Eliminating Disjunctions in Answer Set Programming by Restricted Unfolding

Jianmin Ji[1], **Hai Wan**[2*], **Kewen Wang**[3], **Zhe Wang**[3], **Chuhan Zhang**[2], and **Jiangtao Xu**[2]
[1]School of Computer Science and Technology,
University of Science and Technology of China, Hefei, China
jianmin@ustc.edu.cn
[2]School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
wanhai@mail.sysu.edu.cn
[3]School of Information and Communication Technology, Griffith University, Griffith, Australia
{k.wang,zhe.wang}@griffith.edu.au

Abstract

A disjunctive logic program under the answer set semantics can be equivalently translated to a normal logic program by the shifting transformation, if the program is head-cycle-free. In this paper, we provide an answer-set-preserving rewriting of a general disjunctive program to a normal program by first applying the unfolding transformation on atoms that prevent the program from being head-cycle-free, then shifting the resulting program. Different from other transformations that eliminate disjunctions in answer set programming, the new rewriting is efficient for "almost" head-cycle-free programs, *i.e.*, programs that have only a few atoms that prevent them to be head-cycle-free. Based on the new rewriting, we provide an anytime algorithm to compute answer sets of a disjunctive program by calling solvers for normal logic programs. The experiment shows that the algorithm is efficient for some disjunctive programs. We also extend the rewriting to non-ground answer set programs on finite structures.

1 Introduction

Disjunctive logic programs extend normal logic programs by permitting disjunctions to appear in rule heads, which increases the expressive power of logic programs under the answer set semantics [Baral, 2003] and brings computational penalty as well. In particular, deciding whether a disjunctive program has an answer set is Σ_2^P-complete [Eiter and Gottlob, 1995] while deciding whether a normal program has an answer set is NP-complete. In practice, significant performance difference can be observed after adding disjunctive rules into normal programs, even if only a very small number of disjunctive rules are involved. Meanwhile, it is arguable that many real-life problems can be characterized by programs consisting of a large number of normal rules and a few disjunctive rules. Hence, an interesting question is whether such a disjunctive program can be efficiently rewritten to an equivalent normal program under the answer set semantics.

Ben-Eliyahu and Dechter [1994] identified a class of disjunctive programs, called "Head-Cycle-Free" (HCF), and showed that each HCF program can be converted into an equivalent normal program in polynomial time by shifting head atoms into the body. The class of HCF programs is further generalised to that of Head-Elementary-loop-Free (HEF) programs [Gebser *et al.*, 2011], which can be transformed into normal programs in polynomial time by shifting too. One could identify another proper super class of HEF programs that are polynomial time convertible to normal programs but it would be getting harder to check if a given disjunctive program is in such a class [Ji *et al.*, 2015].

On the other hand, it is well known that a disjunctive program can be equivalently transformed into a negative disjunctive program by a set of program transformations including the unfolding transformation [Brass and Dix, 1997; Eiter and Wang, 2008]. Since shifting preserves the answer set semantics for negative disjunctive programs, each disjunctive program can be equivalently transformed into a normal program [Zhou, 2014]. Given that the unfolding transformation is exponential in the worst case, an exponential blow-up may occur in the unfolding-based rewriting. One source for this blow-up is from that unfolding can be applied *on all atoms* in the given disjunctive program. However, it has not been explored how the efficiency of unfolding can be improved in the context of disjunctive answer set programming. Especially, it is unclear if the unfolding transformation can be applied *on only some atoms* that are necessary.

A natural idea is to apply unfolding on atoms of the program one by one until the resulting program is HCF. However, unfolding may introduce new head-cycles and increase the set of atoms to be unfolded. Hence, generally speaking, one may need to unfold at all the atoms to achieve an HCF program, which indeed results in a negative program as in [Zhou, 2014]. In this paper, we show that we can restrict unfolding to "culprit" atoms, *i.e.*, atoms that prevent the initial program to be HCF, and ignore new atoms that prevent the resulting program to be HCF. We prove that the result of our restricted unfolding, although not necessarily HCF, can be equivalently transformed to a normal program via shifting.

*Corresponding author

This result is of both theoretical and practical interests—it not only shows that the complexity penalty is closely tied to these "culprit" atoms, but also suggests the practicality of our rewriting approach on disjunctive programs that are "almost" HCF, *i.e.*, with only a small number of "culprit" atoms. We make the following contributions:

- We provide an answer-set-preserving rewriting from a disjunctive program to a normal program by only unfolding at "culprit" atoms and shifting the resulting program, which is efficient for "almost" HCF programs.
- If Π' is a normal program rewritten from a disjunctive program Π by restricted unfolding and shifting, we show that the computation of answer sets for Π' can be improved. In particular, although the rewriting may introduce new loops to Π', we only need to consider loops constructed from loops of Π for computing answer sets of Π'. This result suggests that although our rewriting to a normal logic program may lead to an exponential blow-up, there is not necessarily a blow-up in loop formulas.
- As another application of the rewriting, we present an anytime algorithm to compute answer sets of a disjunctive program Π by calling solvers for normal logic programs. In each iteration, the algorithm outputs the answer sets of the shifted program Π' obtained from Π by unfolding at a set A of atoms. The algorithm will output more answer sets of Π when a larger A is applied.
- At last, we extend the rewriting to non-ground answer set programs on finite structures. As a result, we are able to compute answer sets of some first-order disjunctive logic programs by a solver for first-order normal programs such as [Asuncion *et al.*, 2012].

2 Preliminaries

2.1 Disjunctive Logic Programs

We consider fully finite logic programs based on a propositional language $\mathcal{L}$ until Section 7. A *(disjunctive) logic program* (DLP) is a finite set of (disjunctive) rules of the form

$$a_1 \vee \cdots \vee a_k \leftarrow a_{k+1}, \ldots, a_m, \mathit{not}\, a_{m+1}, \ldots, \mathit{not}\, a_n. \quad (1)$$

where $n \geq m \geq k \geq 1$ and $a_1, \ldots, a_n$ are atoms. If $k = 1$, it is a *normal rule*; if $k = m$, it is a *negative rule*; if $m = n$, it is a *positive rule*. A *normal logic program* (NLP) is a finite set of normal rules and a *negative logic program* (resp. *positive logic program*) is a finite set of negative (resp. positive) rules.

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

$$head(r) \leftarrow body(r).$$

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 $\{ a_{k+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n \}$. Given a DLP Π, we use $Atoms(\Pi)$ to denote the set of atoms occurring in Π. We use $tr(r)$ to denote the propositional formula $body(r) \supset head(r)$ and $tr(\Pi) = \bigwedge_{r \in \Pi} tr(r)$.

A set S of atoms *satisfies* a rule r if $S \cup \{\neg p \mid p \notin S\} \models tr(r)$ in the sense of propositional logic. S *satisfies* a program Π if S satisfies every rule in Π.

The *answer sets* of a DLP are defined in [Gelfond and Lifschitz, 1991]. Given a DLP Π and a set S of atoms, the Gelfond-Lifschitz reduct of Π on S, written Π^S, is obtained from Π by deleting:

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.

A set S of atoms is an *answer set* of Π if S is a minimal set satisfying Π^S. We use $AS(\Pi)$ to denote the set of answer sets of Π.

2.2 Loops and Loop Formulas

Lee and Lifschitz [2003] extended the notions of loops and loop formulas [Lin and Zhao, 2004] to DLPs. For a DLP Π, the *positive dependency graph* of Π, written G_Π, is the directed graph whose vertices are atoms in Π, and there is an arc from p to q if there is a rule $r \in \Pi$ such that $p \in head(r)$ and $q \in body^+(r)$. A set L of atoms is a *loop* of Π if the L-induced subgraph of G_Π is strongly connected. Note that, every singleton whose atom occurs in Π is also a loop of Π. We use $Loop(\Pi)$ to denote the set of loops of Π.

Given a DLP Π and a loop L of Π, a rule $r \in \Pi$ is an *external support* of L under Π if $head(r) \cap L \neq \emptyset$ and $L \cap body^+(r) = \emptyset$. Let $R^-(L, \Pi)$ be the set of external support rules of L under Π. The *(conjunctive) loop formula* of L under Π, written $LF(L, \Pi)$, is the following implication:

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

Theorem 1 (Theorem 1 in [Lee and Lifschitz, 2003]) *For a DLP Π and a set S of atoms, S is an answer set of Π iff $S \cup \{\neg p \mid p \notin S\}$ is a model of $tr(\Pi) \wedge \bigwedge_{L \in Loop(\Pi)} LF(L, \Pi)$.*

2.3 Unfolding and Shifting

We first review the unfolding transformation [Gergatsoulis, 1997; Brass and Dix, 1997].

Definition 1 (Elementary unfolding) *Let r_1, r_2 be two rules and there is an atom $a \in head(r_1) \cap body^+(r_2)$. The rule obtained by* elementary unfolding *r_2 using r_1 at a, denoted by $\mathtt{unfold}(r_1, r_2, a)$, is the rule r such that $head(r) = (head(r_1) \setminus \{a\}) \cup head(r_2)$ and $body(r) = body(r_1) \cup (body(r_2) \setminus \{a\})$.*

Definition 2 (Unfolding rules at an atom) *Let Π be a DLP and a an atom in Π. The set of rules obtained by* unfolding rules *at a in Π, denoted by $\mathtt{unfold}_\Pi(a)$, is the set $\{\mathtt{unfold}(r_1, r_2, a) \mid r_1, r_2 \in \Pi$ and $a \in head(r_1) \cap body^+(r_2)\}$.*

Let Π be a DLP and $a \in Atoms(\Pi)$. We denote

$$\mathtt{remove}(\Pi, a) = \Pi \setminus \{r \mid r \in \Pi \text{ and } a \in body^+(r)\} \cup \mathtt{unfold}_\Pi(a).$$

Example 1 *Consider the logic program* Π_1*:*

$$a \vee b \leftarrow c. \quad d \leftarrow b. \quad e \leftarrow a, d. \quad c \leftarrow e. \quad c \leftarrow .$$
$$a \leftarrow b. \quad b \leftarrow a.$$

$\texttt{unfold}_{\Pi_1}(a)$ *is the set of rules:*

$$b \vee e \leftarrow c, d. \quad e \leftarrow b, d. \quad b \leftarrow c. \quad b \leftarrow b.$$

$\texttt{remove}(\Pi_1, a) = \Pi_1 \setminus \{e \leftarrow a, d.\, b \leftarrow a.\} \cup \texttt{unfold}_{\Pi_1}(a)$.

From Theorem 13 in [Gergatsoulis, 1997], we have the following theorem.

Theorem 2 *Let* Π *be a DLP and* a *an atom in* Π*. Then* $AS(\Pi) = AS(\texttt{remove}(\Pi, a))$.

Given a DLP Π, $E \subseteq Atoms(\Pi)$, and $a \in E$, we can inductively define $\texttt{remove}(\Pi, E)$ as follows:

- $\texttt{remove}(\Pi, \{a\}) = \texttt{remove}(\Pi, a)$,
- $\texttt{remove}(\Pi, E) = \texttt{remove}(\texttt{remove}(\Pi, E \setminus \{a\}), \{a\})$.

Corollary 3 *Let* Π *be a DLP and* $E \subseteq Atoms(\Pi)$*. Then* $AS(\Pi) = AS(\texttt{remove}(\Pi, E))$.

Gelfond *et al.* [1991] provided a mapping from a DLP to an NLP by "shifting" head atoms into the bodies. We denote $\texttt{sh}(\Pi)$ to be the NLP obtained from a DLP Π by substituting every rule of form (1) with the k rules

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

It is known that every answer set of $\texttt{sh}(\Pi)$ is also an answer set of Π, but the converse is not true in general. Ben-Eliyahu and Dechter [1994] identified a class of DLP, called "Head-Cycle-Free" (HCF), and showed that the converse is true if Π is an HCF program. In particular, a DLP Π is called HCF, if $|head(r) \cap L| \le 1$ for every rule r of Π and every loop L of Π, and the answer sets of an HCF program Π coincide with the answer sets of the NLP $\texttt{sh}(\Pi)$.

Given a DLP Π, $\texttt{remove}(\Pi, Atoms(\Pi))$ is equivalent to a negative program [Brass and Dix, 1997], which is HCF. Then $AS(\Pi) = AS(\texttt{remove}(\Pi, Atoms(\Pi))) = AS(\texttt{sh}(\texttt{remove}(\Pi, Atoms(\Pi))))$. Based on this result, Zhou [2014] provided a syntactic transformation from a DLP to an NLP by first applying the unfolding transformation on all the atoms, then shifting the resulting program to the NLP.

3 Eliminating Disjunctions by Restricted Unfolding

In this section, we show that a disjunctive program can be equivalently translated to a normal program by first applying the unfolding transformation only on "culprit" atoms, *i.e.*, atoms that prevent the program from being HCF.

For a DLP Π, we denote

$$HC(\Pi) = \{a \mid \text{there exist } r \in \Pi \text{ and } L \in Loop(\Pi) \text{ such that } |head(r) \cap L| > 1 \text{ and } a \in head(r) \cap L\}.$$

Intuitively, $HC(\Pi)$ specifies the set of atoms that prevent Π to be HCF.

Now we provide the lemmas that draw a sketch of the proof for our main result.

Lemma 1 *Let* Π *be a DLP and* $E \subseteq Atoms(\Pi)$*. Then*

$$\bigwedge_{a \in E} LF(\{a\}, \Pi) \supset (tr(\Pi) \equiv tr(\texttt{remove}(\Pi, E))).$$

Lemma 2 *Let* Π *be a DLP,* $E \subseteq Atoms(\Pi)$*, and* $L \in Loop(\texttt{remove}(\Pi, E))$*. If* $L \cap HC(\Pi) = \emptyset$ *then* $LF(L, \texttt{remove}(\Pi, E)) \equiv LF(L, \texttt{sh}(\texttt{remove}(\Pi, E)))$.

As discussed in Introduction, unfolding may introduce new head-cycles and hence increase the set of atoms in HC.

Example 1 (Continued) $\{b, c, e\}$ *is not a loop of* Π_1*, while it is a loop of the program* $\texttt{remove}(\Pi_1, \{a\})$*, then* $HC(\texttt{remove}(\Pi_1, \{a\})) = \{b, e\}$ *while* $HC(\Pi_1) = \{a, b\}$.

For some DLP Π and $E \subseteq Atoms(\Pi)$, exponentially many loops can be introduced by unfolding and $HC(\texttt{remove}(\Pi, E))$ can be greatly larger than $HC(\Pi)$.

Example 2 *Consider the logic program* Π_2*:*

$$a \vee b \leftarrow c. \quad e \leftarrow a, d. \quad a \leftarrow b. \qquad b \leftarrow a.$$
$$a_1 \vee \cdots \vee a_n \leftarrow b. \quad d \leftarrow a_i. \quad (1 \le i \le n)$$

where $n \ge 1$*. Note that* $Loop(\Pi_2) = \bigcup_{p \in Atoms(\Pi_2)} \{\{p\}\} \cup \{\{a, b\}\}$*,* $HC(\Pi_2) = \{a, b\}$*.* $\texttt{remove}(\Pi_2, a) = (\Pi_2 \setminus \{e \leftarrow a, d.\, b \leftarrow a.\}) \cup \{b \vee e \leftarrow c, d.\ b \leftarrow c.\ e \leftarrow b, d.\}$*. For any nonempty set* $S \subseteq \{a_1, \ldots, a_n\}$*,* $S \cup \{b, d\}$ *is a loop of* $\texttt{remove}(\Pi_2, a)$*. Then* $|Loop(\texttt{remove}(\Pi_2, a))| = 4+n+2^n$*,* $|Loop(\Pi_2)| = 6+n$*, and* $HC(\texttt{remove}(\Pi_2, a)) = \{b, d, a_1, \ldots, a_n\}$.

Lemma 3 *Let* Π *be a DLP and* $E \subseteq Atoms(\Pi)$*. Then for each* $L \in Loop(\texttt{remove}(\Pi, E))$*, either* $L \cap E = \emptyset$ *or* $L = \{a\}$ *for some* $a \in E$.

From Theorem 1 and above lemmas, we get our main result.

Theorem 4 *For any DLP* Π*,*

$$AS(\Pi) = AS(\texttt{sh}(\texttt{remove}(\Pi, HC(\Pi)))).$$

Then we can equivalently rewrite a DLP Π to the NLP $\texttt{sh}(\texttt{remove}(\Pi, HC(\Pi)))$.

Now we discuss some optimizations for the new rewriting.

Given a DLP Π with n different atoms and m different rules, (the number of rules) $|\texttt{remove}(\Pi, a)|$ for an atom a is $m+m^2$ in the worst case, $|\texttt{remove}(\Pi, HC(\Pi))|$ is $O(m^{2n})$, and $|\texttt{sh}(\texttt{remove}(\Pi, HC(\Pi)))|$ is $O(n \cdot m^{2n})$. As proved in [Eiter *et al.*, 2004], any answer-set-preserving rewriting of a DLP to an NLP must lead in general to an exponential blow-up, providing the polynomial hierarchy does not collapse. However, when Π is "almost" HCF, the rewriting could be efficient. In particular, if $|HC(\Pi)| = k$ for a constant k, then $|\texttt{sh}(\texttt{remove}(\Pi, HC(\Pi)))|$ is $O(n \cdot m^{2k})$, which is polynomial for the size of Π, *i.e.*, n and m.

We can further reduce the number of unfolded atoms by considering only special loops.

Firstly, $HC(\Pi)$ can be replaced by one of its subsets in the rewriting. For a DLP Π, we denote

$$HC^*(\Pi) = \{a \mid \text{there exist } r \in \Pi \text{ and } L \in Loop(\Pi) \text{ such that } |head(r) \cap L| > 1, r \in R^-(L, \Pi), \text{ and } a \in head(r) \cap L\}.$$

Clearly, $HC^*(\Pi) \subseteq HC(\Pi)$. From Theorem 1, if $HC^*(\Pi) = \emptyset$, then $AS(\Pi) = AS(\mathtt{sh}(\Pi))$, even when Π is not HCF. So the class of HCF programs can be extended to HCF* programs with $HC^*(\Pi) = \emptyset$, for which shifting is answer-set-preserving. We have the following corollary of Theorem 4.

Corollary 5 *For any DLP* Π*,*

$$AS(\Pi) = AS(\mathtt{sh}(\mathtt{remove}(\Pi, HC^*(\Pi)))).$$

Secondly, based on the notions of elementary loops and proper loops, $HC^*(\Pi)$ can be further reduced to some of its subsets in the rewriting.

Gebser *et al.* [2011] showed that not all loops are necessary for identifying the answer sets among the models of a DLP. They introduced the subclass *elementary loops* of loops, and refined Theorem 1 by considering elementary loops only. Later, Ji *et al.* [2015] introduced the subclass *proper loops* of elementary loops and refined Theorem 1 by considering a special form of loop formulas for proper loops only. As recognizing elementary loops and proper loops for DLPs are coNP-complete, Ji *et al.* [2015] introduced *weak elementary loops* and *weak proper loops* which can be identified in polynomial time.

Based on these notions, we can further refine $HC^*(\Pi)$ in the rewriting. For a DLP Π, we denote

$$\begin{aligned} HEL(\Pi) = \{a \mid &\text{ there exist } r \in \Pi \text{ and an elementary} \\ &\text{loop } L \text{ of } \Pi \text{ such that } |head(r) \cap L| > 1, \\ &r \in R^-(L, \Pi), \text{ and } a \in head(r) \cap L\}. \end{aligned}$$

$HPL(\Pi)$, $HWEL(\Pi)$, and $HWPL(\Pi)$ denote the sets specified by the definition of $HEL(\Pi)$ by replacing "elementary loop" with "proper loop", "weak elementary loop", and "weak proper loop", respectively. Then we have the following corollary of Theorem 4.

Corollary 6 *For any DLP* Π*,*

$$\begin{aligned} AS(\Pi) &= AS(\mathtt{sh}(\mathtt{remove}(\Pi, HEL(\Pi)))) \\ &= AS(\mathtt{sh}(\mathtt{remove}(\Pi, HPL(\Pi)))) \\ &= AS(\mathtt{sh}(\mathtt{remove}(\Pi, HWEL(\Pi)))) \\ &= AS(\mathtt{sh}(\mathtt{remove}(\Pi, HWPL(\Pi)))). \end{aligned}$$

4 Reducing Loop Formulas for Computing Answer Sets of Rewritten Programs

From Theorem 1, answer sets can be identified from models of the program that satisfy loop formulas of all loops. In this section, we show that we only need to consider a subset of loops of an unfolded program, *i.e.*, $\mathtt{remove}(\Pi, E)$ for a DLP Π and $E \subseteq Atoms(\Pi)$, for obtaining answer sets. This result could be beneficial for SAT-based answer set programming solvers, such as ASSAT [Lin and Zhao, 2004], cmodels [Giunchiglia *et al.*, 2006], and clasp [Gebser *et al.*, 2007], for computing answer sets of $\mathtt{remove}(\Pi, E)$ and $\mathtt{sh}(\mathtt{remove}(\Pi, E))$.

We first provide a lemma to specify relations of loops and loop formulas between a DLP Π and the unfolded program $\mathtt{remove}(\Pi, a)$ for an atom a.

Lemma 4 *Let* Π *be a DLP and* a *an atom in* Π*. Then the following statements hold:*

- $\{L \mid L \in Loop(\Pi),\ a \notin L\} \subseteq Loop(\mathtt{remove}(\Pi, a))$.
- $\{L \setminus \{a\} \mid L \in Loop(\Pi)\} \setminus \emptyset \subseteq Loop(\mathtt{remove}(\Pi, a))$.
- *For any set* $L \subseteq Atoms(\Pi)$*, if* $a \in L$ *then* $LF(L, \Pi) \equiv LF(L, \mathtt{remove}(\Pi, a))$.
- *For any set* $L \subseteq Atoms(\Pi)$*, if* $a \notin L$ *then*
  - $tr(\Pi) \supset (LF(L, \mathtt{remove}(\Pi, a)) \supset LF(L, \Pi))$*, and*
  - $(LF(L \cup \{a\}, \Pi) \wedge LF(L, \Pi)) \supset LF(L, \mathtt{remove}(\Pi, a))$.

Given a DLP Π and $E \subseteq Atoms(\Pi)$, we denote

$$\begin{aligned} Loop^*(\Pi, E) = \bigcup_{a \in E} \{\{a\}\} \cup \{L \mid &\ L \cap E = \emptyset \text{ and there} \\ &\text{exists } S \subseteq E \text{ such that } L \cup S \in Loop(\Pi)\}. \end{aligned}$$

From Lemma 4, $Loop^*(\Pi, E) \subseteq Loop(\mathtt{remove}(\Pi, E))$. $\mathtt{remove}(\Pi, E)$ may have loops that are not in $Loop^*(\Pi, E)$. For some DLPs, the size of $Loop(\mathtt{remove}(\Pi, E))$ can be exponentially larger than the size of $Loop^*(\Pi, E)$.

Example 2 (Continued) $Loop^*(\Pi_2, \{a\}) = Loop(\Pi_2) \setminus \{\{a, b\}\}$ *and* $|Loop(\mathtt{remove}(\Pi_2, a)) \setminus Loop^*(\Pi_2, \{a\})| = 2^n - 1$.

The following theorem shows that, we only need to consider loops in $Loop^*(\Pi, E)$ for identifying answer sets among the models of $tr(\mathtt{remove}(\Pi, E))$.

Theorem 7 *Let* Π *be a DLP and* $E \subseteq Atoms(\Pi)$*. Then* $S \in AS(\mathtt{remove}(\Pi, E))$ *iff* $S \cup \{\neg p \mid p \notin S\}$ *is a model of*

$$tr(\Pi) \wedge \bigwedge_{L \in Loop^*(\Pi, E)} LF(L, \mathtt{remove}(\Pi, E)).$$

Recalling Theorem 1, we can improve the computation of answer sets for $\mathtt{remove}(\Pi, E)$ on SAT-based answer set programming solvers by only considering loop formulas for loops in a subset of $Loop(\mathtt{remove}(\Pi, E))$.

Corollary 8 *Let* Π *be a DLP and* $E \subseteq Atoms(\Pi)$*. Then* $S \in AS(\mathtt{sh}(\mathtt{remove}(\Pi, E)))$ *iff* $S \cup \{\neg p \mid p \notin S\}$ *is a model of*

$$tr(\Pi) \wedge \bigwedge_{L \in Loop^*(\Pi, E)} LF(L, \mathtt{sh}(\mathtt{remove}(\Pi, E))).$$

Theorem 7 and Corollary 8 show that although our transformation to NLP may result in an exponential blow-up, there is not necessarily a blow-up in loop formulas for the computation. In particular, $|Loop^*(\Pi, E)| \leq |Loop(\Pi)|$ for any set $E \subseteq Atoms(\Pi)$, then the number of loops that need to be considered for computing answer sets of $\mathtt{remove}(\Pi, E)$ or $\mathtt{sh}(\mathtt{remove}(\Pi, E))$ is not larger than that of Π.

5 Approximating a DLP by Restricted Unfolding

In this section, we consider another application of the new rewriting for approximating a DLP Π by an NLP. First, we specify the notion of approximation.

Definition 3 (Sound Approximation) *A DLP Π^* is a* (sound) approximation *of a DLP Π, if $AS(\Pi^*) \subseteq AS(\Pi)$.*

The intuition behind the definition is that, when it is hard to compute answer sets of a DLP Π, we would like to construct an approximation Π^* of Π so that Π^* can compute some answer sets of Π easier. The approximation is sound in the sense that every computed answer set is an answer set of Π.

Definition 4 (Closed Sequence of Sound Approximations) *A sequence of DLPs $\langle \Pi_1, \ldots, \Pi_n \rangle$ is a* closed sequence of (sound) approximations *of a DLP Π, if $AS(\Pi_i) \subseteq AS(\Pi_{i+1})$ for each $1 \leq i \leq n-1$, and $AS(\Pi_n) = AS(\Pi)$.*

Intuitively, given a closed sequence of sound approximations of a DLP Π, we can construct an anytime algorithm by computing answer sets of Π_i's one by one, which is expected to find better solutions the more time it keeps running. Eventually, the algorithm would return all answer sets of Π.

In the following, we specify a closed sequence of sound approximations of a DLP by restricted unfolding and shifting.

Lemma 5 *For any DLP Π and $a \in Atoms(\Pi)$, $AS(\mathtt{remove}(\mathtt{sh}(\Pi), a)) \subseteq AS(\mathtt{sh}(\mathtt{remove}(\Pi, a)))$.*

Theorem 9 *Let Π be a DLP and $E_1 \subseteq E_2 \subseteq HC(\Pi)$. Then*

$$AS(\mathtt{sh}(\Pi)) \subseteq AS(\mathtt{sh}(\mathtt{remove}(\Pi, E_1))) \subseteq AS(\mathtt{sh}(\mathtt{remove}(\Pi, E_2))) \subseteq AS(\Pi).$$

From Theorem 9, we can construct a closed sequence of sound approximations of a DLP.

Corollary 10 *Let Π be a DLP, $HC(\Pi) = \{a_1, \ldots, a_n\}$. Then*

$$\langle \mathtt{sh}(\Pi), \mathtt{sh}(\mathtt{remove}(\Pi, \{a_1\})), \mathtt{sh}(\mathtt{remove}(\Pi, \{a_1, a_2\})), \ldots, \mathtt{sh}(\mathtt{remove}(\Pi, \{a_1, \ldots, a_n\})) \rangle$$

is a closed sequence of sound approximations of Π.

Example 1 (Continued) *$AS(\mathtt{sh}(\Pi_1)) = \emptyset$ and $AS(\Pi_1) = AS(\mathtt{remove}(\Pi_1, a)) = AS(\mathtt{sh}(\mathtt{remove}(\Pi_1, a))) = \{\{a, b, c, d, e\}\}$.*

Now we provide an anytime algorithm (Algorithm 1) to compute answer sets of a DLP by using NLP solvers.

Algorithm 1: Computing answer sets of a DLP Π

```
H := HC(Π);
as := AS(sh(Π));
output as; // current computed answer sets of Π
for each a ∈ H do
    Π := remove(Π, a);
    as := AS(sh(Π));
    output as;
return as;
```

Theorem 11 *For any DLP Π, Algorithm 1 outputs a subset of $AS(\Pi)$ at each step and returns $AS(\Pi)$ at last.*

We can apply the approaches discussed in previous sections to optimize Algorithm 1 as well. In particular, we can replace $HC(\Pi)$ by $HC^*(\Pi)$ or $HWEL(\Pi)$ in the algorithm. We can also use the approach proposed in Section 4 to improve the computation of answer sets for $\mathtt{sh}(\Pi)$.

Algorithm 1 provides an approach to compute an answer set of a DLP by calling NLP solvers. Janhunen *et al.* [2006] also provided such an approach. In their approach, a DLP Π is first rewritten to an NLP $Gen(\Pi)$ to produce candidate models S, then another NLP $Test(\Pi, S)$ is constructed to check whether S is an answer set of Π. The rewriting proposed in [Janhunen *et al.*, 2006] is different from our rewriting in the sense that $AS(\Pi) \subseteq AS(Gen(\Pi))$, *i.e.*, $Gen(\Pi)$ is not a sound approximation of Π, while we provide a closed sequence of sound approximations.

6 Some Experiments

We have implemented a program[1] for rewriting a DLP P to an NLP P^n, where P^n is a simplified program from $\mathtt{sh}(\mathtt{remove}(P, HC(P)))$ by removing redundancies like rules r with $head(r) \cap body^+(r) \neq \emptyset$ or $body^+(r) \cap body^-(r) \neq \emptyset$. By Theorem 4, $AS(P) = AS(P^n)$. If the size of the NLP P^n is similar to the size of the DLP P, it will be more efficient to compute answer sets of P^n using NLP solvers than to compute answer sets of P using DLP solvers.

To corroborate this observation, we tested our program on some benchmarks constructed from Niemelä's [1999] encoding H of the Hamiltonian Circuit (HC) problem. Specifically, we constructed programs of the form $Q = Q_0 \cup H \cup G$ where H is Niemelä's encoding for the HC problem, G is the encoding of the graph for a given HC instance, and the program Q_0 is given as the following:

$$a \vee b \leftarrow reached(1). \qquad a \leftarrow b. \qquad b \leftarrow a.$$

where $reached(1)$ is an atom in H and a, b are new atoms not appearing in H.

Our program will rewrite such a disjunctive program Q into a normal program $Q^n = Q_0^n \cup H \cup G$. It can be seen that $Q^n = \mathtt{sh}(\mathtt{remove}(Q, HC(Q)))$. Here Q_0^n is the program:

$$b \leftarrow reached(1), not\ a. \qquad a \leftarrow reached(1), not\ b.$$
$$a \leftarrow reached(1). \qquad b \leftarrow reached(1).$$

In Table 1, we compare the running times of computing an answer set of Q and Q^n using solvers clasp and cmodels[2]. In the table, rand_m_k(t) stands for a class of randomly generated t graphs with m vertexes and k arcs. The numbers under "Q" refer to the average running times for computing an answer set of corresponding programs Q, using corresponding DLP solvers, *i.e.*, clasp[3] (version 3.1.4) and cmodels[4] (version 3.8.5), and those under "Q^n" refer to the average running

[1] Our implementation is available on the web http://ss.sysu.edu.cn/%7ewh/dlp2nlp.html.

[2] Our experiments were done on an Intel(R) Core(TM) i7-2600 3.40GHz CPU and 32GB RAM. The reported times are in CPU seconds as reported by Linux "/usr/bin/time" command.

[3] The modern versions of clasp have built-in support for DLPs. http://www.cs.uni-potsdam.de/clasp/

[4] http://www.cs.utexas.edu/users/tag/cmodels/

times for computing an answer set of corresponding programs Q^n, using corresponding NLP solvers, *i.e.*, clasp and cmodels. As can be seen, for these programs, computing an answer set of Q^n by NLP solvers is more efficient.

Table 1: Comparing Q and Q^n

Graphs	Q		Q^n	
	clasp	cmodels	clasp	cmodels
rand_200_1800(5)	1.47	4.25	1.14	3.12
rand_250_2200(5)	43.91	1080.47(3)	6.67	508.17(1)
rand_300_2700(5)	616.76	1081.37(3)	332.80	1080.77(3)
rand_350_3000(5)	1102.61(3)	1800(5)	1080.56(3)	1800(5)

* When a solver could not compute an answer set of a program in 1800 seconds, we count the running time as 1800 seconds. We also report the number of such programs in the parentheses.

We also tested some benchmarks that were frequently used to evaluate performance of DLP solvers [Denecker *et al.*, 2009; Gebser *et al.*, 2013]. However, for most non-HCF DLPs P in these benchmarks, the size of P^n is much larger than the size of P and as a result, our method failed to show advantage over direct use of DLP solvers.

7 Eliminating Disjunctions for First-order Programs on Finite Structures

In this section, we extend the new rewriting to non-ground programs with first-order answer set semantics on finite structures. We first review some basic notions.

In the following, we consider the first-order disjunctive logic programs [Ferraris *et al.*, 2011], with similar syntactic forms as DLP programs. A distinction is made between predicates occurring only in rule bodies and the reminder predicates. A predicate P in a first-order DLP Π is *intensional* (or IDB) if P occurs in $head(r)$ for some $r \in \Pi$, and *extensional* (or EDB) otherwise. We use $\mathcal{P}(\Pi)$, $\mathcal{P}_{\text{IDB}}(\Pi)$, and $\mathcal{P}_{\text{EDB}}(\Pi)$ to denote sets of predicates, intensional predicates, extensional predicates occurring in Π respectively. As with [Lee and Meng, 2008], we assume w.l.o.g. that Π is in *normal form* without object constants occurring in rule heads.

We use first-order (ground) substitution, unifier, most general unifier (mgu) defined in the standard way. In contrast to grounded DLP, the answer sets of a first-order DLP is defined using first-order structures. In particular, a *finite structure* $\mathcal{M}$ is a tuple $(D, c_1^{\mathcal{M}}, \ldots, c_m^{\mathcal{M}}, P_1^{\mathcal{M}}, \ldots, P_n^{\mathcal{M}})$, where D is a finite set called the *domain* of $\mathcal{M}$, $c_i^{\mathcal{M}} \in A$ (the interpretation of constant c_i) $(1 \leq i \leq m)$, and $P_j^{\mathcal{M}}$ (the interpretation of a k-ary predicate symbol P_j) $(1 \leq j \leq n)$ a k-ary relation on D. We refer the readers to [Asuncion *et al.*, 2012] for the definition of answer sets of first-order DLP on finite structures.

The answer set semantics of a first-order DLP on finite structures can be captured by first-order loop formulas [Chen *et al.*, 2006; Lee and Meng, 2008] as well.

For a first-order DLP Π, the *positive dependency graph* of Π, written G_Π, is the infinite graph (V, E), where V is the set of atoms constructed from $\mathcal{P}_{\text{IDB}}(\Pi)$, $(a, b) \in E$ if there is a rule $r \in \Pi$ and a substitution θ such that $P(\vec{t})\theta = a$ for some $P(\vec{t}) \in head(r)$ and $Q(\vec{t'})\theta = b$ for some $Q(\vec{t'}) \in body^+(r)$. A finite nonempty subset L of V is called a *(first-order) loop* of Π if the subgraph of G_Π induced by L is strongly connected. Specially, for each atom $a \in V$, $\{a\}$ is a loop of Π. We use $Loop(\Pi)$ to denote the set of loops of Π.

Given a loop L of a DLP Π in normal form, we first rename variables in Π so that no variables of Π occur in L. Then the *(first-order) external support formula* of L for Π, denoted by $ES(L, \Pi)$, is the following disjunction

$$\bigvee_{\substack{\theta, r:\, r\in\Pi,\\ head(r)\theta\cap L\neq\emptyset}} \exists\vec{y}\Bigg(body(r)\theta\wedge \bigwedge_{P(\vec{t})\in body^+(r)\theta,\, P(\vec{t'})\in L} (\vec{t}\neq\vec{t'}) \wedge\neg\Big(\bigvee_{P(\vec{t})\in head(r)\theta}\big(P(\vec{t})\wedge\bigwedge_{P(\vec{t'})\in L}(\vec{t}\neq\vec{t'})\big)\Big)\Bigg).$$

The *(first-order) loop formula* of L in Π, denoted by $LF(L, \Pi)$, is the universal closure of $\bigwedge_{a\in L} a \supset ES(L, \Pi)$. We use $LF(\Pi)$ to denote the set of all loop formulas in Π.

Theorem 12 (Proposition 4 in [Lee and Meng, 2008]) *Let Π be a DLP and $\mathcal{M}$ a finite structure. $\mathcal{M}$ is an answer set of Π if and only if $\mathcal{M}$ is a model of $\{tr(\Pi)\} \cup LF(\Pi)$.*

Given a first-order DLP Π, the shifting result $\texttt{sh}(\Pi)$ is defined in the same way as in the propositional case. Every answer set of $\texttt{sh}(\Pi)$ is also an answer set of Π, but the converse is not true in general. Similarly, a first-order DLP Π is called "Head-Cycle-Free" (HCF), if there is no loop L, rule r in Π, and substitution θ such that $|L \cap head(r)\theta| > 1$.

Based on the progression semantics of answer sets [Zhou and Zhang, 2011], Zhou [2015] provided the following proposition.

Proposition 1 *Let P be an HCF first-order DLP. Then, $AS(\texttt{sh}(\Pi)) = AS(\Pi)$.*

Definition 5 (Elementary unfolding in first-order case) *Given two rules r_1 and r_2 (with suitable variable renaming), there are atoms $a \in head(r_1)$ and $b \in body^+(r_2)$ such that a and b are unifiable with an mgu θ. The rule obtained by* elementary unfolding r_2 at b using r_1 at a, *denoted by* $\texttt{unfold}(r_1, r_2, a, b)$, *is the rule r such that $head(r) = ((head(r_1) \setminus \{a\}) \cup head(r_2))\theta$ and $body(r) = (body(r_1) \cup (body(r_2) \setminus \{b\}))\theta$.*

Let Π be a DLP, r a rule in Π, and b an atom in $body^+(r)$. We denote $U_{set}(\Pi, r, b)$ to be the set of rules constructed by the following procedure:

$S_0 := \{(r', a') \mid r' \in \Pi$, there is an atom $a' \in head(r')$ such that a' and b are unifiable $\}$;

$i := 0$;

while $S_i \neq \emptyset$

  $S_{i+1} := \emptyset$;

  $R_{i+1} := \emptyset$;

  for each $(r', a') \in S_i$ if a' and b are unifiable

    $r'' := \texttt{unfold}(r', r, a', b)$;

    $R_{i+1} = R_{i+1} \cup \{r''\}$;

$\theta :=$ the mgu that a' and b are unfiable with;
$S_{i+1} := S_{i+1} \cup$
$\{(r'', a^*\theta) \mid (r', a^*) \in S_i, a^* \neq a'\}$;
$i := i + 1$;
$U_{set}(\Pi, r, b) := R_1 \cup R_2 \cup \cdots \cup R_i$.

[Gergatsoulis, 1997] showed that the procedure would always terminate. We denote $\texttt{remove}(\Pi, r, b) = (\Pi \setminus \{r\}) \cup U_{set}(\Pi, r, b)$. From Theorem 13 in [Gergatsoulis, 1997], we have the following proposition.

Proposition 2 *Let P be a DLP, r a rule in Π, and b an atom in $body^+(r)$. Then $AS(\Pi) = AS(\texttt{remove}(\Pi, r, b))$.*

Let Π be a DLP and a an atom in $head(r)$ for some $r \in \Pi$. We denote $\texttt{remove}(\Pi, a)$ to be a set of rules constructed by the following procedure:

$R := \Pi$;
while there is a rule $r \in R$ and $b \in body^+(r)$ such that a and b are unifiable
$R := \texttt{remove}(R, r, b)$;
$\texttt{remove}(\Pi, a) := R$.

Note that, by choosing different rules and corresponding atoms in the procedure, $\texttt{remove}(\Pi, a)$ would return different programs. However, they are equivalent in the sense of having the same sets of answer sets.

The above procedure may not terminate for some DLPs due to the unfolding process. Zhou [2015] identified a class of first-order DLPs called *choice-bounded* and showed that the process of applying the unfolding transformation when it is possible would always terminate for choice-bounded DLPs. Then for a choice-bounded DLP Π, the above procedure terminates and $\texttt{remove}(\Pi, a)$ always exists.

Theorem 13 *Let Π be a DLP, a an atom in $head(r)$ for some $r \in \Pi$, and $\texttt{remove}(\Pi, a)$ exists. Then $AS(\Pi) = AS(\texttt{remove}(\Pi, a))$.*

Given a DLP Π, we denote $head(\Pi) = \{a \mid a \in head(r)$ for some $r \in \Pi\}$. Let Π be a DLP and a set $A \subseteq head(\Pi)$. We denote $\texttt{remove}(\Pi, A)$ to be a set of rules constructed by the following procedure:

$R := \Pi$;
for each $a \in A$
$R := \texttt{remove}(R, a)$;
$\texttt{remove}(\Pi, A) := R$.

Corollary 14 *Let P be a DLP, $A \subseteq head(\Pi)$, and $\texttt{remove}(\Pi, A)$ exists. Then $AS(\Pi) = AS(\texttt{remove}(\Pi, A))$.*

Now we extend our rewriting to first-order DLPs.
For a DLP Π, we denote

$$HC(\Pi) = \{a \mid a \in head(\Pi), \text{ there exists } r \in \Pi, \text{ a loop } L \text{ of } \Pi, \text{ and a substitution } \theta \text{ such that } |L \cap head(r)\theta| > 1, a \in head(r) \text{ and } a\theta \in L\}.$$

We provide the following lemma to draw a sketch of the proof for the main result.

Lemma 6 *Let Π be a DLP, $A \subseteq head(\Pi)$, and $\texttt{remove}(\Pi, A)$ exists. Then the following statements hold:*

- *A finite structure M satisfies $\mathcal{LF}(\Pi)$ and $tr(\Pi)$ if and only if M satisfies $\mathcal{LF}(\texttt{remove}(\Pi, A))$ and $tr(\texttt{remove}(\Pi, A))$, where $\mathcal{LF}(\Pi) = \{LF(\{a'\}, \Pi) \mid a' = a\theta$ for some $a \in A$ and substitution $\theta\}$.*
- *For each $L \in Loop(\texttt{remove}(\Pi, A))$, if there does not exist a substitution θ such that $L \cap HC(\Pi)\theta \neq \emptyset$, then $LF(L, \texttt{remove}(\Pi, A)) \equiv LF(L, \texttt{sh}(\texttt{remove}(\Pi, A)))$.*
- *for each $L \in Loop(\texttt{remove}(\Pi, A))$, either there does not exist a substitution θ with $L \cap A\theta \neq \emptyset$ or $L = \{a\}$ for some $a' \in A$ and substitution σ with $a'\sigma = a$.*

Theorem 15 *Let Π be a DLP and $\texttt{remove}(\Pi, HC(\Pi))$ exists. Then $AS(\Pi) = AS(\texttt{sh}(\texttt{remove}(\Pi, HC(\Pi))))$.*

Asuncion *et al.* [2012] provided a solver of first-order NLPs by translating them to first-order sentences on finite structures. From Theorem 15, we can use the solver to compute answer sets of some first-order DLPs.

8 Conclusion

In this paper, we provide a new answer-set-preserving rewriting of a DLP to an NLP by first apply the unfolding transformation on "culprit" atoms, *i.e.*, atoms that prevent the program from being HCF, then shift the resulting program. This result shows that the complexity penalty for extending NLPs by including disjunctive rules is closely tied to these "culprit" atoms. Hence, different from other transformations for eliminating disjunctions in literature, our rewriting is efficient for "almost" HCF programs, *i.e.*, DLPs that have only a few "culprit" atoms. Later, we show that the computation of answer sets for a rewritten program can be improved by reducing the number of loops that need to be considered and the size of these loops is not larger than that of the original program. This result shows that although our transformation to NLP may result in an exponential blow-up, there is not necessarily a blow-up in loop formulas for the computation. Based on the new writing, we provide an anytime algorithm to compute answer sets of a DLP by calling NLP solvers. At last, we extend the rewriting to non-ground answer set programs on finite structures.

9 Acknowledgments

We thank Xiaoping Chen and his research group for their useful discussions. Jianmin Ji's research was partially supported by the National Natural Science Foundation of China under grant 61175057, the National Natural Science Foundation for the Youth of China under grant 61403359, as well as the USTC Key Direction Project and the USTC 985 Project. Hai Wan's research was in part supported by the National Natural Science Foundation of China under grant 61573386, Natural Science Foundation of Guangdong Province under grant 508257922126, Guangdong Province Science and Technology Plan projects under grant 509208496103, and Sun Yat-sen University Young Teachers Cultivation Project under grant 16lgpy40. This work was also partially supported by ARC under grant DP130102302.

References

[Asuncion *et al.*, 2012] Vernon Asuncion, Fangzhen Lin, Yan Zhang, and Yi Zhou. Ordered completion for first-order logic programs on finite structures. *Artificial Intelligence*, 177:1–24, 2012.

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

[Ben-Eliyahu and Dechter, 1994] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial intelligence*, 12(1-2):53–87, 1994.

[Brass and Dix, 1997] Stefan Brass and Jürgen Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *The Journal of Logic Programming*, 32(3):207–228, 1997.

[Chen *et al.*, 2006] Yin Chen, Fangzhen Lin, Yisong Wang, and Mingyi Zhang. First-order loop formulas for normal logic programs. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*, pages 298–307, 2006.

[Denecker *et al.*, 2009] Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Mirosław Truszczyński. The second answer set programming competition. In *Proceedgins of the 10th International Converence on Logic Programming and Nonmonotonic Reasoning (LPNMR-09)*, pages 637–654. Springer, 2009.

[Eiter and Gottlob, 1995] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[Eiter and Wang, 2008] Thomas Eiter and Kewen Wang. Semantic forgetting in answer set programming. *Artificial Intelligence*, 172(14):1644–1672, 2008.

[Eiter *et al.*, 2004] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. On eliminating disjunctions in stable logic programming. In *Principles of the 9th International Conference on Knowledge Representation and Reasoning (KR-04)*, pages 447–457, 2004.

[Ferraris *et al.*, 2011] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription. *Artificial Intelligence*, 175(1):236–263, 2011.

[Gebser *et al.*, 2007] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 386–392, 2007.

[Gebser *et al.*, 2011] Martin Gebser, Joohyung Lee, and Yuliya Lierler. On elementary loops of logic programs. *Theory and Practice of Logic Programming*, 11(06):953–988, 2011.

[Gebser *et al.*, 2013] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Advanced conflict-driven disjunctive answer set solving. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*, pages 912–918. AAAI Press, 2013.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New generation computing*, 9(3-4):365–385, 1991.

[Gelfond *et al.*, 1991] Michael Gelfond, Vladimir Lifschitz, Halina Przymusinska, and Miroslaw Truszczynski. Disjunctive defaults. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 230–237, 1991.

[Gergatsoulis, 1997] Manolis Gergatsoulis. Unfold/fold transformations for disjunctive logic programs. *Information processing letters*, 62(1):23–29, 1997.

[Giunchiglia *et al.*, 2006] Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

[Janhunen *et al.*, 2006] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[Ji *et al.*, 2015] Jianmin Ji, Hai Wan, and Peng Xiao. On elementary loops and proper loops for disjunctive logic programs. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 1518–1524, 2015.

[Lee and Lifschitz, 2003] J. Lee and V. Lifschitz. Loop formulas for disjunctive logic programs. In *Proceedings of the 19th International Conference on Logic Programming (ICLP-03)*, pages 451–465, 2003.

[Lee and Meng, 2008] Joohyung Lee and Yunsong Meng. On loop formulas with variables. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR-08)*, pages 444–453, 2008.

[Lin and Zhao, 2004] F. Lin and Y. Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.

[Niemelä, 1999] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3):241–273, 1999.

[Zhou and Zhang, 2011] Yi Zhou and Yan Zhang. Progression semantics for disjunctive logic programs. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*, pages 286–291, 2011.

[Zhou, 2014] Yi Zhou. From disjunctive to normal logic programs via unfolding and shifting. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-14)*, pages 1139–1140, 2014.

[Zhou, 2015] Yi Zhou. First-order disjunctive logic programming vs normal logic programming. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15)*, pages 3292–3298, 2015.