Computing Loops with at Most One External Support Rule for Disjunctive Logic Programs

Xiaoping Chen[1], Jianmin Ji[1], and Fangzhen Lin[2]

[1] School of Computer Science and Technology,
University of Science and Technology of China, P.R. China
xpchen@ustc.edu.cn, jizheng@mail.ustc.edu.cn
[2] Department of Computer Science and Engineering
Hong Kong University of Science and Technology
flin@cs.ust.hk

Abstract. We extend to disjunctive logic programs our previous work on computing loop formulas of loops with at most one external support. We show that for these logic programs, loop formulas of loops with no external support can be computed in polynomial time, and if the given program has no constraints, an iterative procedure based on these formulas, the program completion, and unit propagation computes the least fixed point of a simplification operator used by DLV. We also relate loops with no external supports to the unfounded sets and the well-founded semantics of disjunctive logic programs by Wang and Zhou. However, the problem of computing loop formulas of loops with at most one external support rule is NP-hard for disjunctive logic programs. 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.

1 Introduction

This paper 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, those that are true in all answer sets, are of interest as they can be used to simplify the given program and help computing its answer sets. The best known example is the well-founded model for normal logic programs, 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 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, Chen *et al.* [1] proposed an iterative procedure of computing consequences of a non-disjunctive logic program based on unit propagation, the program's completion and its loop formulas. They showed that when restricted to loops with no external support, their procedure basically computes the well-founded model. They also considered using loops with at most one external support, and

P.M. Hill and D.S. Warren (Eds.): ICLP 2009, LNCS 5649, pp. 130–144, 2009.

showed that the loop formulas of these loops can be computed in polynomial time.

In this paper, we consider extending this work to disjunctive logic programs. As expected, loops with no external supports are closely related to well-founded models and greatest unfounded sets in disjunctive logic programs as well. However, many other issues are more complicated in disjunctive logic programs. In particular, the problem of computing the loop formulas of loops with at most one external support rule is NP-hard.

This paper is organized as follows. We briefly review the basic notions of logic programming in the next section. We then define loops with at most one external support rule under a given set of literals, and consider how to compute their loop formulas. We then consider how to use these loop formulas to derive consequences of a disjunctive logic program using unit propagation, and discuss related work, especially the greatest unfounded sets, the pre-processing step in DLV, and the well-founded semantics for disjunctive logic programs proposed by Wang and Zhou [2].

2 Preliminaries

In this paper, we consider only fully grounded finite disjunctive logic programs. A *disjunctive logic program* is a finite set of (disjunctive) rules of the form

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

where $n \geq m \geq k \geq 0$ and $a_1, \ldots, a_n$ are atoms. If $k = 0$, then this rule is called a *constraint*, if $k \neq 0$, it is a *proper rule*, and if $k = 1$, it is a *normal rule*. In particular, a *normal logic program* is a finite set of constraints and normal rules.

We will also write rule r of form (1) as

$$head(r) \leftarrow body(r), \tag{2}$$

where $head(r)$ is $a_1 \vee \cdots \vee a_k$, $body(r) = body^+(r) \wedge body^-(r)$, $body^+(r)$ is $a_{k+1} \wedge \cdots \wedge a_m$, and $body^-(r)$ 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 atoms, and $body(r)$ the set $\{ a_{k+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \}$ of literals obtained from the body of the rule with "*not*" replaced by "$\neg$".

Given a disjunctive 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 $\bar{l}$ below, 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

The *answer sets* of a disjunctive logic program is defined as in [3]. Given a disjunctive logic program P and a set S of atoms, the Gelfond-Lifschitz transformation of P on S, written P_S, is obtained from P by deleting:

1. each rule that has a formula $not\, p$ in its body with $p \in S$, and
2. all formulas of the form $not\, p$ in the bodies of the remaining rules.

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 logic program P [4], $Comp(P)$, is defined to be the set of propositional formulas that consists of the implication

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

for every rule r in P, and the implication

$$a \supset \bigvee_{r \in P,\ a \in head(r)} \left(body(r) \wedge \bigwedge_{p \in head(r) \setminus \{a\}} \neg p \right), \tag{4}$$

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 [5].

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 logic programs [4]. Given a disjunctive 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, i.e. 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

$$\bigvee_{p \in L} p \supset \bigvee_{r \in R^-(L)} \left(body(r) \wedge \bigwedge_{q \in head(r) \setminus L} \neg q \right). \tag{5}$$

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 [6], has been extended to disjunctive logic programs [7].

Let P be a disjunctive logic program, A be a set of literals. A set of atoms X is an *unfounded set* for P w.r.t. 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 w.r.t. 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 w.r.t. A.

Note that if P is a normal logic program, unfounded sets defined here coincide with the definition given for normal logic programs in [6]. For normal logic programs, the union of all unfounded sets w.r.t. A is also an unfounded set w.r.t. 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 w.r.t. A does not exist. From Proposition 3.7 in [7], 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 w.r.t. 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 w.r.t. A.

Loops and unfounded sets are closely related [8,9]. In this paper, we show that the greatest unfounded sets (if 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 $unit\_clause(\Gamma)$ 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 Loops with at Most One External Support

The basic theorem about loop formulas says that a set of atoms is an answer set of a logic program iff it is a model of the program's completion and loop formulas[1]. This is the case for normal logic programs [10] as well as disjunctive logic programs [4]. This means that a sentence is a consequence of a logic program iff 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 infinite number of loops and loop formulas. In [1], Chen *et al.* considered using unit propagation as the inference rule, and some special classes of loops whose loop formulas can be computed efficiently. In general terms, their 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 loop formulas.
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.

They showed that when $\vdash_P$ is unit propagation, and the class of loops under U is those that have no external support under U, then the above procedure basically computes the well-founded model for normal logic programs. They also considered loops with at most one external support and showed that their loop formulas can be computed efficiently.

Our main objectives are to extend these results to disjunctive logic programs. We consider first these loops can be computed in disjunctive logic programs.

3.1 Loops with No External Support

It is easy to see that if a loop L has no external support rules, i.e. $R^-(L) = \emptyset$, then its loop formula (5) is equivalent to $\bigwedge_{p \in L} \neg p$, if L has only one external support rule, i.e. $R^-(L) = \{r\}$, then its loop formulas (5) is equivalent to

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

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

which is equivalent to a set of binary clauses.

More generally, if we already know that A is a set of literals that are true in all answer sets, then for any loop L that has at most one external support rule whose body is *active* under A w.r.t. L, its loop formula is still equivalent to either a set of literals or a set of binary clauses under A. A rule r is *active* under A w.r.t. 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 w.r.t. L. In the following, we denote by $R^-(L, A)$ the set of external support rules of L under A. Note that if P is a normal logic program, $R^-(L, A)$ defined here coincides with the same notion defined in [1].

Now given a disjunctive logic program P and a set A of literals, let

$$\begin{aligned} \mathit{loop}_0(P, A) &= \{\, L \mid L \text{ is a loop of } P \text{ such that } R^-(L, A) = \emptyset \,\}, \\ \mathit{floop}_0(P, A) &= \{\, \neg a \mid a \in L \text{ for a loop } L \in \mathit{loop}_0(P, A) \,\}. \end{aligned}$$

Then $\mathit{loop}_0(P, A)$ is the set of loops that do not have any external support rules under A, and $\mathit{floop}_0(P, A)$ 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 $\mathit{floop}_0(P, \emptyset)$.

We now consider how to compute $\mathit{floop}_0(P, A)$. For normal logic programs, Chen *et al.* showed that $\mathit{floop}_0(P, A)$ can be computed in quadratic time. However, for disjunctive logic programs, the problem is NP-hard in the general case.

Proposition 1. *Given a disjunctive logic program P, a set A of literals, and an atom a, deciding whether $\neg a \in \mathit{floop}_0(P, A)$ is NP-complete.*

Fortunately, if the set A is unfounded-free[2], then $\mathit{floop}_0(P, A)$ 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 above when $\vdash_P$ is unit propagation and the class of loops is that of loops without external support.

Our algorithm below for computing $\mathit{floop}_0(P, A)$ is similar to the corresponding one in [1], and is through maximal loops.

Let $ml_0(P, A)$ be the set of maximal loops that do not have any external support rules under A: 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' such that $L \subset L'$. Clearly,

$$\mathit{floop}_0(P, A) = \bigcup_{L \in ml_0(P, A)} \overline{L}.$$

If P is a normal logic program, loops in $ml_0(P, A)$ are pair-wise disjoint [1]. For disjunctive logic programs the property is not true in general, thus the reason that $\mathit{floop}_0(P, A)$ is intractable. However, if A is unfounded-free, then loops in $ml_0(P, A)$ are pair-wise disjoint. This follows from the following proposition:

[2] Recall that A is unfounded-free if $A \cap X = \emptyset$ for each unfounded set X of P w.r.t. A.

Proposition 2. *Let P be a disjunctive logic program, 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 2 is necessary.

Example 1. Consider the following disjunctive logic program P:

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

Let $A = \{b, c\}$, $L_1 = \{a, b\}$ and $L_2 = \{a, c\}$. L_1 and L_2 are 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:

Function $ML_0(P, A, S)$: P a 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

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

Theorem 1. *Let P be a disjunctive logic program, A and S 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. *$ML_0(P, A, Atoms(P)) \subseteq loop_0(P, A)$.*
3. *If A is unfounded-free, then $ML_0(P, A, Atoms(P)) = ml_0(P, A)$.*

3.2 Loops with at Most One External Support

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

$$loop_1(P, A) = \{\, L \mid L \text{ is a loop of } P \text{ such that } R^-(L, A) = \{r\} \,\},$$
$$floop_1(P, A) = \{\neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L}, \text{ for some loop } L \text{ and rule } r \text{ such that } R^-(L, A) = \{r\} \,\}.$$

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.

Like $floop_0(P, A)$, $floop_1(P, A)$ is intractable.

Proposition 3. *Given a disjunctive logic program P, a set A of literals, an atom a, and a literal l, deciding whether $\neg a \vee l \in \mathit{floop}_1(P, A)$ is NP-complete.*

While there is a polynomial algorithm for computing $\mathit{floop}_0(P, A)$ when A is unfounded-free, this is not the case for $\mathit{floop}_1(P, A)$. Proposition 3 holds even when we restrict A to be unfounded-free.

Notice that for normal logic programs, the complexity of $\mathit{floop}_1(P, A)$ is left as an open question in [1]. Instead, a polynomial algorithm is proposed for computing $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$[3] which corresponds to the set of loop formulas of loops with at most one external support [1]. For disjunctive logic programs, $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$ is still intractable even when A is unfounded-free[4].

Given this 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 $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$ as possible. We propose one such approximation algorithm below. 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 $ML_0(P, A, S)$ when A is unfounded-free (Theorem 1).

Proposition 4. *For any disjunctive logic program P and a set A of literals that is unfounded-free for P. $\mathit{floop}_0(P, A)$ and $\mathit{floop}_1(P, A)$ imply the following theory*

$$\bigcup_{\overline{A} \cap body(r) = \emptyset, L \in ML_0(P \setminus \{r\}, A, Atoms(P \setminus \{r\}))} \{\neg a \vee l \mid a \in L, l \in body(r) \cup \overline{head(r) \setminus L}\}. \tag{6}$$

In the following, we use $\mathit{fLoop}_1(P, A)$ to denote (6). According to Proposition 2 of [1], if P is a normal logic program, then $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$ is equivalent to $\mathit{floop}_0(P, A) \cup \mathit{fLoop}_1(P, A)$ for any A. However, for disjunctive logic programs, this two theories are not equivalent, even when A is unfounded-free, as the following example illustrates.

Example 2. Consider the following logic program P:

$$a \vee b \vee c \leftarrow d. \qquad a \leftarrow b, c. \qquad b \leftarrow a. \qquad c \leftarrow b.$$

Let $A = \emptyset$, $\mathit{loop}_1(P, A) = \{ \{a, b, c\}, \{a, b\} \}$, both loops have one external support rule: $a \vee b \vee c \leftarrow d$, thus $\neg a \vee \neg c, \neg b \vee \neg c \in \mathit{floop}_1(P, A)$, but they can not be computed from $\mathit{fLoop}_1(P, A)$.

[3] Not exactly this set, but $\mathit{floop}_0(P, A) \cup \mathit{fLoop}_1(P, A)$, which is logically equivalent to $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$, and especially $UP(\mathit{floop}_0(P, A) \cup \mathit{fLoop}_1(P, A)) = UP(\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A))$.

[4] For normal logic programs, we need to compute $T(P, A)$, we show that $\mathit{floop}_0(P, A) \supset (\mathit{floop}_1(P, A) \equiv \mathit{fLoop}_1(P, A))$ and furthermore $UP(\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)) = UP(\mathit{floop}_0(P, A) \cup \mathit{fLoop}_1(P, A))$. For disjunctive logic programs, deciding whether a literal $l \in T(P, A)$ or even $l \in UP(\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A))$, A is unfounded-free, is NP-hard.

So to summarize, while we can not efficiently compute $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$, we can compute $\mathit{floop}_0(P, A) \cup \mathit{fLoop}_1(P, A)$ which is still helpful for computing consequences of a logic program. To compute $\mathit{floop}_0(P, A) \cup \mathit{fLoop}_1(P, A)$, we first call $ML_0(P, A, \mathit{Atoms}(P))$, and then for each proper rule $r \in P$ such that $\overline{A} \cap \mathit{body}(r) = \emptyset$, we call $ML_0(P \setminus \{r\}, A, \mathit{Atoms}(P \setminus \{r\}))$. The worse case complexity of this procedure is $O(n^3)$, where n is the size of P.

4 Computing Consequences of a Program

Let's now return to the iterative procedure given in the beginning of last section. When $\vdash_P$ is unit propagation UP, and the loop formulas are those from ML_0 (maximal loops with no external support), it becomes the following one:

Function $T_0(P)$ - P is a disjunctive logic program;
$X := \emptyset$; $Y := \mathit{comp}(P) \cup \{\text{loop formulas of loops in } ML_0(P, \emptyset, \mathit{Atoms}(P))\}$;
while $X \neq UP(Y)$ **do**
    $X := UP(Y)$; $Y := Y \cup \{\text{loop formulas of loops in } ML_0(P, X, \mathit{Atoms}(P))\}$;
return $X \cap \mathit{Lit}(P)$.

Clearly $T_0(P)$ runs in polynomial time and returns a set of consequences of P. It is also 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 1, if P has at least one answer set, then at each iteration, the set of literals added to Y, $\{\text{loop formulas of loops in } ML_0(P, X, \mathit{Atoms}(P))\}$, equals to $\mathit{floop}_0(P, X)$, the set of loop formulas with no external support under X.

Proposition 5. *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 $\mathit{floop}_0(P, A) \cup \mathit{floop}_1(P, A)$, we get the following procedure:

Function $T^1(P)$ - P is a disjunctive logic program;
$Y := \mathit{comp}(P) \cup \mathit{floop}_0(P, \emptyset) \cup \mathit{floop}_1(P, \emptyset)$; $X := \emptyset$;
while $X \neq UP(Y)$ **do**
    $X := UP(Y)$; $Y := Y \cup \mathit{floop}_0(P, X) \cup \mathit{floop}_1(P, X)$;
return $X \cap \mathit{Lit}(P)$.

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. As we have shown in the last section, even for unfounded-free A, computing $\mathit{floop}_0(P, X) \cup \mathit{floop}_1(P, X)$ is intractable. Thus we cannot show that the above 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.

Proposition 6. *For any disjunctive logic program P, deciding whether a literal is in $T^1(P)$ is NP-hard.*

In the last section, we propose to use $fLoop_1(P, A)$ 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;
$Y := comp(P) \cup floop_0(P, \emptyset) \cup fLoop_1(P, \emptyset)$; $X := \emptyset$;
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 6 for details.

5 Related Work

Here we relate our work to the preprocessing procedure in DLV [11] and the well-founded semantics of disjunctive programs proposed by Wang and Zhou [2].

5.1 DLV Preprocessing Operator

We now show that $T_0(P)$ coincides with the least fixed point of the operator $\mathcal{W}_\mathcal{P}$ used in DLV for preprocessing a given disjunctive logic program. First, we show that the greatest unfounded set of a disjunctive logic program (if 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. The function $M(P, A)$, the least fixed point of the operator M_P^A defined as follows:

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

Theorem 2. *For any disjunctive logic program P and any $A \subseteq Lit(P)$ such that the greatest unfounded set of P w.r.t. A exists. $M(P, A) = GUS_P(A)$.*

From the above 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 $loop_0^A(P, X) = \bigcup_{L \in ML_0(P, A \cup \overline{X}, Atoms(P))} L$, 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, avoiding the computation of the complement of it.

Now we introduce the $\mathcal{W}_\mathcal{P}$ operator proposed in [7].

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

From Proposition 5.6 in [7], $\mathcal{W}_\mathcal{P}$ has a least fixed point, denoted $\mathcal{W}_\mathcal{P}^\omega(\emptyset)$, is the consequence of the program. $\mathcal{W}_\mathcal{P}^\omega(\emptyset)$ can also be computed efficiently, thus it is considered as a good start point to 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, 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. [12]).

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

Theorem 3. *For any disjunctive logic program P, $\mathcal{W}_\mathcal{P}^\omega(\emptyset) \subseteq T_0(P)$. If P is simplified and without constraints, then $\mathcal{W}_\mathcal{P}^\omega(\emptyset) = T_0(P)$.*

Note that, [7] proved that, if P does not contain constraints, $\mathcal{W}_\mathcal{P}^\omega(\emptyset)$ 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, 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 form (1) by the k rules

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

It is worth to note 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 3. Consider the following logic program P:

$$d \leftarrow not\ e. \qquad e \leftarrow not\ d. \qquad a \vee c \leftarrow d. \qquad a \vee b \leftarrow e.$$
$$a \leftarrow b. \qquad b \leftarrow a. \qquad \leftarrow not\ a. \qquad \leftarrow not\ b.$$

Clearly, $\{a, b, d\}$ and $\{a, b, e\}$ are the only two answer sets of P, $\{a, b, d\}$ is the only answer set of $sh(P)$. Let $A = \{a, b\}$, A is unfounded-free for P and $sh(P)$. $\neg a \vee d \in fLoop_1(sh(P), A)$ which is false for $\{a, b, e\}$, 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, s.t. $a, b \in L$ and $a, b \in head(r)$. If P is head-cycle free, then a set of atoms is an answer of P iff it is an answer set of $sh(P)$.

Proposition 7. *For any head-cycle free disjunctive logic program P and a set A of literals:*

$$floop_0(P, A) = ML_0(P, A, Atoms(P)) = ML_0(sh(P), A, Atoms(P)),$$
$$fLoop_1(P, A) = \bigcup_{\overline{A} \cap body(r) = \emptyset, L \in ML_0(sh(P \setminus \{r\}), A, Atoms(P))} \{\neg a \vee l \mid a \in L, \; l \in body(r) \cup \overline{head(r) \setminus L}\},$$

and $floop_0(P, A)$ implies that $floop_1(P, A)$ is equivalent to $fLoop_1(P, A)$.

5.2 Wang and Zhou's Well-Founded Semantics for Disjunctive Logic Programs

It is proved in [1] 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 [7,13,2]. It is interesting that with a slight change of unit propagation, the procedure computes the same results as the well-founded semantics proposed in [2]. 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 $body(r)$ of $r \in P$ is *true* w.r.t. S, denoted $S \models body(r)$, if $body(r) \subseteq S$; $body(r)$ is *false* w.r.t. 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 extend the notion of unfounded set to under a set of pure disjunctions. Let S be a set of pure disjunctions of a disjunctive logic program P, a set of atoms X is an *unfounded set* for P w.r.t. 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 w.r.t. 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, $S \models (head(r) - X)$ 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 above definition is equivalent to the definition in Preparation. If P has the greatest unfounded set w.r.t. 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:

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

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

From [2], the operator $\mathcal{W}'_P$ always has the least fixed point, denoted by $lfp(\mathcal{W}'_P)$, and the well-founded semantics $U\text{-}WFS$ is defined as $U\text{-}WFS(P) = lft(\mathcal{W}'_P)$.

Now we extend T_0 to treat about 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 w.r.t. 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 w.r.t. 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

$$\mathit{floop}_0(P, S) = \{\, \neg a \mid a \in L \text{ for a loop } L \text{ of } P \text{ such that } R^-(L, S) = \emptyset \,\}.$$

Then $\mathit{floop}_0(P, S)$ 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 above definition of floop_0 is equivalent to the definition in Section 3.

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

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

where $\mathit{positve\_clause}(\Gamma)$ returns the union of all positive clauses (disjunctions) in Γ, let A is the union of all unit clauses in S, then $\mathit{assign}(S, \Gamma)$ is $\{\, c \mid$ for some $c' \in \Gamma, c' \cap A = \emptyset,$ 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:

$$T_0^*(P, S) = UP^*(\mathit{comp}(P) \cup S \cup \mathit{floop}_0(P, S)) \cap DB(P),$$

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

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

Theorem 4. *For any disjunctive logic program P, $U\text{-}WFS(P) \subseteq T_0^*(P)$. If P is simplified and without constraints, then $U\text{-}WFS(P) = T_0^*(P)$.*

6 Some Experimental Results

We have implemented a program that for any given disjunctive logic program P, it first computes $T_1(P)$, and then adds $\{\, \leftarrow \overline{l} \mid l \in T_1(P) \,\}$ to P.

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 (HC) problem,[5] and consider graphs with the same structure proposed in [1]. Specifically, we create some copies of a complete graph, and then randomly add some arcs to connect these copies into a strongly connected graph such that any HC for this graph must go through these special arcs. None of these "must in" arcs can be computed using the $\mathcal{W}_\mathcal{P}$ 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.

[5] From the website of DLV,
http://www.dbai.tuwien.ac.at/proj/dlv/examples/hamcycle

Table 1. Run-time Data for cmodels and DLV

Problem	cmodels	cmodels$_{T_1}$	DLV	DLV$_{T_1}$	T_1	Problem	cmodels	cmodels$_{T_1}$	DLV	DLV$_{T_1}$	T_1
10x10.1	58.29	22.01	43.96	1.04	25.65	9x11.1	>1h	24.05	384.83	1.32	25.71
10x10.2	227.52	22.86	43.90	1.04	24.63	9x11.2	>1h	24.04	385.07	1.32	26.34
10x10.3	361.62	21.77	43.09	1.03	24.46	9x11.3	959.18	24.51	389.44	1.34	27.05
10x10.4	447.36	28.98	44.16	1.05	24.50	9x11.4	797.76	23.29	385.89	1.33	26.07
10x10.5	66.62	21.19	1.28	1.05	21.63	9x11.5	1276.01	21.64	391.15	1.33	26.00
10x10.6	344.12	21.20	43.97	1.03	24.92	9x11.6	1339.06	27.19	1.79	1.34	22.76
10x10.7	289.98	21.32	43.78	1.03	24.80	9x11.7	206.85	23.58	386.94	1.31	25.83
10x10.8	508.95	21.63	43.42	1.04	25.45	9x11.8	2803.17	22.89	389.94	1.34	25.85
10x10.9	246.04	20.86	44.11	1.03	24.87	9x11.9	1837.58	20.70	1.79	1.29	26.16
10x10.10	1481.17	20.78	44.24	1.05	25.45	9x11.10	>1h	21.76	385.01	1.34	27.01

Table 1 contains the running times for these programs.[6] In this table, MxN.K stands for a graph with M copies of the complete graph with N nodes: $C_1, ..., C_M$, 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 $1 \leq i \leq M$ (C_{M+1} is defined to be C_1). The extension K stands for a specific way of adding these arcs. The numbers under "cmodels$_{T_1}$" and "DLV$_{T_1}$" refer to the run times (in seconds) of cmodels (version 3.77 [14]) and DLV (Oct 11 2007 [11]) when the results from $T_1(P)$ are added to the original program as constraints, and those under "T_1" are the run times of our program for computing $T_1(P)$. As can be seen, information from $T_1(P)$ makes cmodels and DLV run much faster when looking for an answer set. In addition to cmodels, we also tried claspD [15], which is very fast on these programs, on average it returned a solution in a few seconds.

7 Conclusion

We have extended the work of Chen *et al.* [1] on computing loops with at most one external support from normal logic programs to disjunctive logic programs. Our main results are that 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. 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 can be done by the preprocessing step of DLV.

For future work, we plan to conduct more experiments with our algorithms and to consider more effective ways of using consequences of a logic program.

[6] Our experiments were done on an AMD Athlon(tm) 64 X2 Dual Core Processor 3600+ and 1GB RAM. The reported times are in CPU seconds as reported by Linux "/usr/bin/time" command.

Acknowledgments. This work has been supported in part by the Natural Science Foundations of China under grants 60745002, 60573009, and 60703095, the National High-Tech Project under grant 2008AA01Z150, and by the Hong Kong RGC CERG 616806. We thank Yisong Wang for useful discussions related to the topic of this paper.

References

1. Chen, X., Ji, J., Lin, F.: Computing loops with at most one external support rule. In: Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning, pp. 401–410 (2008)
2. Wang, K., Zhou, L.: Comparisons and computation of well-founded semantics for disjunctive logic programs. ACM Trans. Comput. Logic 6(2), 295–327 (2005)
3. Lifschitz, V., Tang, L., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence 25(3), 369–389 (1999)
4. Lee, J., Lifschitz, V.: Loop formulas for disjunctive logic programs. In: Proceedings of the 19th International Conference on Logic Programming, pp. 451–465 (2003)
5. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Databases, pp. 293–322. Plenum Press, New York (1978)
6. Van Gelder, A.: The alternating fixpoint of logic programs with negation. In: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pp. 1–10. ACM, New York (1989)
7. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. Information and Computation 135(2), 69–112 (1997)
8. Lee, J.: A model-theoretic counterpart of loop formulas. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 503–508 (2005)
9. Anger, C., Gebser, M., Schaub, T.: Approaching the core of unfounded sets. In: Proceedings of the International Workshop on Nonmonotonic Reasoning (NMR 2006), pp. 58–66 (2006)
10. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artificial Intelligence 157(1-2), 115–137 (2004)
11. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
12. Lin, F., Chen, Y.: Discovering Classes of Strongly Equivalent Logic Programs. Journal of Artificial Intelligence Research 28, 431–451 (2007)
13. Brass, S., Dix, J.: Semantics of (disjunctive) logic programs based on partial evaluation. The Journal of Logic Programming 40(1), 1–46 (1999)
14. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. J. Autom. Reasoning 36(4), 345–377 (2006)
15. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-driven disjunctive answer set solving. In: Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning, pp. 422–432 (2008)