(1)\}$. On the other hand, P has only one c-answer set, $\emptyset$, and $WFS_c(P) = \{\neg p(1), \neg p(0)\}$. In this example, $WFS_c(P)$ is more informative than $WFS_r(P)$.

For a positive basic program P, we know that models of $LComp(P)$ are c-answer sets of P (You and Liu 2008). As loop formulas of loops without external support form a subset of all loop formulas, by monotonicity of propositional logic, $WFS_c(P)$ approximates all c-answer sets of P.

Proposition 3.3
Let P a basic program and M a c-answer set of P. Then, for any atom a, if $a \in WFS_c(P)$ then $a \in M$ and if $\neg a \in WFS_c(P)$ then $a \notin M$.

From Theorem 3.4 it is easy to see that, $WFS_c(P)$ can be computed from $comp(\widehat{P})$ and $loop_0(\widehat{P}, X)$ using unit propagation for any basic program P.

Corollary 3.2
Let P be a basic program. $WFS_c(P) = U(\widehat{P})$.

Finally, we can show a relation with the (ultimate) well-founded semantics of (Pelov et al. 2007), based on a translation, τ_a, which models an aggregate program as a basic program under c-answer sets (Son et al. 2007).

Proposition 3.4
Let P be an aggregate program with only monotonic aggregate atoms. $WFS_c(\tau_a(P))$ coincides with the (ultimate) well-founded semantics of P defined in (Pelov et al. 2007).

4 Loops with at Most One External Support

Unlike the case for normal and disjunctive programs, for positive basic programs, the loop formulas of loops that have exactly one external support rule are not always equivalent to sets of binary clauses. However, there is a special case of these loops whose loop formulas are equivalent to sets of binary clauses. In particular, let P be a positive basic program and L a loop of P, a rule $r \in P$ is a *unary external support* of L if $head(r) \in L$, and $A^*_{c|L}$ has and only has one element for each $A \in body(r)$. Let $R^-_u(L)$ be the set of unary external support rules of L. Given a set X of literals, we say that a rule $r \in R^-(L)$ is a *unary external support of L under X* if for each $A \in body(r)$, there is and only is one $S \uplus J \in A^*_{c|L}$ such that X is consistent with $S \cup \neg.(A_d \setminus (S \cup J))$. We denote by $R^-_u(L,X)$ the set of all these rules. For each $r \in R^-_u(L,X)$, we denote by $body_L(r)$ the union of $S \cup \neg.(A_d \setminus (S \cup J))$ for each $A \in body(r)$, where $S \uplus J$ is the only element in $A^*_{c|L}$ that is consistent with X. It is easy to see that, loop formulas of loops with one external support rule which is also unary correspond to sets of binary clauses. Note that, if P is a normal logic program, then $R^-_u(L) = R^-(L)$ and $R^-_u(L,X) = R^-(L,X)$.

We now consider the set of loop formulas of the loops that have exactly one unary external support rule under a set X of literals:

$$loop^u_1(P,X) = \{\, \neg a \vee l \mid a \in L, l \in body_L(r), \text{ for some loop } L \text{ and rule } r \text{ s.t.} \\ R^-(L,X) = R^-_u(L,X) = \{r\}\,\}.$$

Modified from $ML_0(P,X,S)$, let us define the following procedure for computing $loop_0(P,X) \cup loop^u_1(P,X)$.

Function $ML_1(P,X,S)$

$ML := \emptyset$; $G :=$ the S induced subgraph of G_P;
For each strongly connected component L of G:

Let $V = \{\, a \in L \mid$ there is and only is one rule $r \in R^-(L,X)$ such that $r \in R^-_u(L,X)$ and $a = head(r)\,\}$,
$W = \{head(r) \mid r \in R^-(L,X)\} \setminus V$.

if $R^-(L,X) = \emptyset$ **then** add $\neg.L$ to ML;
else if $R^-(L,X) = R_u^-(L,X) = \{r\}$
then add $\{\neg a \vee l \mid a \in L, l \in body_L(r)\}$ and append $ML_1(P,X,L \setminus \{head(r)\})$ to ML ;
else if $W \neq \emptyset$ **then** append $ML_1(P,X,L \setminus W)$ to ML;
else for each $a \in V$ append $ML_1(P,X,(L \setminus V) \cup \{a\})$ to ML.

return ML.

Proposition 4.1
Let P be a positive basic program, $X \subseteq Lit$, and $S \subseteq \mathscr{A}$. The set of formulas returned from $ML_1(P,X,\mathscr{A})$ is equivalent to $loop_0(P,X) \cup loop_1^u(P,X)$. The procedure terminates in $O(n^3)$, where n is the size of P.

Now if we add $loop_1^u(P,X)$ to the procedure for computing $U(P)$, a more powerful operator can be defined:

$$T^P(X) = UP(comp(P) \cup loop_0(P,X) \cup loop_1^u(P,X) \cup X) \cap Lit.$$

Denote by $T(P)$ the least fixpoint of $T^P(X)$. From Theorem 3.2, for any positive basic program P, $T(P)$ is true in every c-answer set, as well as every r-answer set, of P.

We have implemented the algorithms in this paper in a prototype system that is available on line[1]. For any program P that can be accepted by lparse, it first computes $T(P)$ and then adds $\{\leftarrow not\, a \mid a \in T(P)\} \cup \{\leftarrow a \mid \neg a \in T(P)\}$ to P. The resulting new program is then submitted to an ASP solver. To test the effectiveness of our preprocessor, we use the familiar Hamiltonian Circuit (HC) problem encoded with the following cardinality constraints:

$$\leftarrow 2\{dhc(X,Y) : arc(X,Y)\}, vertex(Y).$$
$$\leftarrow 2\{dhc(X,Y) : arc(X,Y)\}, vertex(X).$$

Our experiments[2] showed that, for most programs, information from $T(P)$ makes both smodels and clasp run faster, when lookahead operators are turned off.

5 Final Remarks

We have proposed using unit propagation with program completion and loop formulas of loops with at most one external support to capture and extend well-founded model of a basic logic programs, continuing our work along this line for normal and disjunctive logic programs (Chen et al. 2013). We believe that this work is of not only theoretical value but practical uses.

For logic programs with constraints, the size of a constraint is typically measured by the size of the constraint's domain. Measured this way, our procedures/algorithms given in this paper are theoretically exponential. However, just like in the case of normal programs where researchers have found practical benefits of using loop formulas even if there could be exponentially many loops, we have shown that the loop formula approach for basic logic programs with c-atoms can also benefit answer set computation. We have confirmed this by experiments using the HC problem for graphs with a special structure to demonstrate this. The same should also hold for logic programs whose dependency graphs have a similar structure.

[1] `http://staff.ustc.edu.cn/~jianmin/cloopC/`
[2] `http://staff.ustc.edu.cn/~jianmin/cloopC/cloopC/experiment.html`

Acknowledgements

This work had been supported by the National Hi-Tech Project of China under grant 2008AA01Z150, the Natural Science Foundation of China under grant 60745002 and 61175057, the USTC Key Direction Project, the Fundamental Research Funds for the Central Universities, the Youth Innovation Fund of USTC, and HK RGC GRF 616909.

References

CHEN, X., JI, J., AND LIN, F. 2008. Computing loops with at most one external support rule. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-08)*. 401–410.

CHEN, X., JI, J., AND LIN, F. 2009. Computing loops with at most one external support rule for disjunctive logic programs. In *Proceedings of the 25th International Conference on Logic Programming (ICLP-09)*. 130–144.

CHEN, X., JI, J., AND LIN, F. 2013. Computing loops with at most one external support rule. *ACM Transactions on Computational Logic (TOCL) 14,* 1.

EITER, T., IANNI, G., LUKASIEWICZ, T., AND SCHINDLAUER, R. 2011. Well-founded semantics for description logic programs in the semantic web. *ACM Transactions on Computational Logic (TOCL) 12,* 2, 1–11.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. Conflict-driven answer set solving. In *Proceddings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*. 386–392.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic programming (ICLP-88)*. 1070–1080.

LIU, L. AND TRUSZCZYNSKI, M. 2006. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research 27,* 1, 299–334.

PELOV, N., DENECKER, M., AND BRUYNOOGHE, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming 7,* 3, 301–353.

SHEN, Y., YOU, J., AND YUAN, L. 2009. Characterizations of stable model semantics for logic programs with arbitrary constraint atoms. *Theory and Practice of Logic Programming 9,* 4, 529–564.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

SON, T., PONTELLI, E., AND TU, P. 2007. Answer sets for logic programs with arbitrary abstract constraint atoms. *Journal of Artificial Intelligence Research 29,* 1, 353–389.

VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

WANG, K. AND ZHOU, L. 2005. Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM Trans. Comput. Logic 6,* 2, 295–327.

WANG, Y., LIN, F., ZHANG, M., AND YOU, J. 2012. A well-founded semantics for basic logic programs with arbitrary abstract constraint atoms. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)*.

YOU, J.-H. AND LIU, G. 2008. Loop formulas for logic programs with arbitrary constraint atoms. In *AAAI-08*. 584–589.

YOU, J.-H., MORRIS, J., AND BI, Y. 2012. Reconciling well-founded semantics of dl-programs and aggregate programs. In *Proceedings of the 28th International Conference on Logic Programming (ICLP-12)*. 235–246.