Computing Loops with at Most One External Support Rule

XIAOPING CHEN and JIANMIN JI, University of Science and Technology of China
FANGZHEN LIN, Hong Kong University of Science and Technology

A consequence of a logic program under answer set semantics is one that is true for all answer sets. This article considers using loop formulas to compute some of these consequences in order to increase the efficiency of answer set solvers. Since computing loop formulas are in general intractable, we consider only loops with either no external support or at most one external support, as their loop formulas are either unit or binary clauses. We show that for disjunctive logic programs, loop formulas of loops with no external support can be computed in polynomial time, and that an iterative procedure using unit propagation on these formulas and the program completion computes the well-founded models in the case of normal logic programs and the least fixed point of a simplification operator used by DLV for disjunctive logic programs. For loops with at most one external support, their loop formulas can be computed in polynomial time for normal logic programs, but are NP-hard for disjunctive programs. So for normal logic programs, we have a procedure similar to the iterative one for loops without any external support, but for disjunctive logic programs, we present a polynomial approximation algorithm. All these algorithms have been implemented, and our experiments show that for certain logic programs, the consequences computed by our algorithms can significantly speed up current ASP solvers cmodels, clasp, and DLV.

Categories and Subject Descriptors: I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Inference engines, Logic programming, Nonmonotonic reasoning and belief revision*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computational logic*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Answer set semantics, normal logic programs, disjunctive logic programs, loop formulas, well-founded semantics, answer set solvers

ACM Reference Format:
Chen, X., Ji, J., and Lin, F. 2013. Computing loops with at most one external support rule. *ACM Trans. Comput. Logic* 14, 1, Article 3 (February 2013), 34 pages.
DOI: http://dx.doi.org/10.1145/2422085.2422088

1. INTRODUCTION

This article is about answer set programming (ASP) where the main computational task is to compute the answer sets of a logic program. In this context, consequences of a logic program, that is, those that are true in all answer sets, are of interest, as they can be used to simplify the given program and help compute its answer sets. The best known example is the well-founded model for normal logic programs [Van Gelder et al. 1991], which always exists and can be computed efficiently. All literals in the well-founded model are consequences, and in all current ASP solvers, a logic program

---

This article expands on two previous conference papers Chen et al. [2008, 2009].
Authors' addresses: X. Chen and J. Ji, School of Computer Science and Technology, University of Science and Technology of China, 96 Jinzhai Road, Hefei City, Anhui Province, P.R. China; emails: {xpchen, jizheng}@mail.ustc.edu.cn; F. Lin, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong; email: flin@cs.ust.hk.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 1529-3785/2013/02-ART3 $15.00
DOI: http://dx.doi.org/10.1145/2422085.2422088

is first simplified by its well-founded model. A natural question then is whether there are other consequences of a logic program that can be computed efficiently and used to simplify the given logic program. Motivated by this, we propose to use loops and loop formulas to compute consequences of a logic program.

The notions of loops and loop formulas were first proposed by Lin and Zhao [2004] for propositional normal logic programs. They showed that a set of atoms is an answer set of a normal logic program if and only if it satisfies the completion and the loop formulas of the program. The notions and the result have been extended to disjunctive logic programs, nested logic programs [Lee and Lifschitz 2003], general logic programs [Ferraris et al. 2006], normal logic programs with variables [Chen et al. 2006], propositional circumscription [Lee and Lin 2006], and arbitrary first-order formulas with sable model semantics [Lee and Meng 2008]. Generally, the consequences of a logic program are the logical consequences of its completion and loop formulas. However, deduction in propositional logic is coNP-hard, and in general, there may be an exponential number of loops [Lifschitz and Razborov 2006]. One way to overcome these problems is to use some tractable inference rules and consider only those loop formulas that can be used effectively by these inference rules and can be computed efficiently at the same time.

In this article, we choose unit propagation as the inference rule. To see which loops would yield unit-propagation-friendly loop formulas, let us look at the form of loop formulas for normal logic programs (similarly for disjunctive logic programs).

According to Lin and Zhao [2004], a loop L is a set of atoms, and its loop formula is a sentence of the following form.

$$\bigvee L \supset \bigvee_{r \in R^-(L)} body(r),$$

where $R^-(L)$ is the set of so-called *external support rules* of L, and $body(r)$ is the conjunction of the literals in the body of the rule r. Without going into details about the definition of external supports and how they are computed, we see that if a loop L has no external supports, then its loop formula is equivalent to the following set of literals.

$$\{\, \neg a \mid a \in L \,\},$$

and if a loop L has exactly one external support rule, say r, then its loop formula is equivalent to the following set of binary clauses.

$$\{\, \neg a \vee l \mid a \in L,\ l \in body(r) \,\}.$$

Thus we see that loops that have at most one external support rule are special in that their loop formulas will yield unit or binary clauses that can be used effectively by unit propagation.

More generally, if we assume a set A of literals, then for any loop that has at most one external support rule whose body is not false under A, its loop formula is equivalent to either a set of literals or a set of binary clauses under A.

Since the completion of a logic program can be computed and converted to a set of clauses in linear time (by introducing new variables if necessary), if these loop formulas can also be computed in polynomial time, we then have a polynomial time algorithm for computing some consequences of a logic program. In general terms, one such procedure is as follows.

Input: a logic program P.
(1) Initialize $U = \emptyset$, and convert $Comp(P)$ to a set C of clauses.
(2) Based on U, compute a set of loop formulas and convert them into a set L of clauses.

(3) Let $K = \{ \varphi \mid U \cup C \cup L \vdash_P \varphi \}$, where $\vdash_P$ is a sound inference rule in propositional logic (such as unit propagation).
(4) If $K \setminus U = \emptyset$, then return K, else let $U = K$ and go back to step 2.

We shall show that the loop formulas of loops with no external support rules can indeed be computed in polynomial time. For normal logic programs, where $\vdash_P$ is unit propagation and the class of loops under U is that which has no external support under U, then the preceding procedure computes essentially the same set of literals as does the *Expand* operator in smodels [Simons et al. 2002]. In particular, it computes the well-founded model [Van Gelder et al. 1991] when the given normal logic program has no constraint. In general, this procedure can be more powerful when loops with at most one external support rule are considered, and these extra consequence can help ASP solvers. This is supported experimentally. Our earlier experimental results [Chen et al. 2008] showed that consequences computed by the preceding procedure can speed up cmodels [Giunchiglia et al. 2006] significantly. However, the sizes of the logic programs that we tried were not big enough to show its benefits on clasp [Gebser et al. 2007]. We have since performed more experiments with a better implementation of the procedure previously outlined. We report here that even for clasp, the computed consequences can speed it up significantly on large programs—the largest program that we tried is an instance of the Hamiltonian Circuit problem with 1,000 nodes. For this program, clasp needs almost 3,000 seconds to return an answer set. Our system runs in 18 seconds to return the consequences of the program, as computed by our preceding procedure, and once these consequences are added to the original program as constraints, clasp returns an answer set in about 6 seconds!

The preceding procedure works in principle for more general logic programs, such as disjunctive logic programs. As expected, loop formulas of loops with no external support rules required in the procedure can still be computed in polynomial time. The consequences computed from the procedure are closely related to the preprocessing step in DLV, well-founded models, and greatest unfounded sets in disjunctive logic programs as well. However, the problem of computing the loop formulas of loops with at most one external support rule turns out to be NP-hard. We thus propose a polynomial algorithm for computing some of these loop formulas and show experimentally that this polynomial approximation algorithm can be effective in practice.

This article is organized as follows. We briefly review the basic notions of logic programming in the next section. Then we define loops with at most one external support rule under a given set of literals and consider how to compute their loop formulas for normal and disjunctive logic programs. We then consider how to use these loop formulas to derive consequences of a program using unit propagation and discuss related work. Since the problem of computing the loop formulas of loops with at most one external support rule is different for normal and disjunctive logic programs, we first consider the case for normal logic programs, then extend it to disjunctive logic programs. We also show experimentally that when loop formulas of loops with at most one external support rule is considered in the procedure, these extra consequences can help ASP solvers find answer sets of certain logic programs.

2. PRELIMINARIES

In this article, we consider only fully grounded finite logic programs.

A *normal logic program* is a finite set of (normal) rules of the following form.

$$H \leftarrow a_1, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n, \tag{1}$$

where $n \geq m \geq 0$ and a_i, $1 \leq i \leq n$, are atoms. H is either empty or an atom. If H is empty, then this rule is also called a *constraint*, and if H is an atom, it is a *proper rule*.

A *disjunctive logic program* is a finite set of (disjunctive) rules of the following form.

$$a_1 \vee \cdots \vee a_k \leftarrow a_{k+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n, \tag{2}$$

where $n \geq m \geq k \geq 0$ and $a_1, \ldots, a_n$ are atoms. Note that if $k \leq 1$, the rule is a *normal rule*.

We will also write rule r of Equation (2) as

$$head(r) \leftarrow body(r), \tag{3}$$

where $head(r)$ is $a_1 \vee \cdots \vee a_k$, $body(r) = body^+(r) \wedge \neg body^-(r)$, $body^+(r)$ is $a_{k+1} \wedge \cdots \wedge a_m$, and $\neg body^-(r)$ is $\neg a_{m+1} \wedge \cdots \wedge \neg a_n$, and we identify $head(r)$, $body(r)$, $body^+(r)$ with their corresponding sets of literals, specially, we identify $body^-(r)$ with the set of atoms occurred in $\neg body^-(r)$.

Given a logic program P, we denote by $Atoms(P)$ the set of atoms in it, and $Lit(P)$ the set of literals constructed from $Atoms(P)$.

$$Lit(P) = Atoms(P) \cup \{\, \neg a \mid a \in Atoms(P) \,\}.$$

Given a literal l, the *complement* of l, written $\bar{l}$, is $\neg a$ if l is a and a if l is $\neg a$, where a is an atom. For a set L of literals, we let $\overline{L} = \{\bar{l} \mid l \in L\}$.

2.1. Answer Sets

Now we review the definitions of *answer sets* for normal and disjunctive logic programs.

To define the answer sets of a normal logic program with constraints, we first define the *stable models* of a normal logic program that do not have any constraints [Gelfond and Lifschitz 1988]. Given a normal logic program P without constraints and a set S of atoms, the Gelfond-Lifschitz transformation of P on S, written P_S, is obtained from P by deleting the following.

(1) Each rule that has a formula $not\, p$ in its body with $p \in S$.
(2) All formulas of the form $not\, p$ in the bodies of the remaining rules.

Clearly, for any S, P_S is a set of rules without any negative literals so that P_S has a unique minimal model, denoted by $\Gamma_P(S)$. Now a set S of atoms is a stable model of P if and only if $S = \Gamma_P(S)$.

In general, given a normal logic program P that may have constraints, a set S of atoms is an answer set of P if and only if it is a stable model of the program obtained by deleting all the constraints in P, and it satisfies all the constraints in P, that is, for any constraint of the form (1) such that H is empty, either $a_i \notin S$ for some $1 \leq i \leq m$ or $a_j \in S$ for some $m + 1 \leq j \leq n$.

The answer sets of a disjunctive logic program are defined as in Gelfond and Lifschitz [1991]. Given a disjunctive logic program P and a set S of atoms, the Gelfond-Lifschitz transformation of P on S, written P_S, is defined the same as for normal logic programs. Clearly, for any S, P_S is the set of rules without any negative literals so that P_S has a set of minimal models, denoted by $\Gamma_P(S)$. Now a set S of atoms is an answer set of P iff $S \in \Gamma_P(S)$.

2.2. Completions

The *completion* of a disjunctive (normal) logic program P [Lee and Lifschitz 2003], $Comp(P)$, is defined as the set of propositional formulas that consists of the implication

$$body(r) \supset head(r), \tag{4}$$

for every rule r in P, and the implication

$$a \supset \bigvee_{r \in P,\, a \in head(r)} \left(body(r) \wedge \bigwedge_{p \in head(r) \setminus \{a\}} \neg p \right), \tag{5}$$

for each atom $a \in Atoms(P)$.

Note that if P is a normal logic program without constraints, $Comp(P)$ is equivalent to the Clark's completion of P [Clark 1978]. If P has constraints, then the completion of P is the union of Clark's completion and the set of sentences corresponding to the constraints in P: if $\leftarrow a_1, ..., a_m, not\, a_{m+1}, ..., not\, a_n$ is a constraint, then its corresponding sentence is $\neg(a_1 \wedge \cdots \wedge a_m \wedge \neg a_{m+1} \wedge \cdots \wedge \neg a_n)$.

As we mentioned, we will convert the completion into a set of clauses and use unit propagation as the inference rule. Since unit propagation is not logically complete, it matters how we transform the formulas in the completion into clauses. In the following, let $comp(P)$ be the set of following clauses.

(1) For each $a \in Atoms(P)$, if there is no rule in P with a in its head, then add $\neg a$.
(2) If r is not a constraint, then add $head(r) \vee \bigvee \overline{body(r)}$.
(3) If r is a constraint, then add the clause $\bigvee \overline{body(r)}$.
(4) If a is an atom and $r_1, \ldots, r_t$, $t > 0$, are all the rules in P with a in their heads, then introduce t new variables $v_1, \ldots, v_t$, and add the following clauses.

$$\neg a \vee v_1 \vee \cdots \vee v_t,$$
$$v_i \vee \bigvee \overline{body(r_i)} \vee \bigvee_{p \in head(r_i) \setminus \{a\}} p, \text{ for each } 1 \leq i \leq t,$$
$$\neg v_i \vee l, \text{ for each } l \in body(r_i) \cup \overline{head(r_i) \setminus \{a\}} \text{ and } 1 \leq i \leq t.$$

2.3. Loops and Loop Formulas

We now briefly review the notions of loops and loop formulas in disjunctive (normal) logic programs [Lee and Lifschitz 2003].

Given a disjunctive (normal) logic program P, the *positive dependency graph* of P, written G_P, is the directed graph whose vertices are atoms in P, and there is an arc from p to q if there is a rule $r \in P$ such that $p \in head(r)$ and $q \in body^+(r)$. A set L of atoms is said to be a loop of P if for any p and q in L, there is a non-empty path from p to q in G_P such that all the vertices in the path are in L, that is, the L-induced subgraph of G_P is strongly connected.

Given a loop L, a rule r is an *external support* of L if $head(r) \cap L \neq \emptyset$ and $L \cap body^+(r) = \emptyset$. In the following, let $R^-(L)$ be the set of external support rules of L. Then the *loop formula* of L under P, written $LF(L, P)$, is the following implication.

$$\bigvee_{p \in L} p \supset \bigvee_{r \in R^-(L)} \left(body(r) \wedge \bigwedge_{q \in head(r) \setminus L} \neg q \right). \tag{6}$$

2.4. Unfounded Sets

The notion of unfounded sets for normal logic programs, which provide the basis for negative conclusions in the well-founded semantics [Van Gelder 1989], has been extended to disjunctive logic programs [Leone et al. 1997].

Let P be a disjunctive logic program and A be a set of literals and A set of atoms X is an *unfounded set* for P with respect to A if, for each $a \in X$, for each rule $r \in P$ such that $a \in head(r)$, at least one of the following conditions holds.

(1) $\overline{A} \cap body(r) \neq \emptyset$, that is, the body of r is false with respect to A.
(2) $body^+(r) \cap X \neq \emptyset$, that is, some positive body literal belongs to X.
(3) $(head(r) \setminus X) \cap A \neq \emptyset$, that is, an atom in the head of r, distinct from elements in X, is true with respect to A.

Note that if P is a normal logic program, unfounded sets defined here coincide with the definition given for normal logic programs in Van Gelder [1989]. For normal logic programs, the union of all unfounded sets with respect to A is also an unfounded set with respect to A (called the *greatest unfounded set*). But this is not generally true for disjunctive logic programs; thus for some disjunctive logic program P and set of literals A, the union of two unfounded sets is not an unfounded set, and the greatest unfounded set of P with respect to A does not exist. From Proposition 3.7 in Leone et al. [1997], the greatest unfounded set exists for any P if A is unfounded-free. Formally, a set of literals A is *unfounded-free* for a disjunctive logic program P, if $A \cap X = \emptyset$ for each unfounded set X for P with respect to A. If A is unfounded-free for P, then the greatest unfounded set exists. In the following, we use $GUS_P(A)$ to denote the greatest unfounded set for P with respect to A.

Loops and unfounded sets are closely related [Anger et al. 2006; Lee 2005]. In this article, we show that the greatest unfounded sets (if they exist) can be computed from loops that have no external support rules.

2.5. Unit Propagation

We use unit propagation as the inference rule for deriving consequences from the completion and loop formulas of a logic program. Given a set Γ of clauses, we let $UP(\Gamma)$ be the set of literals that can be derived from Γ by unit propagation. Formally, it can be defined as follows.

Function $UP(\Gamma)$
  if $(\emptyset \in \Gamma)$ **then return** Lit;
  $A := unit\_clause(\Gamma)$;
  if A is inconsistent **then return** Lit;
  if $A \neq \emptyset$ **then return** $A \cup UP(assign(A, \Gamma))$ **else return** $\emptyset$;

where Lit is the set of literals in the language, $unit\_clause(\Gamma)$ returns the union of all unit clauses in Γ, and $assign(A, \Gamma)$ is $\{ c \mid \text{ for some } c' \in \Gamma, c' \cap A = \emptyset, \text{ and } c = c' \setminus \overline{A} \}$.

3. COMPUTING LOOPS WITH AT MOST ONE EXTERNAL SUPPORT FOR NORMAL LOGIC PROGRAMS

The basic theorem about loop formulas says that a set of atoms is an answer set of a logic program if and only if it is a model of the program's completion and loop formulas.[1] This is the case for normal logic programs [Lin and Zhao 2004] as well as for disjunctive logic programs [Lee and Lifschitz 2003]. This means that a sentence is a consequence of a logic program if and only if it is a logical consequence of the program's completion and loop formulas. The problem is that logical entailment in propositional logic is coNP-complete and that, in the worst case, there may be an exponential number of loops and loop formulas. Here, we suggest using unit propagation as the inference

[1]Or the program and its loop formulas if singletons are always considered loops.

rule and some special classes of loops whose loop formulas can be computed efficiently. We consider loops with at most one external support rule.

In this section, we consider the case for normal logic programs.

3.1. Loops with at Most One External Support

Consider first loops without any external support rules. If a loop L has no external support rules, that is $R^-(L) = \emptyset$, then its loop formula (Equation (6)) is equivalent to $\bigwedge_{p\in L} \neg p$. More generally, if we already know that A is a set of literals that are true in all answer sets and that $\overline{A} \cap body(r) \neq \emptyset$ for every rule $r \in R^-(L)$, then under A, the loop formula of L is equivalent to $\bigwedge_{p\in L} \neg p$.

Thus we extend the notion of external support rules and have it conditioned on a given set of literals. Let P be a logic program and A a set of literals. We say that a rule $r \in R^-(L)$ is an *external support of L under A* if $\overline{A} \cap body(r) = \emptyset$. In the following, we denote by $R^-(L,A)$ the set of external support rules of L under A. Now given a logic program P and a set A of literals, let

$$\begin{aligned} loop_0(P,A) &= \{\, L \mid L \text{ is a loop of } P \text{ such that } R^-(L,A) = \emptyset \,\}, \text{ and} \\ floop_0(P,A) &= \{\, \neg a \mid a \in L \text{ for a loop } L \in loop_0(P,A) \,\}. \end{aligned}$$

Then $loop_0(P,A)$ is the set of loops that does not have any external support rules under A, and $floop_0(P,A)$ is equivalent to the set of loop formulas of these loops. In particular, the set of loop formulas of loops without any external support rules is equivalent to $floop_0(P,\emptyset)$.

Similarly, we can consider the set of loops that has exactly one external support rule under a set A of literals and the set of loop formulas of these loops.

$$\begin{aligned} loop_1(P,A) &= \{\, L \mid L \text{ is a loop of } P \text{ such that } R^-(L,A) = \{r\}, \text{ for only one rule} \\ &\qquad r \in P \,\}, \text{ and} \\ floop_1(P,A) &= \{\neg a \vee l \mid a \in L, l \in body(r), \text{ where } L \in loop_1(P,A) \text{ and } r \text{ is the} \\ &\qquad \text{only external support rule of } L \text{ under } A \,\}. \end{aligned}$$

In particular, $floop_1(P,\emptyset)$ is equivalent to the set of loop formulas of the loops that have exactly one external support rule in P.

Notice that if L is a loop of P without any external supports under A, then L is also an unfounded set of P with respect to A. However, the other way around is not true in general. An unfounded set does not need to be a loop. For instance, consider $P = \{a \leftarrow not\, b.\ b \leftarrow not\, a.\}$. There are no loops here, but $\{b\}$ is an unfounded set with respect to $A = \{a\}$. Furthermore, to our best knowledge, there is no corresponding notion to loops with exactly one external support in the literature on unfounded sets.

We now consider how to compute $floop_0(P,A)$ and $floop_1(P,A)$. We start with $floop_0(P,A)$. Let $ml_0(P,A)$ be the set of maximal loops that do not have any external support rules under A, that is, a loop is in $ml_0(P,A)$ if it is a loop of P such that $R^-(L,A) = \emptyset$ and there does not exist any other such loop L' where $L \subset L'$. Clearly,

$$floop_0(P,A) = \bigcup_{L\in ml_0(P,A)} \overline{L}.$$

The following proposition is immediate.

PROPOSITION 3.1. *Let P be a normal logic program and A a set of literals. If L_1 and L_2 are two loops of P that do not have any external support rules under A, and $L_1 \cap L_2 \neq \emptyset$, then $L_1 \cup L_2$ is also a loop of P that does not have any external support rules under A.*

Thus for any normal logic program P and $A \subseteq Lit(P)$, loops in $ml_0(P,A)$ are pairwise disjoint. This means that there can only be at most $|Atoms(P)|$ loops in $ml_0(P,A)$.

To compute $ml_0(P,A)$, consider G_P, the positive dependency graph of P. If L is a loop of P, then there must be a strongly connected component C of G_P such that $L \subseteq C$. For L to be in $ml_0(P,A)$, it is impossible that $L \subset C$ and $R^-(C,A) = \emptyset$, then there are two cases: either $L = C$ and $R^-(C,A) = \emptyset$ or $L \subset C$, $R^-(C,A) \neq \emptyset$ and $R^-(L,A) = \emptyset$. If it is the latter, then for any $r \in R^-(C,A)$, it must be that $head(r) \notin L$, for otherwise, r must be in $R^-(L,A)$, which is a contradiction with $R^-(L,A) = \emptyset$. Thus if $R^-(C,A) \neq \emptyset$, then any subset of C that is in $ml_0(P,A)$ must also be a subset of $S = C \setminus \{head(r) \mid r \in R^-(C,A)\}$. Thus instead of G_P, we can recursively search the S induced subgraph of G_P. This motivates the following procedure for computing $ml_0(P,A)$.

$\mathcal{ML}_0(P,A) := \mathcal{ML}_0(P,A,Atoms(P))$;
Function $\mathcal{ML}_0(P,A,S)$: P a normal program, A and S sets of literals of P
  $\mathcal{ML} := \emptyset$; $G :=$ the S induced subgraph of G_P;
  For each strongly connected component L of G:
    if $R^-(L,A) = \emptyset$ **then** add L to $\mathcal{ML}$;
    else append $\mathcal{ML}_0(P,A,L \setminus \{ head(r) \mid r \in R^-(L,A) \})$ to $\mathcal{ML}$;
  return $\mathcal{ML}$;

where G_P is the positive dependency graph of P.

From our previous discussion, the following result is immediate.

THEOREM 3.2. *For any normal logic program P, any $A \subseteq Lit(P)$, and any $C \subseteq Atoms(P)$, the preceding function $\mathcal{ML}_0(P,A,C)$ returns the following set of loops.*

$$\{ L \mid L \subseteq C \text{ is a loop of } P \text{ such that } R^-(L,A) = \emptyset, \text{ and there does not exist any other such loop } L' \text{ such that } L \subset L' \}$$

in $O(n^2)$, where n is the size of P as a set. Particularly, $\mathcal{ML}_0(P,A) = ml_0(P,A)$.

We now consider the problem of computing $floop_1(P,A)$. For any rule r, let $ml_1(P,A,r)$ be the set of maximal loops of P that have r as their only external support rule under A, that is, L is in $ml_1(P,A,r)$ if it is a loop of P such that $R^-(L,A) = \{r\}$ and there is no other such loop L' such that $L \subset L'$. Notice that this definition is meaningful only if r is a proper rule of P. If r is a constraint, then it can never be an external support rule of any loop, thus $ml_1(P,A,r) = \emptyset$. Now let

$$ml_1(P,A) = \bigcup_{r \in P} ml_1(P,A,r).$$

It is easy to see that loops in $ml_1(P,A,r)$ are pairwise disjoint. Thus the size of $ml_1(P,A,r)$ is bounded by $|Atoms(P)|$, and $ml_1(P,A)$ by $m\,|Atoms(P)|$, where m is the number of proper rules in P.

It is easy to see that $floop_1(P,A)$ is the following set.

$$\bigcup_{L \in ml_1(P,A)} \{\neg a \vee l \mid a \in L, l \in body(r), R^-(L,A) = \{r\}\}.$$

Thus to compute $floop_1(P,A)$, we need to compute only $ml_1(P,A)$, and for the latter, we only need to compute $ml_1(P,A,r)$ for all proper rules $r \in P$ such that $\overline{A} \cap body(r) = \emptyset$. At first glance, this problem can be trivially reduced to that of computing $ml_0(P \setminus \{r\},A)$, the set of maximal loops that does not have any external support rules under A in the program obtained from P by deleting r from it. However, while it is true that if $L \in ml_0(P \setminus \{r\},A)$ and r is the only external support rule of L under A in P, then $L \in ml_1(P,A,r)$, the converse is not true in general.

Example 1. Consider the following logic program P.

$$a \leftarrow b,\ c.$$
$$b \leftarrow a.$$
$$b \leftarrow c.$$
$$c \leftarrow b.$$

It is easy to see that $ml_1(P, \emptyset, b \leftarrow c) = \{\{a, b\}\}$, but $ml_0(P \setminus \{b \leftarrow c\}, \emptyset) = \{\{a, b, c\}\}$.

We do not yet know any efficient way of computing $ml_1(P, A, r)$, but for the purpose of computing $\mathit{floop}_1(P, A) \cup \mathit{floop}_0(P, A)$, $ml_0(P \setminus \{r\}, A)$ is enough.

PROPOSITION 3.3. *For any normal logic program P and a set A of literals, $\mathit{floop}_0(P, A)$ implies that $\mathit{floop}_1(P, A)$ is equivalent to the following theory.*

$$\bigcup_{\overline{A} \cap body(r) = \emptyset,\ L \in ml_0(P \setminus \{r\}, A)} \{\neg a \vee l \mid a \in L, l \in body(r)\}. \tag{7}$$

Note that we have proved that $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$ is logically equivalent to $\mathit{floop}_0(P, A) \cup (7)$. From the proof we can see that if a binary clause $C \in \mathit{floop}_1(P, A)$ is not entailed by $\mathit{floop}_0(P, A)$, then C is in Equation (7), and if a binary clause $C' \in (7)$ is not entailed by $\mathit{floop}_0(P, A)$ then C is in $\mathit{floop}_1(P, A)$, thus $\mathit{UP}(\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)) = \mathit{UP}(\mathit{floop}_0(P, A) \cup (7))$.

So to summarize, for normal logic programs, to compute $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$, we first compute $ml_0(P, A)$, and then for each proper rule $r \in P$ such that $\overline{A} \cap body(r) = \emptyset$, we compute $ml_0(P \setminus \{r\}, A)$. The worst case complexity of this procedure is $O(n^3)$, where n is the size of P. There are a lot of redundancies in this procedure, as the computations of $ml_0(P, A)$ and $ml_0(P \setminus \{r\}, A)$ overlap a lot. These redundancies can and should be eliminated in the actual implementation.

3.2. Computing Consequences of a Normal Logic Program

By Lin and Zhao's theorem [2004] on loop formulas, logical consequences of $\mathit{comp}(P) \cup \mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$ are also consequences of P. Since $\mathit{comp}(P)$ and $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$ can all be computed in polynomial time with a polynomial time inference rule, we thus get a polynomial time algorithm for computing some consequences of a logic program. In this article, we consider using UP, the unit propagation.

Consider first loops without any external support rules. When $\vdash_P$ is unit propagation UP and the loop formulas are those from ML_0 (maximal loops with no external support), the iterative procedure given in the Introduction becomes the following.

Function $T_0(P)$: P is a logic program
  $X := \emptyset$; $Y := \mathit{comp}(P) \cup \{$ loop formulas of loops in $\mathit{ML}_0(P, \emptyset)\}$;
  while $X \neq \mathit{UP}(Y)$ **do**
    $X := \mathit{UP}(Y)$; $Y := Y \cup \{$ loop formulas of loops in $\mathit{ML}_0(P, X)\}$;
  return $X \cap \mathit{Lit}(P)$;

Note that function $T_0(P)$ returns consequences of a program P and that $\mathit{comp}(P)$ may contain extra variables when the completion is converted to clauses, thus $\cap \mathit{Lit}(P)$ is needed for the return of the function.

Formally, this preceding procedure computes the least fixed point of the following operator.

$$f(X) = \mathit{UP}(\,\mathit{comp}(P) \cup X \cup \mathit{floop}_0(P, X)\,) \cap \mathit{Lit}(P). \tag{8}$$

As it turns out, this least fixed point is essentially the well-founded model when the given normal logic program has no constraints. This means (surprisingly perhaps) that

the well-founded models amount to repeatedly applying unit propagation to the program completion and loop formulas of loops that do not have any "applicable" external support rules. In general, for normal logic programs that may have constraints, the least fixed point of the preceding operator of Equation (9) is essentially $Expand(P,\emptyset)$ in smodels [Baral 2003; Simons et al. 2002]. More generally, given a set A of literals, $Expand(P,A)$ corresponds to the least fixed point of the following operator.

$$U_A^P(X) = UP(\, comp(P) \cup A \cup X \cup floop_0(P,X)\,) \cap Lit(P). \tag{9}$$

Clearly, the function T_0 can start from A; we only need to replace $X := \emptyset$ by $X := A$ in the procedure which computes the least fixed point of U_A^P.

Now if we add in $floop_1(P,A)$, a more powerful operator can be defined.

$$T_A^P(X) = UP(\, comp(P) \cup A \cup X \cup floop_0(P,X) \cup floop_1(P,X)\,) \cap Lit(P).$$

In the following, we denote by $T_1(P,A)$ the least fixed point of the operator T_A^P. Clearly, $U_A^P(X) \subseteq T_A^P(X)$, and the least fixed point of U_A^P, denoted by $T_0(P,A)$ (in particular, $T_0(P,\emptyset) = T_0(P)$), is contained in $T_1(P,A)$, the least fixed point of T_A^P. The following example shows that the containments can be proper.

Example 2. Consider the following logic program P.

$$\begin{aligned} x &\leftarrow not\ e.\\ e &\leftarrow not\ x.\\ n &\leftarrow x.\\ n &\leftarrow m.\\ m &\leftarrow n.\\ &\leftarrow not\ n. \end{aligned}$$

Clearly, $x \in T_1(P,\emptyset)$ but $x \notin T_0(P,\emptyset)$.

PROPOSITION 3.4. *Let P be a normal logic program and A a set of literals in P. If S is an answer set of P that satisfies A, then S also satisfies $T_1(P,A)$.*

Notice that $T_1(P,A)$, the least fixed point of the operator T_A^P, can be computed by an iterative procedure like the one described in the Introduction.

3.3. Expand in Smodels

We mentioned that the least fixed point of our operator U_A^P defined by Equation (9) coincides with the output of the *Expand* operator used in smodels. We now make this precise. Our following presentation of the *Expand* operator in smodels follows that of Baral [2003].

Given a normal logic program P and a set A of literals, the goal of $Expand(P,A)$ is to extend A as much as possible and as efficiently as possible so that all answer sets of P that agree with A also agree with $Expand(P,A)$. It is defined in terms of two functions named $Atleast(P,A)$ and $Atmost(P,A)$. They form the lower and upper bound of what can be derived from the program P based on A in the sense that those in $Atleast(P,A)$ must be in and those not in $Atmost(P,A)$ must be out. Formally, $Expand(P,A)$ is defined to be the least fixed point of the following operator.

$$E_A^P(X) = Atleast(P, A \cup X) \cup \overline{Atoms(P) \setminus Atmost(P, A \cup X)}. \tag{10}$$

The function $Atleast(P,A)$ is defined as the least fixed point of the operator F^P_A defined as follows.

$$\begin{aligned}
F_1(P,X) &= \{\, a \in Atoms(P) \mid \text{ there is a rule } r \text{ in } P \text{ such that } a = head(r) \text{ and} \\
&\quad X \models body(r) \,\}. \\
F_2(P,X) &= \{\, \neg a \mid a \in Atoms(P) \text{ and for all } r \in P, \text{ if } a = head(r), \text{ then} \\
&\quad X \models \neg body(r) \,\}. \\
F_3(P,X) &= \{\, x \mid \text{ there exists an atom } a \in X \text{ such that there is only one rule } r \\
&\quad \text{in } P \text{ such that } a = head(r), x \in body(r), \text{ and } X \not\models \neg body(r) \,\}. \\
F_4(P,X) &= \{\, \bar{x} \mid \text{ there exists } \neg a \in X \text{ such that there is a rule } r \text{ in } P \text{ such that} \\
&\quad a = head(r) \text{ and } X \cup \{x\} \models body(r) \,\}. \\
F_5(P,X) &= \begin{cases} Lit(P) & \text{if } X \text{ is inconsistent,} \\ \emptyset & \text{otherwise.} \end{cases} \\
F^P_A(X) &= A \cup X \cup F_1(P,X) \cup F_2(P,X) \cup F_3(P,X) \cup F_4(P,X) \cup F_5(P,X),
\end{aligned}$$

where $X \models body(r)$ if $body(r) \subseteq X$, and $X \models \neg body(r)$ if $\overline{X} \cap body(r) \neq \emptyset$.

The function $Atmost(P,A)$ is defined as the least fixed point of the following operator G^P_A.

$$\begin{aligned}
G^P_A(X) &= \{\, a \mid \text{there is a rule } r \in P \text{ such that } head(r) = a, X \setminus A^- \models body^+(r), \\
&\quad \text{and } body^-(r) \cap A^+ = \emptyset \,\} \setminus A^-,
\end{aligned}$$

where $A^+ = \{\, a \mid a \text{ is an atom and } a \in A \,\}$, and $A^- = \{\, a \mid a \text{ is an atom and } \neg a \in A \,\}$.

In the following, a normal program P is said to be *simplified* if, for any rule $r \in P$, $head(r) \notin body^+(r) \cup body^-(r)$. Notice that any normal logic program is strongly equivalent to a simplified normal logic program: if $head(r) \in body^+(r)$, then $\{r\}$ is strongly equivalent to the empty set and thus can be deleted safely from any logic program, and if $head(r) \in body^-(r)$, then $\{r\}$ is strongly equivalent to $\{\leftarrow body(r)\}$ (cf. [Lin and Chen 2007]).

The following theorem relates $T_0(P,A)$ and $Expand(P,A)$. The proof is given in the Appendix.

THEOREM 3.5. *For any normal logic program P and any set $A \subseteq Lit(P)$, $Expand(P,A) \subseteq T_0(P,A)$. If P is simplified, then $Expand(P,A) = T_0(P,A)$.*

As we mentioned, if P has no constraint, then $Expand(P,\emptyset)$ is the same as the well-founded model of P [Baral 2003]. Thus for simplified logic programs, the well-founded model can be computed by a bottom-up procedure using unit propagation on sets of clauses from the program completion and the loop formulas of the loops that do not have any external support rules.

The following example shows that if P has a rule such as $p \leftarrow not\, p$, T_0 may be stronger than $Expand$.

Example 3. Consider the following program P.

$$\begin{aligned}
p &\leftarrow not\, q. \\
q &\leftarrow not\, p. \\
f &\leftarrow not\, p. \\
f &\leftarrow not\, f.
\end{aligned}$$

Clearly, $T_0(P,\emptyset) = \{f, q, \neg p\}$, but $Expand(P,\emptyset) = \emptyset$.

3.4. Some Experiments

We have implemented a system[2] that for any given normal logic program P, it first computes $T_1(P, \emptyset)$ and then adds the following set of constraints, $\{ \leftarrow l \mid l \in T_1(P, \emptyset) \}$, to P. Clearly, adding these constraints to P does not change the answer sets. The effectiveness of this strategy obviously depends on the underlying ASP solver as well as whether the consequences computed by our system are new to the ASP solver.

For the normal logic programs used at the First Answer Set Programming System Competition[3], our system does not return anything beyond the well-founded model of P. Thus for these programs, our system does not add anything new. This does not mean that our system is not suitable for these benchmark problems. A typical logic program has variables, and the instances that are used to ground these variables are often important in determining the hardness of the grounded logic program. Niemelä's encoding of the Hamiltonian circuit (HC) problem is used in the competition as well, but the graphs used are all generated randomly. As we shall see next, when the graphs have some specific structures which occur in many practical problems, our system can speed up the current ASP solvers on these problems significantly.

For Niemelä's encoding of the HC problem [Niemelä 1999]. Instead of randomly generated graphs, we consider graphs that represent networks consisting of sets of components that are densely connected inside but have only a few connections among them. These networks are ubiquitous. Examples include countries consisting of big cities that are connected by only a few highways, cities consisting of populated neighborhoods that are connected by a few main roads, and circuits that are often composed of components that are highly connected inside but have only a few connections between them.

To simplify things a bit, we model these networks by graphs consisting of some complete subgraphs that are connected by a few arcs between them. Specifically, we consider graphs of the form MxN: a graph with N copies of the complete graph with M nodes, $C_1, \ldots, C_N$, and with exactly one arc from C_i to C_{i+1} and exactly one arc from C_{i+1} to C_i for each $1 \leq i \leq N$ (C_{N+1} is defined to be C_1).

Clearly, a graph of the form M x N has a Hamiltonian circuit, and each such circuit must go through the arcs connecting the complete subgraphs. Furthermore, all except for one of the "must-in" arcs can be computed by $T_1(P, \emptyset)$, thus adding their corresponding constraints to P should help ASP solvers in computing the answer sets. This is confirmed by our experiments.

Table I contains the running times for these Hamiltonian circuit programs.[4] For each M x N entry in the table, we randomly created 20 different such graphs (two M x N graphs may differ on which arcs are chosen to connect two neighboring complete subgraphs), and the times reported in the table refers to the average times for the resulting 20 programs. The programs are first grounded by gringo (version 3.0.1 [Gebser et al. 2007])[5], then computed by different ASP solvers. The numbers under "cmodels$_{T_1}$" and "clasp$_{T_1}$" refer to the runtimes (in seconds) of cmodels (version 3.79 [Giunchiglia et al. 2006]) and clasp (version 1.3.4 [Gebser et al. 2007]) when the results from $T_1(P, \emptyset)$ are added to the original program as constraints, and those under "T_1" are the runtimes of our program for computing $T_1(P, \emptyset)$. As can be seen, information from $T_1(P, \emptyset)$ makes both cmodels and clasp run much faster when looking for an answer set.

---

[2]Our implementation is available at `http://www.cs.ust.hk/cloop/`. The current implementation is significantly more efficient that the one reported in Chen et al. [2008].

[3]`http://asparagus.cs.uni-potsdam.de/contest/`

[4]Our experiments were done on a 4×AMD Opteron 844 (1.8 GHz) CPU and 8 GB RAM. The reported times are in CPU seconds, as reported by the Linux "/usr/bin/time" command.

[5]The result is similar when programs are grounded by Lparse (version 1.1.2) in `http://www.tcs.hut.fi/Software/smodels/`.

Table I. Runtime Data for Normal Logic Programs

Problem	cmodels	cmodels$_{T_1}$	clasp	clasp$_{T_1}$	T_1
10 x 10	9.93	0.70	0.07	0.05	0.29
10 x 20	18.24	2.50	0.24	0.13	0.63
10 x 30	38.66	5.16	0.57	0.25	1.03
10 x 40	85.15	7.98	1.05	0.38	1.52
10 x 50	73.31	14.12	2.10	0.48	1.97
15 x 10	586.57	5.42	0.46	0.19	1.02
15 x 20	>2h	12.62	1.24	0.52	2.26
15 x 30	>2h	25.07	6.46	0.68	3.64
15 x 40	>2h	45.24	100.80	1.58	5.47
15 x 50	>2h	72.09	234.00	2.33	7.14
20 x 10	2080.62	19.62	0.73	0.44	2.47
20 x 20	>2h	67.38	25.04	1.19	5.59
20 x 30	>2h	147.72	85.82	2.81	8.85
20 x 40	>2h	219.62	465.13	4.36	13.01
20 x 50	>2h	352.35	2940.28	6.20	17.36

In addition to cmodels and clasp, we also tried DLV [Leone et al. 2006] and smodels (version 2.34). DLV did not return within two hours (including grounding times) for most of the 10 x 2 graphs, but returned an answer set under 0.1 seconds when the results from $T_1(P, \emptyset)$ were added.

Our experiences with smodels are especially interesting and informative. Similar to DLV, smodels performed badly even on the smallest graphs that we tried. However, adding consequences from $T_1(P, \emptyset)$ did not help either. This is actually to be expected, as smodels apply the *Lookahead* operator, and whatever can be computed by $T_1(P, \emptyset)$ can also be computed by the *Lookahead* operator.[6] It is not easy to modify smodels by taking out the *Lookahead* operator, but clasp has an option to turn on an operator like this one, and we were surprised to find that once this option was turned on, clasp became much slower. For instance, it took an hour for it to return an answer set for most of the 20 x 30 graphs.

We can only hypothesize why smodels performed badly on these programs and why clasp performed almost just as badly when its lookahead option is turned on. Whatever the reasons, it is clear that an operator like lookahead should be applied selectively. A similar discussion is given in Liu and You [2007]. Our system computes $T_1(P, \emptyset)$ once as a preprocessing step, and the results help cmodels and clasp on logic programs that contain certain crucial must-in rules. It is an interesting question as to how to integrate this procedure into search by applying it when a search decision leads to a logic program that contains some crucial must-in rules. Of course, the difficult part is to figure out which program has this property.

We remark here that the crucial feature of the graphs that we consider here is that they consist of sets of subgraphs that are densely connected inside but with a single path linking them together. Whether the subgraphs are identical copies of a complete graph is not important. Furthermore, while our experiments were done on the Hamiltonian circuit problem, they should carry over to any logic programs whose positive dependency graphs have a similar structures. As we previously mentioned, these structures are ubiquitous in practical problems. When these problems are modeled by logic programs, the dependency graphs of these programs will reflect the structures as well.

Finally, in retrospect, it is not surprising that the consequences computed and added to the input program speeds up the current ASP solvers. These new consequences

[6]Literals in $T_1(P, \emptyset)$ are consequences of P; clearly, they can be computed from *Lookahead*.

are from loops with exactly one external support rule. These external support rules are crucial, as they must be used in computing an answer set. In a sense, these new consequences are like "backbones" in SAT. Once they are added to the input, a big search space is pruned.

4. COMPUTING LOOPS WITH AT MOST ONE EXTERNAL SUPPORT FOR DISJUNCTIVE LOGIC PROGRAMS

In the previous section, we show that for normal logic programs, loop formulas of loops with at most one external support rule can be computed in polynomial time and that an iterative procedure based on these formulas, the program completion, and unit propagation compute consequences of the program which can help ASP solvers to find answer sets of certain logic programs.

In this section, we tend to extend these results to disjunctive logic programs. We consider first whether these loops can be computed in disjunctive logic programs.

4.1. Loops with No External Support

It is easy to see that if a loop L has no external support rules, that is, $R^-(L) = \emptyset$, then its loop formula (Equation (6)) is equivalent to $\bigwedge_{p\in L} \neg p$; if L has only one external support rule, that is, $R^-(L) = \{r\}$, then its loop formula (Equation (6)) is equivalent to the following.

$$\bigwedge_{p\in L} \neg p \vee \left(body(r) \wedge \bigwedge_{q\in head(r)\setminus L} \neg q \right),$$

which will be equivalent to a set of binary clauses.

More generally, if we already know that A is a set of literals that is true in all answer sets, then for any loop L that has no external support rules whose body is active under A with respect to L, its loop formula is still equivalent to a set of literals under A. A rule r is active under A with respect to L if $\overline{A} \cap body(r) = \emptyset$ and $A \cap (head(r) \setminus L) = \emptyset$.

Thus we extend the notion of external support rules and have it conditioned on a given set of literals. Let P be a disjunctive logic program and A a set of literals. We say that a rule r is an *external support rule of L under A* if $r \in R^-(L)$ is active under A with respect to L. In the following, we denote by $R^-(L,A)$ the set of external support rules of L under A.

Given a disjunctive logic program P and a set A of literals—similar to normal logic programs—we define the following.

$$\begin{aligned} loop_0(P,A) &= \{\, L \mid L \text{ is a loop of } P \text{ such that } R^-(L,A) = \emptyset \,\}, \\ floop_0(P,A) &= \{\, \neg a \mid a \in L \text{ for a loop } L \in loop_0(P,A) \,\}. \end{aligned}$$

We now consider how to compute $floop_0(P,A)$ for disjunctive logic programs. It is shown that $floop_0(P,A)$ can be computed in quadratic time for normal logic programs. However, for disjunctive logic programs, the problem is NP-hard in the general case.

PROPOSITION 4.1. *Given a disjunctive logic program P, a set A of literals, and an atom a, decide whether $\neg a \in floop_0(P,A)$ is NP-complete.*

Fortunately, if the set A is unfounded-free[7], then $floop_0(P,A)$ can be computed in quadratic time. As we shall see, this restriction is enough for computing consequences of a logic program using the procedure outlined in the Introduction when the inference rule is unit propagation and the class of loops is that of loops without external support.

[7]Recall that A is unfounded-free if $A \cap X = \emptyset$ for each unfounded set X of P with respect to A.

Our following algorithm for computing $floop_0(P,A)$ is similar to the corresponding one for normal programs and is through maximal loops.

For a disjunctive logic program P, let $ml_0(P,A)$ be the set of maximal loops that does not have any external support rules under A. Clearly,

$$floop_0(P,A) = \bigcup_{L \in ml_0(P,A)} \overline{L}.$$

If P is a normal logic program, loops in $ml_0(P,A)$ are pairwise disjoint. For disjunctive logic programs, the property is not true in general; this is the reason that $floop_0(P,A)$ is intractable. However, if A is unfounded-free, then loops in $ml_0(P,A)$ are pairwise disjoint. This follows from the following proposition.

PROPOSITION 4.2. *Let P be a disjunctive logic program and A be a set of literals such that $A \cap (L_1 \cup L_2) = \emptyset$. If L_1 and L_2 are two loops of P that do not have any external support rules under A, and $L_1 \cap L_2 \neq \emptyset$, then $L_1 \cup L_2$ is also a loop of P that does not have any external support rules under A.*

The following example shows that the condition $A \cap (L_1 \cup L_2) = \emptyset$ in Proposition 4.2 is necessary.

Example 4. Consider the following disjunctive logic program P.

$$\begin{aligned} a \vee b \vee c &\leftarrow . \\ a &\leftarrow b,\ c. \\ b &\leftarrow a. \\ c &\leftarrow a. \end{aligned}$$

Let $A = \{b,c\}$, $L_1 = \{a,b\}$, and $L_2 = \{a,c\}$. L_1 and L_2 belong to $loop_0(P,A)$, but $L_1 \cup L_2 = \{a,b,c\}$ is a loop of P that has one external support under A.

Now consider the following algorithm.

$ML_0(P,A) := ML_0(P,A,Atoms(P))$;
Function $ML_0(P,A,S)$: P a disjunctive program, A and S sets of literals of P
  $ML := \emptyset$; $G :=$ the S induced subgraph of G_P;
  For each strongly connected component L of G:
    if $R^-(L,A) = \emptyset$ **then** add L to ML;
    else append $ML_0(P,A,L \setminus \bigcup_{r \in R^-(L,A)} H(r,A))$ to ML;
  return ML;

where G_P is the positive dependency graph of P, and

$$H(r,A) = \begin{cases} head(r) & \text{if } head(r) \cap A = \emptyset, \\ head(r) \cap A & \text{if } head(r) \cap A \neq \emptyset. \end{cases}$$

Specially, $ML_0(P,A)$ is a short term for $ML_0(P,A,Atoms(P))$. Note that if P is a normal logic program, then the algorithm presented here is the same as the corresponding algorithm for normal logic programs.

Similarly, we have the following theorem.

THEOREM 4.3. *Let P be a disjunctive logic program and A and S be sets of literals in P.*

(1) The function $ML_0(P,A,S)$ runs in $O(n^2)$, where n is the size of P as a set.
(2) $ML_0(P,A) \subseteq loop_0(P,A)$.
(3) If A is unfounded-free, then $ML_0(P,A) = ml_0(P,A)$.

4.2. Loops with at Most One External Support

Similarly, for disjunctive logic programs, we can consider the set of loops that has exactly one external support rule under a set A of literals and the set of loop formulas of these loops.

$$\begin{aligned} loop_1(P,A) &= \{\, L \mid L \text{ is a loop of } P \text{ such that } R^-(L,A) = \{r\}, \text{ for only one rule } r \in P \,\}, \\ floop_1(P,A) &= \{\, \neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L}, \text{ where } L \in loop_1(P,A) \text{ and } r \text{ is the only external support rule of } L \text{ under } A \,\}. \end{aligned}$$

Like $floop_0(P,A)$, $floop_1(P,A)$ can be computed in polynomial time for normal logic programs, but for disjunctive logic programs, it is intractable.

PROPOSITION 4.4. *Given a disjunctive logic program P, a set A of literals, an atom a, and a literal l, decide whether $\neg a \vee l \in floop_1(P,A)$ is NP-complete.*

While there is a polynomial algorithm for computing $floop_0(P,A)$ when A is unfounded-free, this is not the case for $floop_1(P,A)$. Proposition 4.4 holds even when we restrict A to be unfounded-free.

Notice that for normal logic programs, the complexity of $floop_1(P,A)$ is left as an open question. Instead, a polynomial algorithm is proposed for computing $floop_0(P,A) \cup floop_1(P,A)$[8], which corresponds to the set of loop formulas of loops with at most one external support. For disjunctive logic programs, $floop_0(P,A) \cup floop_1(P,A)$ is still intractable even when A is unfounded-free.[9]

Given these negative results about computing loop formulas of loops with at most one external support in disjunctive logic programs, we turn our attention to polynomial algorithms that can compute as many loop formulas from $floop_0(P,A) \cup floop_1(P,A)$ as possible. We propose one such approximation algorithm next. It is based on the observation that if a loop has one external support rule, then it often has no external support when this rule is deleted. This would reduce the problem of computing loops with one external support rule to that of loops with no external support, and for the latter, we can use the function $\mathit{ML}_0(P,A,S)$ when A is unfounded-free (Theorem 4.3).

PROPOSITION 4.5. *For any disjunctive logic program P and a set A of literals that is unfounded-free for P, $floop_0(P,A)$ and $floop_1(P,A)$ imply the following theory.*

$$\bigcup_{\overline{A} \cap body(r) = \emptyset,\, L \in \mathit{ML}_0(P \setminus \{r\}, A)} \{\, \neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L} \,\}. \tag{11}$$

In the following, we use $\mathit{FLoop}_1(P,A)$ to denote Equation (11). Note that if P is a normal logic program, then $floop_0(P,A) \cup \mathit{FLoop}_1(P,A)$ is equivalent to $floop_0(P,A) \cup (7)$. According to Proposition 3.3, if P is a normal logic program, then $floop_0(P,A) \cup floop_1(P,A)$ is equivalent to $floop_0(P,A) \cup \mathit{FLoop}_1(P,A)$ for any A. However, for disjunctive logic programs, these two formulas are not equivalent even when A is unfounded-free, as the following example illustrates.

[8]Not exactly this set, but $floop_0(P,A) \cup (7)$, which is logically equivalent to $floop_0(P,A) \cup floop_1(P,A)$, and especially $UP(floop_0(P,A) \cup (7)) = UP(floop_0(P,A) \cup floop_1(P,A))$.

[9]For disjunctive logic programs, deciding whether a literal $l \in UP(floop_0(P,A) \cup floop_1(P,A))$ where A is unfounded-free is NP-hard. Note that we can use the translation proposed in the proof of Proposition 4.4 to reduce the SAT problem to it.

Example 4.5. Consider the following logic program P.

$$\begin{aligned} a \vee b \vee c &\leftarrow d. \\ a &\leftarrow b, c. \\ b &\leftarrow a. \\ c &\leftarrow b. \end{aligned}$$

Let $A = \emptyset$, $loop_1(P,A) = \{\{a,b,c\},\{a,b\}\}$, both loops have one external support rule: $a \vee b \vee c \leftarrow d$, thus $\neg a \vee \neg c, \neg b \vee \neg c \in floop_1(P,A)$, but they can not be computed from $FLoop_1(P,A)$.[10]

So to summarize, for disjunctive logic programs, while we can not efficiently compute $floop_0(P,A) \cup floop_1(P,A)$, we can compute $floop_0(P,A) \cup FLoop_1(P,A)$, which is still helpful for computing consequences of a logic program. To compute $floop_0(P,A) \cup FLoop_1(P,A)$, we first call $ML_0(P,A)$, and then for each proper rule $r \in P$ such that $\overline{A} \cap body(r) = \emptyset$, we call $ML_0(P \setminus \{r\},A)$. The worse case complexity of this procedure is $O(n^3)$, where n is the size of P.

4.3. Computing Consequences of a Disjunctive Logic Program

Let's now consider computing consequences of a disjunctive logic program using the loop formulas computed in the last section.

Consider the iterative procedure given in the Introduction. When $\vdash_P$ is unit propagation UP and the loop formulas are those from ML_0 (for disjunctive logic programs), it tends to be the same as the function T_0 presented in Section 3.

Clearly, $T_0(P)$ runs in polynomial time and returns a set of consequences of a disjunctive logic program P. It is easy to see that at each iteration, the set X computed by the procedure is also a set of consequences of P. Thus by the following proposition and Theorem 4.3, if P has at least one answer set, then at each iteration, the set of literals added to the set of loop formulas of loops in $ML_0(P,X)$ equals $floop_0(P,X)$, that is, the set of loop formulas with no external support under X.

PROPOSITION 4.6. *Let P be a disjunctive logic program that has an answer set. If A is a set of literals that are consequences of P, then A is unfounded-free for P.*

Similarly, using $floop_0(P,A) \cup floop_1(P,A)$, we get the following procedure.

Function $T^1(P)$: P is a disjunctive logic program
  $X := \emptyset$; $Y := comp(P) \cup floop_0(P,\emptyset) \cup floop_1(P,\emptyset)$;
  while $X \neq UP(Y)$ **do**
    $X := UP(Y)$; $Y := Y \cup floop_0(P,X) \cup floop_1(P,X)$;
  return $X \cap Lit(P)$;

Again, it is easy to see that at each iteration, X is a set of consequences of P, and in particular, $T^1(P)$ returns a set of consequences of P. For normal logic programs, by Proposition 3.3, $floop_0(P,A) \cup floop_1(P,A)$ can be computed in polynomial time, thus $T^1(P)$ runs in polynomial time. For disjunctive logic programs, as we have shown in the previous section, computing $floop_0(P,A) \cup floop_1(P,A)$ is intractable, even for unfounded-free A. Thus we cannot show that the preceding procedure is polynomial. However, this still leaves open the question of whether $T^1(P)$ can be computed by some other methods that hopefully can be shown to run in polynomial time. Unfortunately, this does not seem to be likely, as we can show that computing $T^1(P)$ is also intractable.

[10]Note that let $r = a \vee b \vee c \leftarrow d.$, $ML_0(P \setminus \{r\},A) = \{\{a,b,c\}\}$, and $\{a,b\}$ is not a maximal loop without external support rules in $P \setminus \{r\}$.

PROPOSITION 4.7. *For any disjunctive logic program P, decide whether a literal in $T^1(P)$ is NP-hard.*

In the last section, we propose using $FLoop_1(P,A)$ as a polynomial approximation of $floop_1(P,A)$. We can thus make use of this operator.

Function $T_1(P)$: P is a disjunctive logic program
  $X := \emptyset$; $Y := comp(P) \cup floop_0(P,\emptyset) \cup FLoop_1(P,\emptyset)$;
  while $X \neq UP(Y)$ **do**
    $X := UP(Y)$; $Y := Y \cup floop_0(P,X) \cup FLoop_1(P,X)$;
  return $X \cap Lit(P)$;

This is the function that we have implemented and used in our experiments. See Section 4.5 for details.

4.4. Related Work

Here we relate $T_0(P)$ for disjunctive program P to the preprocessing procedure in DLV [Leone et al. 2006] and the well-founded semantics of disjunctive programs proposed by Wang and Zhou [2005].

4.4.1. DLV Preprocessing Operator. We now show that $T_0(P)$ coincides with the least fixed point of the operator $\mathcal{W}_P$ used in DLV for preprocessing a given disjunctive logic program. First we show that the greatest unfounded set of a disjunctive logic program (if it exists) can be computed from loop formulas of loops that have no external support rules.

Given a disjunctive logic program P and A a set of literals, we use $M(P,A)$ to denote the least fixed point of the operator M_P^A, defined as follows.

$$\begin{aligned} loop_0^A(P,X) &= \{\, a \mid \text{ there is a loop } L \text{ of } P \text{ such that } a \in L \text{ and } R^-(L, A \cup \overline{X}) = \emptyset \,\}, \\ F_2^A(P,X) &= \{\, a \mid a \in Atoms(P) \text{ and for all } r \in P, \text{ if } a \in head(r) \text{ then} \\ &\qquad \overline{A} \cap body(r) \neq \emptyset, X \cap body(r) \neq \emptyset, \text{ or } (head(r) \setminus \{a\}) \cap A \neq \emptyset \,\}, \\ M_P^A(X) &= X \cup loop_0^A(P,X) \cup F_2^A(P,X). \end{aligned}$$

THEOREM 4.8. *For any disjunctive logic program P and any $A \subseteq Lit(P)$ such that the greatest unfounded set of P with respect to A exists, $M(P,A) = GUS_P(A)$.*

From this theorem, we can compute $GUS_P(A)$ by $M(P,A)$. We do not yet know any efficient way of computing $loop_0(P,A)$ for any possible A, but if A is restricted to be unfounded-free, then $GUS_P(A)$ always exists, and $loop_0^A(P,X) = \bigcup_{L \in ML_0(P, A \cup \overline{X})} L$, which can be computed in polynomial time. Furthermore, $F_2^A(P,X)$ can be computed in linear time. So, if A is unfounded-free, we have proposed a loop-oriented approach for computing $GUS_P(A)$ in polynomial time. Note that different from other current approaches, $GUS_P(A)$ is computed directly here, thus avoiding the computation of its complement.

Now we introduce the $\mathcal{W}_P$ operator proposed in Leone et al. [1997].

$$\begin{aligned} \mathcal{T}_P(X) &= \{\, a \in Atoms(P) \mid \text{ there is a rule } r \in P \text{ such that } a \in head(r), \\ &\qquad head(r) \setminus \{a\} \subseteq \overline{X}, \text{ and } body(r) \subseteq X \,\}, \\ \mathcal{W}_P(X) &= \mathcal{T}_P(X) \cup \overline{GUS_P(X)}. \end{aligned}$$

From Proposition 5.6 in Leone et al. [1997], $\mathcal{W}_P$ has a least fixed point, denoted $\mathcal{W}_P^\omega(\emptyset)$, which is the consequence of the program. $\mathcal{W}_P^\omega(\emptyset)$ can also be computed

efficiently, thus it is considered as a good start point from which compute answer sets and is implemented in DLV.

In the following, a disjunctive logic program P is said to be simplified if for any $r \in P$, $head(r) \cap (body^+(r) \cup body^-(r)) = \emptyset$. Notice that any disjunctive logic program is strongly equivalent to a simplified program: if $head(r) \cap body^+(r) \neq \emptyset$, then $\{r\}$ is strongly equivalent to the empty set and thus can be safely deleted from any logic program, and if $head(r) \cap body^-(r) \neq \emptyset$, then $\{r\}$ is strongly equivalent to $\{r'\}$ such that $head(r') = head(r) \setminus body^-(r)$ and $body(r') = body(r)$ (cf. [Lin and Chen 2007]). Clearly, if P is a normal program, the notion of simplified defined here coincides with that of the previous section.

The following theorem relates $T_0(P)$ and $\mathcal{W}_P^\omega(\emptyset)$.

THEOREM 4.9. *For any disjunctive logic program P, $\mathcal{W}_P^\omega(\emptyset) \subseteq T_0(P)$. If P is simplified and without constraints, then $\mathcal{W}_P^\omega(\emptyset) = T_0(P)$.*

Note that Leone et al. [1997] proved that if P does not contain constraints, $\mathcal{W}_P^\omega(\emptyset)$ coincides with the well-founded model of a normal logic program P' obtained by "shifting" some head atoms to the bodies of the rules. Thus, if P is simplified and without constraints, then $T_0(P)$ coincides with the well-founded model of P' as well.

Given a disjunctive logic program P, we denote by $sh(P)$ the normal program obtained from P by substituting every rule of Equation (2) by the following k rules.

$$a_i \leftarrow a_{k+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n, not\, a_1, \ldots, not\, a_{i-1}, not\, a_{i+1}, \ldots, not\, a_k. \quad (1 \leq i \leq k)$$

It is worth noting that $FLoop_1(sh(P), A)$ may be not a consequence of a disjunctive logic program, even when A is unfounded-free for P or $sh(P)$.

Example 4.6. Consider the following logic program P.

$$\begin{aligned} d &\leftarrow not\, e. \\ e &\leftarrow not\, d. \\ a \vee c &\leftarrow d. \\ a \vee b &\leftarrow e. \\ a &\leftarrow b. \\ b &\leftarrow a. \\ &\leftarrow not\, a. \\ &\leftarrow not\, b. \end{aligned}$$

Clearly, $\{a, b, d\}$ and $\{a, b, e\}$ are the only two answer sets of P, and $\{a, b, d\}$ is the only answer set of $sh(P)$. Let $A = \{a, b\}$, A is unfounded-free for P and $sh(P)$. $\neg a \vee d \in FLoop_1(sh(P), A)$ is false for $\{a, b, e\}$ and thus not a consequence of P.

A disjunctive logic program P is *head-cycle free* if there does not exist a loop L and a rule r such that $a, b \in L$ and $a, b \in head(r)$. If P is head-cycle free, then a set of atoms is an answer of P if and only if it is an answer set of $sh(P)$.

PROPOSITION 4.10. *For any head-cycle free disjunctive logic program P and a set A of literals,*

$$ml_0(P, A) = ML_0(P, A) = ML_0(sh(P), A),$$

$$FLoop_1(P, A) = \bigcup_{\substack{\overline{A} \cap body(r) = \emptyset, \\ L \in ML_0(sh(P \setminus \{r\}), A)}} \{ \neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L} \},$$

and $floop_0(P, A)$ implies that $floop_1(P, A)$ is equivalent to $FLoop_1(P, A)$.

4.4.2. Wang and Zhou's Well-Founded Semantics for Disjunctive Logic Programs. Theorem 3.5 states that T_0 computes the well-founded model when the given normal logic program is simplified and has no constraints. However, there have been several competing proposals for extending the well-founded semantics to disjunctive logic programs [Brass and Dix 1999; Leone et al. 1997; Wang and Zhou 2005]. It is interesting that with a slight change of unit propagation, the procedure computes the same results as the well-founded semantics proposed in Wang and Zhou [2005]. We now make this precise; first, we give one of the definitions of the well-founded semantics proposed by Wang and Zhou.

Given a disjunctive logic program P, a *positive* (*negative*) disjunction is a disjunction of atoms (negative literals) of P. A *pure disjunction* is either a positive one or a negative one. If A and $B = A \vee A'$ are two disjunctions, then we say A is a *subdisjunction* of B, denoted $A \subseteq B$. Let S be a set of pure disjunctions; we say that the $body(r)$ of $r \in P$ is true with respect to S, denoted $S \models body(r)$, if $body(r) \subseteq S$; $body(r)$ is false with respect to S, denoted $S \models \neg body(r)$, if either (1) the complement of a literal in $body(r)$ is in S or (2) there is a disjunction $a_1 \vee \cdots \vee a_n \in S$ such that $\{ not\, a_1, \ldots, not\, a_n \} \subseteq body(r)$.

Now we define the notion of an unfounded set under a set of pure disjunctions. Let S be a set of pure disjunctions of a disjunctive logic program P, then a set of atoms X is an *unfounded set* for P with respect to S if for each $a \in X$, $r \in P$ such that $a \in head(r)$, at least one of the following conditions holds.

(1) The body of r is false with respect to S.
(2) There is $x \in X$ such that $x \in body^+(r)$.
(3) If $S \models body(r)$, then $S \models (head(r) – X)$. Here $(head(r) – X)$ is the disjunction obtained from $head(r)$ by removing all atoms in X, and $S \models (head(r) - X)$ means there is a subdisjunction $A' \subseteq (head(r) - X)$ such that $A' \in S$.

Note that if S is just a set of literals, then the preceding definition is equivalent to the definition in Section 2. If P has the greatest unfounded set with respect to S, we denote it by $\mathcal{U}_P(S)$. However, $\mathcal{U}_P(S)$ may be unfounded for some S.

Now we are ready to define the well-founded operator $\mathcal{W}'_P$ for any disjunctive logic program P and set of pure disjunctions S.

$$\begin{aligned} \mathcal{T}'_P(S) &= \{ A \text{ a pure disjunction} \mid \text{there is } r \in P : A \vee a_1 \vee \cdots \vee a_k \leftarrow body(r), \\ &\qquad \text{such that } S \models body(r) \text{ and } not\, a_1, \ldots, not\, a_k \in S \}, \\ \mathcal{W}'_P(S) &= \mathcal{T}'_P(S) \cup \overline{\mathcal{U}_P(S)}. \end{aligned}$$

Note that $\mathcal{T}'_P(S)$ is a set of positive disjunctions rather than a set of atoms.

From Wang and Zhou [2005], the operator $\mathcal{W}'_P$ always has the least fixed point, which is denoted by $lfp(\mathcal{W}'_P)$, and the well-founded semantics *U-WFS* is defined as $U\text{-}WFS(P) = lfp(\mathcal{W}'_P)$.

Now we extend T_0 to treat pure disjunctions. First, we extend the notion $floop_0(P, A)$ to under a set of pure disjunctions S.

A rule r is *active* under S with respect to a loop L, if $S \not\models \neg body(r)$ and $S \not\models (head(r) \setminus L)$. A rule r is an *external support rule* of L under S, if $r \in R^-(L)$ is active under S with respect to L. We use $R^-(L, S)$ to denote the set of external support rules of L under S.

Given a disjunctive logic program P and a set S of pure disjunctions, let

$$floop_0(P, S) = \{ \neg a \mid a \in L \text{ for a loop } L \text{ of } P \text{ such that } R^-(L, S) = \emptyset \}.$$

Then $floop_0(P, S)$ is equivalent to the set of loop formulas of the loops that do not have any external support rules under S. Clearly, if S is just a set of literals, the preceding definition of $floop_0$ is equivalent to the definition in Section 4.

Now we extend unit propagation to return pure disjunctions. Given a set Γ of clauses, we use UP^* to denote the set of pure disjunctions returned by the following extended unit propagation.

Function $UP^*(\Gamma)$
  if $(\emptyset \in \Gamma)$ **then return** Lit;
  $S := pure\_clause(\Gamma)$;
  if S is inconsistent **then return** Lit;
  if $S \neq \emptyset$ **then return** $S \cup UP^*(assign(S, \Gamma))$ **else return** $\emptyset$;

where $pure\_clause(\Gamma)$ returns the union of all positive clauses (disjunctions) and negative literals in Γ, A is the union of all unit clauses in S[11], then $assign(S, \Gamma)$ is $\{ c \mid \text{for some } c' \in \Gamma, c' \cap A = \emptyset, \text{ and } c = c' \setminus \overline{A} \}$.

We use the new unit propagation in T_0; formally, the procedure computes the least fixed point of the following operator.

$$T_0^*(P, S) = UP^*(comp(P) \cup S \cup floop_0(P, S)) \cap DB(P),$$

where $DB(P)$ denotes the set of pure disjunctions formed by the literals in $Lit(P)$. We use $T_0^*(P)$ to denote such a least fixed point.

The following theorem relates $U\text{-}WFS(P)$ and $T_0^*(P)$.

THEOREM 4.11. *For any disjunctive logic program P, $U\text{-}WFS(P) \subseteq T_0^*(P)$. If P is simplified and without constraints, then $U\text{-}WFS(P) = T_0^*(P)$.*

4.5. Some Experiments

We have also implemented a program that, for any given disjunctive logic program P, first computes $T_1(P)$ and then adds $\{ \leftarrow \bar{l} \mid l \in T_1(P) \}$ to P. Our implementation is also available on the Web.[12]

We tried our program on a number of benchmarks. First, for the disjunctive logic programs at the First Answer Set Programming System Competition, $T_1(P)$ does not return anything beyond the well-founded model of P. Next we tried the disjunctive encoding of the Hamiltonian circuit problem[13], and consider graphs with the same structure proposed in Section 3.4. Similarly, none of these must-in arcs can be computed; using the $\mathcal{W}_P$ operator, except one of them. Others can be computed from $T_1(P)$; thus adding the corresponding constraints to P should help ASP solvers in computing the answer sets.

Table II contains the running times for these programs.[14] Similar to the experiments described in Section 3.4, we also randomly created 20 different graphs for each M x N, and the times reported in the table refers to the average times for these corresponding 20 problems. The problems are first grounded by gringo (while Lparse will exhaust RAM before it grounds a 20 x 20 problem), then computed by different ASP solvers. In this table, the numbers under "cmodels$_{T_1}$" and "claspD$_{T_1}$" refer to the runtimes (in seconds) of cmodels (version 3.79) and claspD (version 1.1 [Drescher et al. 2008]) when the results from $T_1(P)$ are added to the original program as constraints. As can be seen, information from $T_1(P)$ makes cmodels and claspD run much faster when looking for an answer set. We also tried DLV, which would not be terminated in 2 hours (including

[11]Note that A is a set of literals.
[12]`http://www.cs.ust.hk/cloop/`
[13]`http://www.dbai.tuwien.ac.at/proj/dlv/examples/hamcycle`
[14]Our experiments were done on the same computer described in Section 3.4.

Table II. Runtime Data for Disjunctive Logic Programs

Problem	cmodels	cmodels$_{T_1}$	claspD	claspD$_{T_1}$	T_1
10 x 10	1194.90[1]	2.44	0.14	0.12	0.38
10 x 20	1318.68[1]	5.48	0.61	0.35	1.23
10 x 30	580.491	16.01	2.63	0.85	2.53
10 x 40	2318.57[1]	32.27	8.42	2.64	4.53
10 x 50	1864.36[1]	57.06	13.40	4.55	6.59
15 x 10	>2h	9.86	0.66	0.54	1.76
15 x 20	>2h	33.67	4.56	1.56	6.04
15 x 30	>2h	60.76	14.99	3.41	12.81
15 x 40	>2h	122.00	40.02	9.15	22.56
15 x 50	>2h	211.29	96.31	17.13	35.08
20 x 10	>2h	16.03	2.04	1.54	5.20
20 x 20	>2h	75.38	13.68	4.49	19.21
20 x 30	>2h	189.10	58.17	8.86	39.19
20 x 40	>2h	416.63	120.32	22.03	73.75
20 x 50	>2h	728.85	251.15	36.86	136.77

[1]In the program does not terminate in 2 hours for some instances, we count their time as 2 hours.

grounding times) for most 20 x 5 and 20 x 10 problems, but with information from $T_1(P)$, it will compute an answer for these problems in less than 10 and 50 seconds separately.

5. CONCLUSION

In this article, we consider loops that have at most one external support rule for normal and disjunctive logic programs. These loops are special in that their loop formulas are equivalent to sets of unit or binary clauses. We have considered how they, together with the program completion, can be used to deduce useful consequences of a logic program under unit propagation.

Our main results are that, for normal logic programs, the set of loop formulas of loops with at most one external support rule under a set of literals can be computed in polynomial time, and an iterative procedure using these loop formulas, program completion, and unit propagation is proposed for computing consequences of the program. When restricted to loop formulas of loops with no external support rules, the procedure basically computes the well-founded model. We have implemented this procedure as a preprocessing step and show experimentally that for some interesting Hamiltonian circuit problems, this preprocessing step significantly improves the performances of cmodels and clasp, which are among the best ASP solvers today.

For disjunctive logic programs, the set of loop formulas of loops that do not have any external support under an unfounded-free set of literals can be computed in polynomial time, and an iterative procedure using these loop formulas, program completion, and unit propagation outputs the same set of consequences as computed by the preprocessing step of DLV and is basically the same as Wang and Zhou's well-founded model semantics of disjunctive logic programs [2005]. However, the problem of computing loop formulas of loops with at most one external support is intractable. As a result, we consider a polynomial time algorithm for computing some of these loop formulas, and our experimental results show that this algorithm is sometimes useful for simplifying a disjunctive logic program beyond that which can be done by the preprocessing step of DLV.

The following are two interesting directions for future work.

— We have used unit propagation as the inference rule. One could use others as long as they are efficient enough. For example, in addition to unit propagation, one could consider adding the following rule: infer l from $l \vee a$ and $l \vee \neg a$, or even the full resolution on binary clauses.
— We have used these loop formulas for deriving consequences of a logic program. They can of course be used in ASP solvers such as ASSAT and cmodels directly. Whether this has any benefit requires further study.

APPENDIX: PROOFS OF THEOREMS AND PROPOSITIONS

A.1. Proof of Proposition 3.3

PROOF (PROPOSITION 3.3). Suppose $\mathit{floop}_0(P,A)$ and $\mathit{floop}_1(P,A)$ are true. We show that for any proper rule r of P such that $\overline{A} \cap \mathit{body}(r) = \emptyset$, and any $L \in \mathit{ml}_0(P \setminus \{r\}, A)$,

$$\{\neg a \vee l \mid a \in L, l \in \mathit{body}(r)\} \tag{12}$$

is true. First, L is also a loop of P; second, either $R^-(L,A) = \emptyset$ or $R^-(L,A) = \{r\}$. For the first case, $\overline{L} \subseteq \mathit{floop}_0(P,A)$, then $\mathit{floop}_0(P,A)$ implies $\neg a$ for each $a \in L$, thus Equation (12) is true. For the second case, Equation (12) is contained in $\mathit{floop}_1(P,A)$ and thus true.

Now suppose $\mathit{floop}_0(P,A)$ and Equation (7) are true. We show that $\mathit{floop}_1(P,A)$ is true, that is, for any proper rule $r \in P$ such that $\overline{A} \cap \mathit{body}(r) = \emptyset$ and any $L \in \mathit{ml}_1(P,A,r)$, Equation (12) holds. First, L is a loop of $P \setminus \{r\}$ that has no external support rules under A. Thus there exists $L' \in \mathit{ml}_0(P \setminus \{r\}, A)$ such that $L \subseteq L'$. Now L' is also a loop of P, and with respect to P, either $R^-(L',A) = \emptyset$ or $R^-(L',A) = \{r\}$. In the first case, $\overline{L} \subseteq \overline{L'} \subseteq \mathit{floop}_0(P,A)$, thus Equation (12) holds. In the second case, L' must be in $\mathit{ml}_1(P,A,r)$ as well, thus $L = L'$ and Equation (12) holds. □

A.2. Proof of Proposition 3.4

PROOF (PROPOSITION 3.4). According to Theorem 1 in Lin and Zhao [2004], S is an answer set of P if and only if it satisfies $\mathit{comp}(P)$ and each loop formula of P. Given a set X of literals, if X is a set of consequences of $P \cup \{\leftarrow p \mid \neg p \in A\} \cup \{\leftarrow \mathit{not}\, p \mid p \in A\}$, then $T_A^P(X)$ is a set of consequences of this program, thus $T_1(P,A)$ is also a set of consequences. □

A.3. Proof of Theorem 3.5

LEMMA A.1. *For any normal logic program P and any $A \subseteq \mathit{Lit}(P)$*[15]*,*

$$M(P,A) = \overline{\mathit{Atoms}(P) \setminus \mathit{Atmost}(P,A)}.$$

PROOF. Let $AM(P,A) = \overline{\mathit{Atoms}(P) \setminus \mathit{Atmost}(P,A)}$ and $X \subseteq AM(P,A)$; clearly, $\overline{X} \cap \mathit{Atmost}(P,A) = \emptyset$. First we prove that $M(P,A) \subseteq AM(P,A)$. As $M(P,A)$ is the least fixed point of our operator M_P^A, we only need to prove that $M_P^A(X) \subseteq AM(P,A)$.

Let L be a loop of P and $R^-(L, A \cup X) = \emptyset$; clearly, $L \cap \mathit{Atmost}(P, A \cup X) = \emptyset$. For any set of atoms Y such that $Y \subseteq \mathit{Atmost}(P,A)$ and $Y \subseteq \mathit{Atmost}(P, A \cup X)$, as $\overline{X} \cap \mathit{Atmost}(P,A) = \emptyset$, $Y \setminus A^- = Y \setminus (A^- \cup \overline{X})$, so $G_A^P(Y) = G_{A \cup X}^P(Y)$, thus $\mathit{Atmost}(P, A \cup X) = \mathit{Atmost}(P,A)$. Now we have $L \cap \mathit{Atmost}(P,A) = \emptyset$, then $\overline{\mathit{loop}_0^A(P,X)} \cap \mathit{Atmost}(P,A) = \emptyset$ and $\mathit{loop}_0^A(P,X) \subseteq AM(P,A)$.

[15] $M(P,A)$ is defined in Section 4.4.1.

Let a be an atom such that $a \in \overline{F_2^A(P,X)}$, then for all $r \in P$, if $a = head(r)$ then $A \cup X \models \neg body(r)$. Clearly, $a \notin Atmost(P, A \cup X)$ and $a \notin Atmost(P,A)$, thus $F_2^A(P,X) \subseteq AM(P,A)$.

So $M_P^A(X) \subseteq AM(P,A)$ and $M(P,A) \subseteq AM(P,A)$. Now we prove that $AM(P,A) \subseteq M(P,A)$. Let $S = AM(P,A) \setminus M(P,A)$, we want to prove $S = \emptyset$. The sketch of the proof is the following: if $S \neq \emptyset$, then there exists a loop $L \subseteq \overline{S}$ and $R^-(L, M(P,A)) = \emptyset$; clearly, $\overline{L} \subseteq loop_0^A(P, M(P,A))$ and $\overline{L} \subseteq M(P,A)$, so $S \cap M(P,A) \neq \emptyset$, which conflicts to the definition of S. Now we give the detail of the proof.

For any atom a, if $\neg a \in S$, then there exists a rule r such that $head(r) = a$ and $Atmost(P,A) \setminus A^- \not\models body^+(r)$, which is equivalent to $((Atoms(P) \setminus Atmost(P,A)) \cup A^-) \cap body^+(r) \neq \emptyset$. As $Atoms(P) \setminus Atmost(P,A) = \overline{S} \cup \overline{M(P,A)}$, there exists a rule r such that $head(r) = a$ and $\overline{S} \cap body^+(r) \neq \emptyset$.

Let $G_P^{\overline{S}}$ be the $\overline{S}$ induced subgraph of G and $L = \{a \mid$ for each $b \in \overline{S}$. If there is a path from a to b in $G_P^{\overline{S}}$, then there is a path from b to a in $G_P^{\overline{S}}\}$. Now we prove that $L \neq \emptyset$ and $R^-(L, M(P,A)) = \emptyset$.

For any atom $a \in \overline{S}$, let $H^-(a) = \{b \mid b \in \overline{S}$; there is a path from a to b and there is not any path from b to a in $G_P^{\overline{S}}\}$ and $H^+(a) = \{b \mid b \in \overline{S}$, there is a path from b to a in $G_P^{\overline{S}}\}$. If $L = \emptyset$, then for each atom $a \in \overline{S}$, $H^-(a) \neq \emptyset$. If an atom $b \in H^-(a)$, then $a \in H^+(b)$ and $H^-(b) \subseteq H^-(a) \setminus \{a\}$. If another atom $c \in H^-(b)$, then $H^-(c) \subseteq H^-(a) \setminus \{a, b\}$. So we can form an infinite list of atoms $\{a_1, a_2, \ldots\}$, where $a_{i+1} \in H^-(a_i)$ and $a_{i+1} \neq a_j$ $(1 \leq j \leq i, 1 \leq i \leq \infty)$, but $\overline{S}$ is finite, so $L \neq \emptyset$.

Clearly, L is a loop of P, $L \subseteq \overline{S}$, $L \neq \emptyset$, and for each external support rule r of L, $body^+(r) \cap \overline{S} = \emptyset$. Furthermore, each rule whose head belongs to $\overline{S}$ is not satisfied under $M(P,A) \cup S$, so $R^-(L, M(P,A)) = \emptyset$. Then $\overline{L} \subseteq loop_0^A(P, M(P,A))$, $\overline{L} \subseteq M(P,A)$, and $\overline{L} \subseteq S$, which conflicts with the definition of S.

So $S = \emptyset$ and $M(P,A) = \overline{Atoms(P) \setminus Atmost(P,A)}$. □

LEMMA A.2. *For any normal logic program P and any $A \subseteq Lit(P)$,*

$$Expand(P,A) \subseteq T_0(P,A).$$

PROOF. $Expand(P,A)$ is the least fixed point of the operator E_A^P defined by Equation (10). For any set X of literals such that $X \subseteq T_0(P,A)$, we want to prove $E_A^P(X) \subseteq T_0(P,A)$. First we prove that $Atleast(P, A \cup X) \subseteq T_0(P,A)$.

$Atleast(P, A \cup X)$ is the least fixed point of operator $F_{A \cup X}^P$. Let $Y \subseteq T_0(P,A)$; we consider $F_i(P,Y)$ $(1 \leq i \leq 5)$, respectively.

Let x be a literal; if $x \in F_1(P,Y)$, then there is a rule $x \leftarrow l_1, \ldots, l_n$ and $\{l_1, \ldots, l_n\} \subseteq Y$. The corresponding clause $x \vee \bar{l}_1 \vee \cdots \vee \bar{l}_n$ is in $comp(P)$, then $x \in UP(comp(P) \cup Y)$. As $Y \subseteq T_0(P,A)$ and $T_0(P,A)$ is the least fixed point of our operator U_A^P defined by Equation (9), $x \in UP(comp(P) \cup T_0(P,A))$, $x \in T_0(P,A)$. So $F_1(P,Y) \subseteq T_0(P,A)$.

If $\bar{x} \in F_2(P,Y)$, then for every rule r, if $head(r) = x$, then $body(r)$ is false under Y. The corresponding clause $\bar{x} \vee v_1 \vee v_2 \vee \cdots \vee v_n$ is in $comp(P)$ and $\neg v_1, \neg v_2, \ldots, \neg v_n \in UP(comp(P) \cup Y)$, where v_i is the new variable and stands for the body of the rule. Clearly $\bar{x} \in UP(comp(P) \cup Y)$. In particular, if $n = 0$, there is a clause $\bar{x}$ in $comp(P)$, then $\bar{x} \in UP(comp(P) \cup Y)$. Similar to the proof for $F_1(P,Y)$, $F_2(P,Y) \subseteq T_0(P,A)$.

If $x \in F_3(P,Y)$, then there exists a clause $\neg a \vee v_1 \vee v_2 \vee \cdots \vee v_n$ in $comp(P)$, $a \in Y$, and for any $j \neq i$, $\neg v_j \in Y$, then $v_i \in UP(comp(P) \cup Y)$. As the clause $\neg v_i \vee x$ is in $comp(P)$, $x \in UP(comp(P) \cup Y)$, thus $F_3(P,Y) \subseteq T_0(P,A)$.

If $\bar{x} \in F_4(P,Y)$, then there is a literal $\neg a \in Y$, a rule $a \leftarrow l_1,\ldots,l_n$, and $\{l_1,\ldots,l_n\}$ is true under $Y \cup \{x\}$. As Y is consistent, then $\{l_1,\ldots,l_n\}$ is not true under Y, so x is equivalent to a literal l_i $(1 \leq i \leq n)$ and for any $j \neq i$, $1 \leq j \leq n$, $l_j \in Y$. The corresponding clause is $a \vee \bar{l}_1 \vee \cdots \vee \bar{l}_n$, then $\bar{x} \in UP(comp(P) \cup Y)$, so $F_4(P,Y) \subseteq T_0(P,A)$.

If Y is inconsistent, then $F_5(P,Y) = Lit(P)$. $T_0(P,A)$ is also inconsistent, then $T_0(P,A) = Lit(P)$, $F_5(P,Y) \subseteq T_0(P,A)$.

As for any $Y \subseteq T_0(P,A)$, $F^P_{A\cup X}(Y) \subseteq T_0(P,A)$, we have proved that $Atleast(P,A \cup X) \subseteq T_0(P,A)$.

Now we prove that $\overline{Atoms(P) \setminus Atmost(P,A)} \subseteq T_0(P,A)$.

From Lemma A.1, $M(P,A \cup X) = \overline{Atoms(P) \setminus Atmost(P,A \cup X)}$. Clearly, for any $Y \subseteq T_0(P,A)$, $loop^A_0(P,Y) \subseteq T_0(P,A)$ and $F^A_2(P,Y) \subseteq T_0(P,A)$, so $M^A_P(P,Y) \subseteq T_0(P,A)$, $M(P,A \cup X) \subseteq T_0(P,A)$.

So $E^P_A(X) \subseteq T_0(P,A)$ and $Expand(P,A) \subseteq T_0(P,A)$. □

LEMMA A.3. *For any simplified logic program P and any $A \subseteq Lit(P)$,*

$$T_0(P,A) \subseteq Expand(P,A).$$

PROOF. $T_0(P,A)$ is the least fixed point of our operator U^P_A defined by Equation (9). For any set X of literals such that $X \subseteq Expand(P,A)$, we want to prove $U^P_A(X) \subseteq Expand(P,A)$. X is consistent; if not, $Expand(P,A) = Lit(P)$ and $U^P_A(X) \subseteq Expand(P,A)$. First we prove that $UP(comp(P) \cup X) \cap Lit(P) \subseteq Expand(P,A)$.

From the definition of $comp(P)$ in Section 2, some new variables are introduced, and there are four kinds of clauses in $comp(P)$. We consider them one by one. First we give some notions that will be used in the proof.

A set of literals $X \subseteq Lit(comp(P))$ is called a *proper submodel* of a set of clauses $comp(P)$ if, for any new variable v_i which stands for the body $\{l_1,\ldots,l_m\}$, $v_i \in X$ implies $\{l_1,\ldots,l_m\} \subseteq X$ and $\neg v_i \in X$ implies that there exists some j, $1 \leq j \leq m$, such that $\bar{l}_j \in X$. For any set of literals $Y \subseteq Lit(comp(P))$, $Sub(Y) = \{v_i \mid v_i \in Y, v_i$ stands for the body $\{l_1,\ldots,l_m\}$, and $\{l_1,\ldots,l_m\} \not\subseteq Y\}$.

For any set of literals $X \subseteq Lit(comp(P))$ such that X is a proper submodel of $comp(P)$ and $(X \cap Lit(P)) \subseteq Expand(P,A)$.

For type 1, the clause is $\neg a$ and $\neg a \in F_2(X \cap Lit(P))$.

For type 2, the clause is $\bar{l}_1 \vee \cdots \vee \bar{l}_n \vee l$. Assume that it can be reduced to a unit clause under X. As the rule is a simplified rule, the head of the rule does not belong to the set of the atoms that appear in the body; then there are only two cases. If the remaining literal is l, then $\{l_1,\ldots,l_n\} \subseteq X$, so $l \in F_1(P, X \cap Lit(P))$, $l \in Expand(P,A)$. If the remaining literal is $\bar{l}_i$, then $\bar{l} \in X$, and for any $j \neq i, 1 \leq j \leq n, l_j \in X$, so $\bar{l}_i \in F_4(P, X \cap Lit(P))$, $\bar{l}_i \in Expand(P,A)$. So if the unit clause c is reduced from one of this kind of clauses, then $c \in Expand(P,A)$.

For type 3, the clauses are $\neg a \vee v_1 \vee \cdots \vee v_m$, $v_i \vee \bar{l}_{1_i} \vee \cdots \vee \bar{l}_{n_i}$, $\neg v_i \vee l_{1_i}, \ldots \neg v_i \vee l_{n_i}$ $(1 \leq i \leq m)$.

For the clause $\neg a \vee v_1 \vee \cdots \vee v_m$, assume that it can be reduced to a unit clause under X. There are only two cases. If the remaining literal is $\neg a$, then $\{\neg v_1,\ldots,\neg v_m\} \subseteq X$. As X is a proper submodel of $comp(P)$, for any rule $r \in P$, if $head(r) = a$, then $body(r)$ is false under $X \cap Lit(P)$, so $\neg a \in F_2(P, X \cap Lit(P))$, $\neg a \in Expand(P,A)$. If the remaining literal is v_i, $v_i \notin Lit(P)$, so we do not need to consider this case.

For the clause $v_i \vee \bar{l}_1 \vee \cdots \vee \bar{l}_n$, assume that it can be reduced to a unit clause under X. There are only two cases. If the remaining literal is $\bar{l}_j$, then $\neg v_i \in X$, and for any

$k \neq j$, $1 \leq k \leq n$, $l_k \in X$. As X is a proper submodel of $comp(P)$, then $\bar{l}_j \in X$, so $\bar{l}_j \in Expand(P,A)$. If the remaining literal is v_i, $v_i \notin Lit(P)$.

For the clause $\neg v_i \vee l_j$, assume that it can be reduced to a unit clause under X. There are only two cases. If the remaining literal is l_j, then $v_i \in X$. As X is a proper submodel of $comp(P)$, $l_j \in X$. So $l_j \in Expand(P,A)$. If the remaining literal is $\neg v_i$, $v_i \notin Lit(P)$.

So if the unit clause c is reduced from the clause of type 3, then $\{c\} \cap Lit(P) \subseteq Expand(P,A)$.

For type 4, the clause is $\bar{l}_1 \vee \cdots \vee \bar{l}_n$. Assume that it can be reduced to a unit clause under X. If the remaining literal is $\bar{l}_i$, then for any $j \neq i, 1 \leq j \leq n, l_j \in X$, so $\bar{l}_i \in F_4(P, X \cap Lit(P))$. So if the unit clause c is reduced from one of this kind of clause, then $c \in Expand(P,A)$.

$unit\_clause(assign(X, comp(P)))$ returns the union of unit clauses reduced from $comp(P)$ under X; we denote $UPO(comp(P),X)$ for short. So for any $X \subseteq Lit(comp(P))$, X is a proper submodel of $comp(P)$ and $X \cap Lit(P) \subseteq Expand(P,A)$, we have $UPO(comp(P),X) \cap Lit(P) \subseteq Expand(P,A)$.

Let $Y = UPO(comp(P),X)$, if Y is not a proper submodel of $comp(P)$, then there exists some $v_i \in Y$ which stands for the body $\{l_1, \ldots, l_n\}$ and for some $1 \leq j \leq n$, $l_j \notin Y$. v_i can only come from the clause $\neg a \vee v_1 \vee \cdots \vee v_m$, so $a \in Y$, and for all $k \neq i$, $1 \leq k \leq m$, $\neg v_k \in Y$, then $\{l_1, \ldots, l_n\} \subseteq F_3(P, Y \cap Lit(P))$, $UPO(comp(P), Sub(Y)) \cap Lit(P) \subseteq Expand(P,A)$. Let $Z = Y \setminus Sub(Y)$, it is clear that Z is a proper submodel of $comp(P)$, then $UPO(comp(P), Z) \cap Lit(P) \subseteq Expand(P,A)$. So $UPO(comp(P), Y) \cap Lit(P) = (UPO(comp(P), Z) \cup UPO(comp(P), Sub(Y))) \cap Lit(P) \subseteq Expand(P,A)$.

So $UP(comp(P) \cup X) \cap Lit(P) \subseteq Expand(P,A)$.

Now we prove that $loop_0(P,X) \subseteq Expand(P,A)$.

From Lemma A.1, for any $Y \subseteq Expand(P,A)$, $loop_0^A(P,Y) \subseteq Expand(P,A)$, then $loop_0(P,X) \subseteq Expand(P,A)$.

So $U_A^P(X) \subseteq Expand(P,A)$ and $T_0(P,A) \subseteq Expand(P,A)$. □

PROOF (THEOREM 3.5). Directly from Lemmas A.2 and A.3. □

A.4. Proof of Proposition 4.1

PROOF (PROPOSITION 4.1). Deciding whether $\neg a \in floop_0(P,A)$ is an NP problem. We can proceed as follows: guess a loop L of P such that $a \in L$ and verify that L has no external support rules under A.

We prove the hardness for NP by reducing the 3-SAT problem to the problem. Now we give the detail of the proof.

Given a set of atoms $Atoms = \{a_1, a_2, \ldots, a_n\}$ and $Lit = Atoms \cup \overline{Atoms}$, let Γ be a set of clauses of the form $l_1 \vee l_2 \vee l_3$, where $l_i \in Lit$ $(1 \leq i \leq 3)$, deciding whether Γ is satisfiable is a 3-SAT problem.

Let p be an atom not in $Atoms$, we denote by $Loop(Atoms)$ the following logic program.

$$a_i \leftarrow p.$$
$$p \leftarrow a_i, p \quad (1 \leq i \leq n).$$

Clearly, every subset of $Atoms$ extended with p is a loop of $Loop(Atoms)$, and every loop of $Loop(Atoms)$ is in this form.

Let C be a clause in Γ and a, b, c be the three atoms mentioned in C, then there are only eight cases to form C from a, b, c. We use $Tran(C)$ to denote a disjunctive rule according to C, which is different for each case. We give the details in Table III.

Table III. Rule $Tran(C)$ of Clause C for Each Case

Clause C	Rule $Tran(C)$
$a \vee b \vee c$	$p \leftarrow a, b, c.$
$a \vee b \vee \neg c$	$p \vee c \leftarrow a, b.$
$a \vee \neg b \vee c$	$p \vee b \leftarrow a, c.$
$a \vee \neg b \vee \neg c$	$p \vee b \vee c \leftarrow a.$
$\neg a \vee b \vee c$	$p \vee a \leftarrow b, c.$
$\neg a \vee b \vee \neg c$	$p \vee a \vee c \leftarrow b.$
$\neg a \vee \neg b \vee c$	$p \vee a \vee b \leftarrow c.$
$\neg a \vee \neg b \vee \neg c$	$p \vee a \vee b \vee c \leftarrow .$

Let $Sat(\Gamma) = \{ Tran(C) \mid C \in \Gamma \}$; clearly, $Sat(\Gamma)$ is a disjunctive logic program. Furthermore, if an interpretation $I \subseteq Atoms$ does not satisfy a clause C, then $Tran(C)$ is an external support rule of the loop $I \cup \{p\}$ under $Atoms$; otherwise, $Tran(C)$ is not an external support rule under $Atoms$.

We use $Tran(\Gamma)$ to denote the disjunctive logic program: $Loop(Atoms) \cup Sat(\Gamma)$. From the preceding discussion, for any interpretation $I \subseteq Atoms$, $I \cup \{p\}$ is a loop of $Tran(\Gamma)$, and if I satisfies a clause C in Γ, then $Tran(C)$ is not an external support rule of the loop $I \cup \{p\}$ under $Atoms$; otherwise, $Tran(C)$ is an external support rule of $I \cup \{p\}$ under $Atoms$.

So if there is an interpretation I that satisfies every clause in Γ, then loop $I \cup \{p\}$ of $Tran(\Gamma)$ has no external support rules under $Atoms$, and $\neg p \in floop_0(Tran(\Gamma), Atoms)$. If there does not exist such an interpretation, then for any $I \subseteq Atoms$, $I \cup \{p\}$ has at least one external support rule under $Atoms$; thus, there does not exist a loop of $Tran(\Gamma)$ with p that has no external support rules under $Atoms$, then $\neg p \notin floop_0(Tran(\Gamma), Atoms)$. So Γ is satisfiable if and only if $\neg p \in floop_0(Tran(\Gamma), Atoms)$.

It is known that the 3-SAT problem is NP-complete, thus deciding whether $\neg a \in floop_0(P, A)$ is NP-hard, the problem is NP-complete. □

A.5. Proof of Proposition 4.2

PROOF (PROPOSITION 4.2). Suppose otherwise, and let $r \in R^{-}(L_1 \cup L_2, A)$. Then $head(r) \cap (L_1 \cup L_2) \neq \emptyset$, $body^{+}(r) \cap (L_1 \cup L_2) = \emptyset$, $\bar{A} \cap body(r) = \emptyset$, $A \cap (head(r) \setminus (L_1 \cup L_2)) = \emptyset$, and $A \cap (L_1 \cup L_2) = \emptyset$. Thus $A \cap head(r) = \emptyset$, and either $head(r) \cap L_1 \neq \emptyset$ or $head(r) \cap L_2 \neq \emptyset$. So either $r \in R^{-}(L_1, A)$ or $r \in R^{-}(L_2, A)$, a contradiction with the assumption that L_1 and L_2 do not have any external support rules under A. □

A.6. Proof of Theorem 4.3

PROOF (THEOREM 4.3). The complexity part is straightforward, and the loops computed from $\mathcal{ML}_0(P, A, Atoms(P))$ are loops without external support rules for P with respect to A. Note that generally $\mathcal{ML}_0(P, A) \not\subseteq ml_0(P, A)$.

For the correctness when A is unfounded-free, observe that if L is a loop of P, then there must be a strongly connected component C of G_P such that $L \subseteq C$. Now if $L \in ml_0(P, A)$, then there must be such a component C such that either $L = C$ and $R^{-}(C, A) = \emptyset$, or $L \subset C$, $R^{-}(C, A) \neq \emptyset$ and $R^{-}(L, A) = \emptyset$. In the latter case, for any $r \in R^{-}(C, A)$, if $head(r) \cap A = \emptyset$, then $head(r) \cap L = \emptyset$; otherwise, r must be in $R^{-}(L, A)$, which is a contradiction with $R^{-}(L, A) = \emptyset$. If $head(r) \cap A \neq \emptyset$, as A is unfounded-free, then for any loop $L \in \mathcal{ML}_0(P, A, Atoms(P))$, $L \cap A = \emptyset$, and thus $(head(r) \cap A) \cap L = \emptyset$. So if $R^{-}(C, A) \neq \emptyset$, then any subset of C that is in $ml_0(P, A)$ must also be a subset of $S = C \setminus \bigcup_{r \in R^{-}(L,A)} H(r, A)$. Thus instead of G_P, we can recursively search that S induced subgraph of G_P. □

A.7. Proof of Proposition 4.4

PROOF (PROPOSITION 4.4). Clearly, deciding whether $\neg a \vee l \in \mathit{floop}_1(P,A)$ is an NP problem. We can proceed as follows: guess a loop L of P such that $a \in L$ and verify that L has only one external support rule r under A such that $l \in \mathit{body}(r) \cup \overline{\mathit{head}(r) \setminus L}$.

Similar to the proof of Proposition 4.1, we prove the hardness for NP by reducing the 3-SAT problem to the problem. We also follow the notions proposed in the preceding proof.

We use $\mathit{Tran}_1(\Gamma)$ to denote the disjunctive logic program: $\mathit{Loop}(\mathit{Atoms}) \cup \mathit{Sat}(\Gamma) \cup \{ p \leftarrow q,\ q \leftarrow \}$, where q is an atom not in Atoms and different from p. From the discussion in the preceding proof, for any interpretation $I \subseteq \mathit{Atoms}$, $I \cup \{p\}$ is a loop of $\mathit{Tran}_1(\Gamma)$, and if I satisfies a clause C in Γ, then $\mathit{Tran}(C)$ is not an external support rule of the loop $I \cup \{p\}$ under Atoms; otherwise, $\mathit{Tran}(C)$ is an external support rule of $I \cup \{p\}$ under Atoms.

So if there is an interpretation I that satisfies every clause in Γ, then loop $I \cup \{p\}$ of $\mathit{Tran}_1(\Gamma)$ has only one external support rule: $p \leftarrow q$ under Atoms, and $\neg p \vee q \in \mathit{floop}_1(\mathit{Tran}_1(\Gamma), \mathit{Atoms})$. If there does not exist such an interpretation, then for any $I \subseteq \mathit{Atoms}$, $I \cup \{p\}$ has more than one external support rule under Atoms; thus there does not exist a loop of $\mathit{Tran}_1(\Gamma)$ with p that has only one external support rule under Atoms, then $\neg p \vee q \notin \mathit{floop}_1(\mathit{Tran}_1(\Gamma), \mathit{Atoms})$. So Γ is satisfiable if and only if $\neg p \vee q \in \mathit{floop}_1(\mathit{Tran}_1(\Gamma), \mathit{Atoms})$. Furthermore, as q is a consequence of $\mathit{Tran}_1(\Gamma)$, $\mathit{Atoms} \subseteq U(\mathit{Tran}_1(\Gamma), \emptyset)$.

It is known that the 3-SAT problem is NP-complete; thus for $A \subseteq U(P, \emptyset)$, deciding whether $\neg a \vee l \in \mathit{floop}_1(P,A)$ is NP-hard, the problem is NP-complete. □

A.8. Proof of Proposition 4.5

PROOF (PROPOSITION 4.5). For any rule r in the disjuctive logic program P, we also define $\mathit{ml}_1(P,A,r)$, that is, the set of maximal loops of P that have r as their only external support rule under A, and $\mathit{ml}_1(P,A) = \bigcup_{r \in P} \mathit{ml}_1(P,A,r)$. Let

$$\mathit{floop}'_1(P,A) = \bigcup_{L \in \mathit{ml}_1(P,A)} \{ \neg a \vee l \mid \ a \in L, l \in \mathit{body}(r) \cup \overline{\mathit{head}(r) \setminus L}, \\ R^-(L,A) = \{r\} \}.$$

Clearly, if $L \in \mathit{ml}_1(P,A)$, then $L \in \mathit{loop}_1(P,A)$, so $\mathit{floop}_1(P,A)$ implies $\mathit{floop}'_1(P,A)$. We only need to prove that $\mathit{floop}'_1(P,A)$ implies the Equation (11) under $\mathit{floop}_0(P,A)$.

Suppose $\mathit{floop}_0(P,A)$ and $\mathit{floop}'_1(P,A)$ are true. We show that for any rule $r \in P$ such that $\overline{A} \cap \mathit{body}(r) = \emptyset$, and any $L \in \mathit{ml}_0(P \setminus \{r\}, A)$,

$$\{ \neg a \vee l \mid a \in L, l \in \mathit{body}(r) \cup \overline{\mathit{head}(r) \setminus L} \}, \tag{13}$$

is true. First, L is also a loop for P, and second, either $R^-(L,A) = \emptyset$ or $R^-(L,A) = \{r\}$. For the first case, $\overline{L} \subseteq \mathit{floop}_0(P,A)$, thus Equation (13) is true. For the second case, $L \in \mathit{loop}_1(P,A)$, and there does not exist any other such loop L' such that $L \subset L'$, so $l \in \mathit{ml}_1(P,A,r)$, Equation (13) is contained in $\mathit{floop}'_1(P,A)$, and thus true.

So $\mathit{floop}_0(P,A) \cup \mathit{floop}'_1(P,A)$ implies the Equation (11), and $\mathit{floop}_0(P,A) \cup \mathit{floop}_1(P,A)$ implies the Equation (11). □

A.9. Proof of Proposition 4.6

PROOF (PROPOSITION 4.6). Let S be an answer set of P and $\mathcal{S} = S \cup \overline{\mathit{Atoms}(P) \setminus S}$; clearly, $A \subseteq \mathcal{S}$. Let X be an unfounded set for P with respect to A, from the definition of unfounded sets, X is also an unfounded set for P with respect to $\mathcal{S}$. Theorem 4.6

in Leone et al. [1997] demonstrates that $\mathcal{S}$ is unfounded-free. So if A is not unfounded-free, then there exists an unfounded set X for P with respect to A such that $X \cap A \neq \emptyset$, thus X is also an unfounded set with respect to $\mathcal{S}$ and $X \cap \mathcal{S} \neq \emptyset$, a contradiction with $\mathcal{S}$ is unfounded-free. So A is also unfounded-free. □

A.10. Proof of Proposition 4.7

PROOF (PROPOSITION 4.7). Similar to the proof of Proposition 4.1, we prove the hardness for NP by reducing the 3-SAT problem to the problem, deciding whether a literal $l \in T^1(P)$. We follow the notions proposed in the preceding proof.

We use $Tran_2(\Gamma)$ to denote the disjunctive logic program: $Loop(Atoms) \cup Sat(\Gamma) \cup \{\, p \leftarrow q,\ q \leftarrow not\, m,\ m \leftarrow not\, q,\ \leftarrow not\, p \,\}$, where q and m are different atoms not in $Atoms$ and different from p. Clearly, $Atoms \cup \{p\} \subset T^1(P)$. We want to know whether q belongs to $T^1(P)$.

From the discussion in the proof of Proposition 4.1, if Γ is satisfiable, then there exists an interpretation $I \subseteq Atoms$ that satisfies every clause in Γ; thus $I \cup \{p\}$ is a loop of $Tran_2(\Gamma)$ and $I \cup \{p\}$ has only one external support rule $p \leftarrow q$ under $Atoms \cup \{p\}$, as $p \in T^1(P)$, then $q \in T^1(P)$. If Γ is unsatisfiable, then every possible interpretation does not satisfy Γ; thus every possible loop in $Tran_2(\Gamma)$ has more than one external support rule, so $q \notin T^1(P)$.

So Γ is satisfiable if and only if $q \in T^1(P)$. Thus deciding whether a literal $l \in T^1(P)$ is NP-hard. □

A.11. Proof of Theorem 4.8

LEMMA A.4. *For any disjunctive logic program P, any $A \subseteq Lit(P)$, and any $X \subseteq GUS_P(A)$, $GUS_P(A) = GUS_P(A \cup \overline{X})$.*

PROOF. Clearly, $GUS_P(A) \subseteq GUS_P(A \cup \overline{X})$. Now we prove that $GUS_P(A \cup \overline{X}) \subseteq GUS_P(A)$.

$GUS_P(A \cup \overline{X})$ is an unfounded set of P with respect to $A \cup \overline{X}$ and $X \subseteq GUS_P(A) \subseteq GUS_P(A \cup \overline{X})$, from the definition of unfounded sets, $GUS_P(A \cup \overline{X})$ is also an unfounded set of P with respect to A, so $GUS_P(A \cup \overline{X}) \subseteq GUS_P(A)$.

So $GUS_P(A) = GUS_P(A \cup \overline{X})$. □

PROOF (THEOREM 4.8). Let $X \subseteq GUS_P(A)$. First we prove that $M(P,A) \subseteq GUS_P(A)$. As $M(P,A)$ is the least fixed point of our operator M_P^A, we only need to prove that $M_P^A(X) \subseteq GUS_P(A)$.

Note that if an atom $a \in F_2^A(P,X)$, then for each $r \in P$, $a \in head(r)$ implies (1) $\overline{A \cup \overline{X}} \neq \emptyset$, or (2) $(head(r) \setminus \{a\}) \cap A \neq \emptyset$. From the definition of unfounded sets, $\{a\}$ is an unfounded set for P with respect to $A \cup \overline{X}$. Furthermore, $GUS_P(A)$ exists, and from Lemma A.4, $GUS_P(A \cup \overline{X})$ exists, then $\{a\} \subseteq GUS_P(A \cup \overline{X})$, and thus $F_2^A(P,X) \subseteq GUS_P(A \cup \overline{X})$. Let L be a loop of P and $R^-(L, A \cup \overline{X}) = \emptyset$; clearly, L is an unfounded set of P with respect to $A \cup \overline{X}$, and $L \subseteq GUS_P(A \cup \overline{X})$. So $M_P^A(X) \subseteq GUS_P(A \cup \overline{X})$. From Lemma A.4, $M_P^A(X) \subseteq GUS_P(A)$, thus $M(P,A) \subseteq GUS_P(A)$.

Now we prove that $GUS_P(A) \subseteq M(P,A)$. Let $S = GUS_P(A) \setminus M(P,A)$, we want to prove $S = \emptyset$. The sketch of the proof is the following: if $S \neq \emptyset$, then there exists a loop $L \subseteq S$ and $R^-(L, A \cup \overline{M(P,A)}) = \emptyset$, clearly, $L \subseteq loop_0^A(P, M(P,A))$ and $L \subseteq M(P,A)$, so $S \cap M(P,A) \neq \emptyset$, which conflicts to the definition of S. Now we give the detail of the proof.

For any atom a, if $a \in S$, then every rule $r \in P$ such that $a \in head(r)$, we have $\overline{A} \cap body(r) \neq \emptyset$, $A \cap (head(r) \setminus GUS_P(A)) \neq \emptyset$, or $body^+(r) \cap GUS_P(A) \neq \emptyset$. As $GUS_P(A) = S \cup M(P,A)$, there exists a rule r such that $a \in head(r)$, $S \cap body^+(r) \neq \emptyset$.

Let G_P^S be the S induced subgraph of the positive dependence graph G_P, and $L = \{ a \mid$ for each $b \in S$; if there is a path from a to b in G_P^S, then there is a path from b to a in $G_P^S \}$. Now we prove that $L \neq \emptyset$ and $R^-(L, A \cup \overline{M(P,A)}) = \emptyset$.

For any atom $a \in S$, let $H^-(a) = \{ b \mid b \in S$, there is a path from a to b and there is not any path from b to a in $G_P^S \}$. Clearly, for each atom $a \in S$, there exists an atom $b \in S$ such that there is an arc from a to b in G_P^S. If $L = \emptyset$, then for each atom $a \in S$, $H^-(a) \neq \emptyset$. If an atom $b \in H^-(a)$, then $H^-(b) \subseteq H^-(a) \setminus \{a\}$. If another atom $c \in H^-(b)$, then $H^-(c) \subseteq H^-(a) \setminus \{a, b\}$. So we can form an infinite list of atoms $\{a_1, a_2, \ldots\}$, where $a_{i+1} \in H^-(a_i)$ and $a_{i+1} \neq a_j$ $(1 \leq j \leq i,\ 1 \leq i \leq \infty)$. But S is finite, so $L \neq \emptyset$.

Clearly, L is a loop of P, $L \subseteq S$, $L \neq \emptyset$. Let r be an external support rule of L; if $body^+(r) \cap S \neq \emptyset$, then there exist two atoms a and b such that $a \in L$, $b \in S \setminus L$, and there is a path from a to b in G_P^S, thus there is also a path from b to a in G_P^S. Note that, for each atom $c \in S$, if there is a path from b to c in G_P^S, then there is a path from a to c, thus there is a path from c to a and also a path from c to b. So $b \in L$, which conflicts to $b \in S \setminus L$; thus for each external support rule r of L, $body^+(r) \cap S = \emptyset$. Furthermore, all rules $r \in P$ such that $head(r) \cap S \neq \emptyset$, $\overline{A} \cap body(r) \neq \emptyset$, $(M(P,A) \cup S) \cap body(r) \neq \emptyset$, or $A \cap (head(r) \setminus (S \cup M(P,A))) \neq \emptyset$, so $R^-(L, A \cup \overline{M(P,A)}) = \emptyset$. Then $L \subseteq loop_0^A(P, M(P,A))$, $L \subseteq M(P,A)$, and $L \subseteq S$, which conflicts to the definition of S.

So $S = \emptyset$ and $M(P,A) = GUS_P(A)$. □

A.12. Proof of Theorem 4.9

LEMMA A.5. *For any disjunctive logic program P, $\mathcal{W}_P^\omega(\emptyset) \subseteq T_0(P)$.*

PROOF. $\mathcal{W}_P^\omega(\emptyset)$ is the least fixed point of the operator $\mathcal{W}_P$. For any set X of literals such that $X \subseteq T_0(P)$, we want to prove $\mathcal{W}_P(X)) \subseteq T_0(P)$. First we prove that $\mathcal{T}_P(X) \subseteq T_0(P)$.

If an atom $a \in \mathcal{T}_P(X)$, then there is a rule $r \in P$ such that $a \in head(r)$, $(head(r) \setminus \{a\}) \subseteq \overline{X}$, and $body(r) \subseteq X$. The corresponding clause $head(r) \vee \bigvee \overline{body(r)}$ is in $comp(P)$, then $a \in UP(comp(P) \cup X)$. So $\mathcal{T}_P(X) \subseteq T_0(P)$.

Now we prove that $\overline{GUS_P(X)} \subseteq T_0(P)$. From Theorem 4.8, $M(P,X) = GUS_P(X)$. $T_0(P)$ returns a set of consequence of P, then every subset of $T_0(P)$ is unfounded-free for P. Note that if $X_1 \subseteq X_2 \subseteq T_0(P)$, then $\bigcup_{L \in ML_0(P,X_1)} L \subseteq \bigcup_{L \in ML_0(P,X_2)} L$, from the definition of $T_0(P)$, $\bigcup_{L \in ML_0(P,X_1)} \overline{L} \subseteq \bigcup_{L \in ML_0(P,X_2)} \overline{L} \subseteq T_0(P)$. Additionally, for any $\overline{Y} \subseteq T_0(P)$, $X \cup \overline{Y}$ is unfounded-free for P, from Theorem 4.3, $\bigcup_{L \in ML_0(P, X \cup \overline{Y})} L = \bigcup_{L \in ml_0(P, X \cup \overline{Y})} L = \overline{loop_0^X(P,Y)}$. So for any $\overline{Y} \subseteq T_0(P)$, $\overline{loop_0^X(P,Y)} \subseteq T_0(P)$. Now we prove that $\overline{F_2^X(P,Y)} \subseteq T_0(P)$.

For any atom $a \in F_2^X(P,Y)$ and any rule $r \in P$, if $a \in head(r)$, then $\overline{X} \cap body(r) \neq \emptyset$, $Y \cap body(r) \neq \emptyset$, or $(head(r) \setminus \{a\}) \cap X \neq \emptyset$. The corresponding clause $\neg a \vee v_1 \vee v_2 \vee \cdots \vee v_n$ is in $comp(P)$ and $\neg v_1, \neg v_2, \ldots, \neg v_n \in UP(comp(P) \cup X \cup \overline{Y})$, where v_i is the new variable and stands for the body of the rule. Clearly, $\neg a \in UP(comp(P) \cup X \cup \overline{Y})$. In particular, if $n = 0$, there is a clause $\neg a$ in $comp(P)$. So $\overline{F_2^X(P,Y)} \subseteq T_0(P)$.

So $\overline{M_P^X(P,Y)} \subseteq T_0(P)$, $\overline{M(P,X)} \subseteq T_0(P)$. □

LEMMA A.6. *For any simplified disjunctive logic program P that does not contain constraints, $T_0(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$.*

PROOF. $T_0(P)$ is equivalent to the least fixed point of our operator f defined by Equation (8). For any set X of literals such that $X \subseteq \mathcal{W}_P^\omega(\emptyset)$, we want to prove $f(X) \subseteq \mathcal{W}_P^\omega(\emptyset)$. Note that X is always consistent. First we prove that $UP(comp(P) \cup X) \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

Note that P is a simplified program from which, without constraints (from the definition of $comp(P)$ in Section 2), some new variables are introduced and there are three kinds of clauses in $comp(P)$. We consider them one by one. First we give some notions that will be used in the proof.

A set of literals $X \subseteq Lit(comp(P))$ is called a *proper submodel* of a set of clauses $comp(P)$ if for any new variable v_i which stands for the set of literals $body(r_i) \cup \overline{head(r_i) \setminus \{a\}}$ (formula $body(r_i) \wedge \bigwedge_{p \in head(r_i) \setminus \{a\}} \neg p$), $v_i \in X$ implies $(body(r_i) \cup \overline{head(r_i) \setminus \{a\}}) \subseteq X$, and $\neg v_i \in X$ implies that there exists a literal $l \in body(r_i) \cup \overline{head(r_i) \setminus \{a\}}$ such that $\bar{l} \in X$. For any set of literals $Y \subseteq Lit(comp(P))$, $Sub(Y) = \{ v_i \mid v_i \in Y, v_i$ stands for the set $body(r_i) \cup \overline{head(r_i) \setminus \{a\}}$, and $(body(r_i) \cup \overline{head(r_i) \setminus \{a\}}) \not\subseteq Y \}$.

Consider any set of literals $X \subseteq Lit(comp(P))$ such that X is a proper submodel of $comp(P)$ and $(X \cap Lit(P)) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

For type 1, the clause is $\neg a$ for an atom a for which there is no rule in P with a as its head. Clearly, $\{a\}$ is an unfounded set for P with respect to $\emptyset$, then $\neg a \in \mathcal{W}_P^\omega(\emptyset)$.

For type 2, the clause is $head(r) \vee \bigvee \overline{body(r)}$ for each rule r in P. Assume that it can be reduced to a unit clause under X. As the rule is a simplified rule, atoms in the head do not appear in the body, then there are only two cases. If the remaining literal $l \in head(r)$, then $(body(r) \cup \overline{head(r) \setminus \{l\}}) \subseteq X$, so $l \in T_P(X)$ and $l \in \mathcal{W}_P^\omega(\emptyset)$. If the remaining literal $l \in \overline{body(r)}$, then $(\overline{head(r)} \cup (body(r) \setminus \{\bar{l}\})) \subseteq X \subseteq \mathcal{W}_P^\omega(\emptyset)$, thus $head(r) \subseteq GUS_P(X \cap Lit(P))$ and $(body(r) \setminus \{\bar{l}\}) \subseteq X$, so $l \in \mathcal{W}_P^\omega(\emptyset)$. So if the unit clause c is reduced from one of this kind of clauses, then $c \in \mathcal{W}_P^\omega(\emptyset)$.

For type 3, the clauses correspond to the clauses formed in item 4 for the definition of $comp(P)$ in Section 2.

For the clause $\neg a \vee v_1 \vee \cdots \vee v_t$, assume that it can be reduced to a unit clause under X. There are only two cases. If the remaining literal is $\neg a$, then $\{\neg v_1, \ldots, \neg v_t\} \subseteq X$. As X is a proper submodel of $comp(P)$, then $\{a\}$ is an unfounded set for P with respect to $X \cap Lit(P)$, thus $\neg a \in \mathcal{W}_P^\omega(\emptyset)$. If the remaining literal is v_i, $v_i \notin Lit(P)$, so we do not need to consider this case.

For clauses of the form $v_i \vee \bigvee \overline{body(r_i)} \vee \bigvee_{p \in head(r_i) \setminus \{a\}} p$ or $\neg v_i \vee l$, if the unit clause c is reduced from these clauses, as X is a proper submodel of $comp(P)$, then $\{c\} \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

In the algorithm for unit propagation, $unit\_clause(assign(X, comp(P)))$ returns the union of unit clauses reduced from $comp(P)$ under X, we denote $UPO(comp(P), X)$ for short. From the preceding discussion, we have proved that for any $X \subseteq Lit(comp(P))$, X is a proper submodel of $comp(P)$ and $X \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$, we have $UPO(comp(P), X) \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

Let $Y = UPO(comp(P), X)$. If Y is not a proper submodel of $comp(P)$, then there exists some $v_i \in Y$ which stands for the set $body(r_i) \cup \overline{head(r_i) \setminus \{a\}}$, and there is a literal l in this set that $l \notin Y$. v_i can only come from the clause $\neg a \vee v_1 \vee \cdots \vee v_t$, so $a \in Y$, and for all $k \neq i$, $1 \leq k \leq t$, $\neg v_k \in Y$. As $Y \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$, $a \in T_P(Y \cap Lit(P))$, then there is a rule r that $a \in head(r)$ and $body(r) \cup \overline{head(r) \setminus \{a\}} \subseteq Y$. So $UPO(comp(P), Sub(Y)) \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$. Let $Z = Y \setminus Sub(Y)$. It is clear that Z is a proper submodel

of $comp(P)$, then $UPO(comp(P), Z) \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$. So $UPO(comp(P), Y) \cap Lit(P) = (UPO(comp(P), Z) \cup UPO(comp(P), Sub(Y))) \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

So $UP(comp(P) \cup X) \cap Lit(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

Now we prove that for any $X \subseteq \mathcal{W}_P^\omega(\emptyset)$, $FLoop_0(P, X) \subseteq \mathcal{W}_P^\omega(\emptyset)$.

Clearly, X is unfounded-free, from Theorem 4.8, $FLoop_0(P, X) = floop_0(P, X) \subseteq \overline{GUS_P(X)} \subseteq \mathcal{W}_P^\omega(\emptyset)$.

So $f(X) \subseteq \mathcal{W}_P^\omega(\emptyset)$ and $T_0(P) \subseteq \mathcal{W}_P^\omega(\emptyset)$. □

PROOF (THEOREM 4.9). The theorem can be proved from Lemmas A.5 and A.6 directly. □

A.13. Proof of Proposition 4.10

LEMMA A.7. *For any disjunctive logic program P and a set A of literals, $floop_0(P, A)$ implies that $floop'_1(P, A)$ is equivalent to the following theory.*

$$\bigcup_{\overline{A} \cap body(r) = \emptyset, L \in ml_0(P \setminus \{r\}, A)} \{\neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L}\}. \tag{14}$$

PROOF. Suppose $floop_0(P, A)$ and $floop'_1(P, A)$ are true. We show that for any rule $r \in P$ such that $\overline{A} \cap body(r) = \emptyset$, and any $L \in ml_0(P \setminus \{r\}, A)$,

$$\{\, \neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L} \,\} \tag{15}$$

is true. First, L is also a loop for P, and second, either $R^-(L, A) = \emptyset$ or $R^-(L, A) = \{r\}$. For the first case, $\overline{L} \subseteq floop_0(P, A)$, thus Equation (15) is true. For the second case, $L \in loop_1(P, A)$, and there does not exist any other such loop L' such that $L \subset L'$, so $l \in ml_1(P, A, r)$, Equation (15) is contained in $floop'_1(P, A)$, and thus true.

Now suppose $floop_0(P, A)$ and Equation (14) are true. We show that $floop'_1(P, A)$ is true, that is, for any loop $L \in ml_1(P, A, r)$, Equation (15) is true. Clearly, $L \in ml_0(P \setminus \{r\}, A)$ or there exists a loop $L' \in ml_0(P \setminus \{r\}, A)$ such that $L \subset L'$. For the first case, Equation (14) is true, thus Equation (15) is true. For the second case, $L' \in loop_0(P, A)$, $floop_0(P, A)$ is true, thus Equation (15) is true. □

PROOF (PROPOSITION 4.10). For the first equation, clearly if P is head-cycle free, then $loop_0(P, A) = loop_0(sh(P), A)$. Let C be a strongly connected component of G_P, loop $L \subseteq C$, and $L \in ml_0(P, A)$. There are two cases: either $L = C$ and $R^-(C, A) = \emptyset$ or $L \subset C$, $R^-(C, A) \neq \emptyset$, and $R^-(L, A) = \emptyset$. In the later case, if $head(r) \cap A = \emptyset$, then $head(r) \cap L = \emptyset$; otherwise, r must be in $R^-(L, A)$, which is a contradiction with $R^-(L, A) = \emptyset$.

If $head(r) \cap A \neq \emptyset$, let $X = head(r) \cap L$. As P is head-cycle free, then $|X| \leq 1$, that is, there is at most one atom $a \in X$. As $r \in R^-(C, A)$, so $body^+(r) \cap C = \emptyset$, $\overline{A} \cap body(r) = \emptyset$, and $A \cap (head(r) \setminus C) = \emptyset$. $L \subset C$, let $Y = A \cap (head(r) \setminus L)$. If $Y \neq \emptyset$, let $b \in Y$; clearly, $a \neq b$, $a, b \in C$, $b \in head(r)$, which conflicts with that P is head-cycle free, so $Y = \emptyset$. Thus $a \notin L$; otherwise, r must be in $R^-(L, A)$, which is a contradiction with $R^-(L, A) = \emptyset$. So $(head(r) \cap A) \cap L = \emptyset$.

Thus if $R^-(C, A) \neq \emptyset$, then any subset of C that is in $ml_0(P, A)$ must also be a subset of $S = C \setminus \{\, H(r, A) \mid r \in R^-(C, A) \,\}$. Thus instead of G_P, we can recursively search the S included subgraph of G_P.

So if P is head-cycle free, then $ML_0(P, A) = ml_0(P, A)$. As $sh(P)$ is a normal program, from Theorem 3.2, $ML_0(sh(P), A) = ml_0(sh(P), A) = ml_0(P, A)$. So $ml_0(P, A) = ML_0(P, A) = ML_0(sh(P), A)$.

As $ML_0(sh(P \setminus \{r\}), A) = ML_0(P \setminus \{r\}, A)$, so the second equation is true.

Furthermore, as P is head-cycle free, if a loop $L \in ml_1(P,A,r)$, then for any loop $L' \supset L$ such that $R^-(L',A) = \{r\}$ and $head(r) \setminus L = head(r) \setminus L'$; otherwise, P is not head-cycle free. So $floop_1(P,A) \equiv floop'_1(P,A)$. Then from Lemma A.7 and the first equation, $floop_1(P,A) \equiv FLoop_1(P,A)$ under $floop_0(P,A)$. □

A.14. Proof of Theorem 4.11

PROOF (THEOREM 4.11). The proof is similar to the proof for Theorem 4.9; we just need to replace the set A of literals to the set S of pure disjunctions. □

REFERENCES

Anger, C., Gebser, M., and Schaub, T. 2006. Approaching the core of unfounded sets. In *Proceedings of the International Workshop on NonmonotoniReasoning (NMR'06)*. 58–66.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Brass, S. and Dix, J. 1999. Semantics of (disjunctive) logiprograms based on partial evaluation. *J. LogiProgram. 40,* 1, 1–46.

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 logiprograms. In *Proceedings of the 25th International Conference on LogiProgramming (ICLP'09)*. 130–144.

Chen, Y., Lin, F., Wang, Y., and Zhang, M. 2006. First-order loop formulas for normal logiprograms. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*. 298–307.

Clark, K. L. 1978. Negation as failure. In *Logiand Databases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, NY, 293–322.

Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., and Schaub, T. 2008. Conflict-driven disjunctive answer set solving. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*. 422–432.

Ferraris, P., Lee, J., and Lifschitz, V. 2006. A generalization of the Lin-Zhao theorem. *Ann. Math. Artif. Intell. 47,* 1, 79–101.

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.

Gebser, M., Schaub, T., and Thiele, S. 2007. Gringo: A new grounder for answer set programming. In *Proceedings of the 9th International Conference on LogiProgramming and Nonmonotoni Reasoning (LPNMR'07)*. 266–271.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logiprogramming. In *Proceedings of the 5th International Conference on LogiProgramming (ICLP'88)*. 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logiprograms and disjunctive databases. *New Gen. Comput. 9*, 365–385.

Giunchiglia, E., Lierler, Y., and Maratea, M. 2006. Answer set programming based on propositional satisfiability. *J. Autom. Reason. 36,* 4, 345–377.

Lee, J. 2005. A model-theoreticounterpart of loop formulas. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. 503–508.

Lee, J. and Lifschitz, V. 2003. Loop formulas for disjunctive logiprograms. In *Proceedings of the 19th International Conference on LogiProgramming (ICLP'03)*. 451–465.

Lee, J. and Lin, F. 2006. Loop formulas for circumscription. *Artif. Intell. 170,* 2, 160–185.

Lee, J. and Meng, Y. 2008. On loop formulas with variables. In *Proceedings of the 11th International Conference on Knowledge Representation and Reasoning (KR'08)*. 444–453.

Leone, N., Rullo, P., and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Inf. Comput. 135,* 2, 69–112.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. 2006. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic 7,* 3, 499–562.

Lifschitz, V. and Razborov, A. 2006. Why are there so many loop formulas? *ACM Trans. Comput. Logic 7,* 2, 261–268.

Lin, F. and Chen, Y. 2007. Discovering classes of strongly equivalent logiprograms. *J. Artif. Intell. Res. 28*, 431–451.

Lin, F. and Zhao, Y. 2004. ASSAT: Computing answer sets of a logiprogram by SAT solvers. *Artif. Intell. 157,* 1–2, 115–137.

Liu, G. and You, J. 2007. On the effectiveness of looking ahead in search for answer sets. In *Proceedings of the 9th International Conference on LogiProgramming and NonmonotoniReasoning (LPNMR'07)*. 303–308.

Niemelä, I. 1999. Logiprograms with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell. 25,* 3, 241–273.

Simons, P., Niemelä, I., and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artif. Intell. 138,* 1–2, 181–234.

Van Gelder, A. 1989. The alternating fixpoint of logiprograms with negation. In *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'89)*. 1–10.

Van Gelder, A., Ross, K., and Schlipf, J. S. 1991. The well-founded semantics for general logiprograms. *J. ACM 38,* 3, 620–650.

Wang, K. and Zhou, L. 2005. Comparisons and computation of well-founded semantics for disjunctive logiprograms. *ACM Trans. Comput. Logic 6,* 2, 295–327.

Received April 2011; revised August 2011, December 2011; accepted January 2012