First-Order Disjunctive Logic Programming vs Normal Logic Programming

Yi Zhou
Artificial Intelligence Research Group
School of Computing, Engineering and Mathematics
University of Western Sydney, NSW, Australia

Abstract

In this paper, we study the expressive power of first-order disjunctive logic programming (DLP) and normal logic programming (NLP) under the stable model semantics. We show that, unlike the propositional case, first-order DLP is strictly more expressive than NLP. This result still holds even if auxiliary predicates are allowed, assuming $\texttt{NP} \neq \texttt{coNP}$. On the other side, we propose a partial translation from first-order DLP to NLP via unfolding and shifting, which suggests a sound yet incomplete approach to implement DLP via NLP solvers. We also identify some NLP definable subclasses, and conjecture to exactly capture NLP definability by unfolding and shifting.

1 Introduction

Answer Set Programming (ASP) has become a predominant declarative programming paradigm [Niemelä, 1999; Marek and Truszczynski, 1999] and has been used in many application domains [Baral, 2003] due to its convenience for modeling [Gebser *et al.*, 2012] and its efficient implementations such as clasp.

Although the syntax of ASP is normally presented in a first-order way, the original stable model/answer set semantics is essentially propositional as it is defined by grounding on the Hebrand universe [Gelfond and Lifschitz, 1988; 1991]. More importantly, it enforces the Unique Name Assumption and does not consider unknown objects, which is inadequate for certain application domains to reason about open worlds, e.g., the semantic Web [Eiter, 2007].

To address this issue, a number of approaches have been developed to define the stable model/answer set semantics directly on a first-order level [Bartholomew *et al.*, 2011; Bartholomew and Lee, 2013; Chen *et al.*, 2006; Denecker *et al.*, 2012; Ferraris *et al.*, 2011; Lee and Meng, 2011; Lifschitz, 2012; Lin and Zhou, 2011; Pearce and Valverde, 2004; Shen *et al.*, 2014; Zhang and Zhou, 2010]. Based on which, a new method for answer set solving has been developed [Asuncion *et al.*, 2012a; 2012b].

Similar to classical first-order logic, the answer sets of a first-order logic program are structures. Then, a fundamental problem arises what are the expressive power of different first-order ASP formalisms? In this paper, we study this problem for two of the most important fragments of first-order ASP, namely disjunctive logic programming (DLP) and normal logic programming (NLP). Clearly, NLP is a subclass of DLP. The question is the other way around, that is, whether there exists an equivalent (i.e., having the same set of answer sets) normal program for any given disjunctive program. This problem is not only of theoretical importance but also of practical relevances as it can be applied to utilizing NLP solvers for computing the answer sets of disjunctive programs.

The answer is yes in the propositional case. Eiter et al. [2004] showed that the set of stable models of any propositional disjunctive program can be captured by a normal program. Recently, Zhou [2014] proposed a syntactic translation for this purpose via unfolding and shifting.

However, the answer is no in the first-order case. In this paper, we show that first-order DLP is strictly more expressive than NLP. That is, there exists a first-order disjunctive program, which is not equivalent to any normal program over the same vocabulary.

What if we allow to introduce auxiliary predicates? Note that this is a quite common technique in the KR community. Normally, to encode a problem in ASP, one needs to use auxiliary predicates. We show that, on finite structures, DLP has the same expressive power as NLP by allowing auxiliary predicates if and only if $\texttt{NP} = \texttt{coNP}$, which is generally believed to be false in the complexity theory.

The above negative results point out that it seems impossible to translate first-order DLP to NLP. Hence, we consider to weaken the conditions. One approach is to consider partially correct translations. In this paper, we propose such a translation based on unfolding and shifting. More precisely, given a disjunctive program, we can add some unfolded rules then shift the program. We show that, the answer sets of the resulting program, which is a normal one, are also answer sets of the original disjunctive program, but not necessarily vice versa. In this sense, this method suggests a sound yet incomplete DLP solver by simply calling NLP solvers.

Another approach is to identify certain translatable subclasses. In this paper, we investigate several NLP-definable subclasses, including head-cycle-free programs and choice-bounded programs. We conjecture that, on arbitrary structures, a disjunctive program can be translated into a normal one iff this can be done via unfolding and shifting.

2 Preliminaries

We consider a first-order language with equality but without function symbols. We assume the readers are familiar with some basic notions and notations in classical first-order logic. An atom is called an *equality* atom if it is of the form $t_1 = t_2$, where t_1 and t_2 are terms. Otherwise, it is called a *proper* atom. A *substitution* is a vector $x_1/t_1 \cdots, x_n/t_n$, where $x_i, 1 \leq i \leq n$ are variables and $t_i, 1 \leq i \leq n$ are terms. A structure is called *finite* if its domain is finite. Let $\mathcal{M}$ be a structure whose vocabulary contains another vocabulary σ. The *restriction* of $\mathcal{M}$ on σ, denoted by $\mathcal{M}|\sigma$, is the σ-structure that agree with $\mathcal{M}$ on all interpretations of predicates and constants in σ. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two structures of two disjoint vocabularies σ_1 and σ_2 respectively, by $\mathcal{M}_1 \cup \mathcal{M}_2$, we denote the $\sigma_1 \cup \sigma_2$-structure that agree with $\mathcal{M}_1$ ($\mathcal{M}_2$) on all interpretations of predicates and constants in σ_1 (σ_2 resp.).

Let A be a domain. A *ground atom* on A is of the form $P(\overrightarrow{a})$, where P is a predicate and $\overrightarrow{a}$ a tuple of elements that matches the arity of P. For convenience, we also use $P(\overrightarrow{a}) \in \mathcal{M}$, or $\mathcal{M} \models P(\overrightarrow{a})$, to denote $\overrightarrow{a} \in P^{\mathcal{M}}$. Let $\mathcal{H}$ be a set of ground atoms. We use $\mathcal{H} \subseteq \mathcal{M}$, or $\mathcal{M} \models \mathcal{H}$, to denote that for every $P(\overrightarrow{a}) \in \mathcal{H}$, $\mathcal{M} \models P(\overrightarrow{a})$. For convenience, we also use $\mathcal{M} \cup \mathcal{H}$ to denote the structure obtained from $\mathcal{M}$ by extending every interpretation of predicates with the corresponding ground atoms in $\mathcal{H}$, i.e., for every P, $P^{\mathcal{M}\cup\mathcal{H}} = P^{\mathcal{M}} \cup \{\overrightarrow{a} \mid P(\overrightarrow{a}) \in \mathcal{H}\}$.

A *disjunctive rule* (*rule* for short) is of the form:

$$\alpha_1; \ldots; \alpha_m \leftarrow \beta_{m+1}, \ldots, \beta_n, \mathsf{not}\,\gamma_{n+1}, \ldots, \mathsf{not}\,\gamma_k, \quad (1)$$

where $0 \leq m \leq n \leq k$, $\alpha_i, 1 \leq i \leq m$ is a proper atom, β_j ($m+1 \leq j \leq n$) and γ_s ($n+1 \leq s \leq k$) are atoms. Let r be a rule of the form (1). The *head* of r, denoted by $Head(r)$, is $\{\alpha_1, \ldots, \alpha_m\}$; the *body* of r, denoted by $Body(r)$, is $\{\beta_{m+1}, \ldots, \beta_n, \mathsf{not}\,\gamma_{n+1}, \ldots, \mathsf{not}\,\gamma_k\}$. For convenience, we use $Pos(r)$ to denote $\{\beta_{m+1}, \ldots, \beta_n\}$, the *positive body* of r, and $Neg(r)$ to denote $\{\mathsf{not}\,\gamma_{n+1}, \ldots, \mathsf{not}\,\gamma_k\}$, the *negative body* of r respectively. A *disjunctive logic program* (*program* for short if clear from the context) is a finite set of rules. A rule is said to be *normal* if $m = 1$, and *datalog* if $m = 1$ and $k = n$. Consequently, a program is said to be *normal* (*datalog*) if all rules in it are normal (datalog resp.). Let r be a rule of the form (1) and η an assignment from variables and constants to domain elements. A *ground rule* of r on η is the following rule

$$\alpha_1\eta; \ldots; \alpha_m\eta \leftarrow \beta_{m+1}\eta, \ldots, \beta_n\eta, \mathsf{not}\,\gamma_{n+1}\eta, \ldots, \mathsf{not}\,\gamma_k\eta.$$

We distinguish between intensional and extensional predicates. A predicate P in a program Π is said to be *intensional* if P occurs in the head of some rules in Π, and *extensional* otherwise. Let Π be a program. We use $\tau(\Pi)$, $\tau_{int}(\Pi)$ and $\tau_{ext}(\Pi)$ to denote the vocabulary, the intensional vocabulary and the extensional vocabulary of Π respectively.

A number of approaches have been proposed to define the stable model/answer set semantics for disjunctive programs directly on a first-order level. Here, we briefly review the translations to second-order logic [Ferraris *et al.*, 2011] and the progress semantics [Zhou and Zhang, 2011] as they will be needed in our paper.

Given a program Π. Let $\tau_{int}(\Pi)^* = \{Q_1^*, \ldots, Q_n^*\}$ be a new set of predicates corresponding to $\tau_{int}(\Pi)$. Given a rule r in Π of the form (1), by $\widehat{r}$, we denote the universal closure of the following formula

$$\beta_{m+1} \wedge \cdots \wedge \beta_n \wedge \neg\gamma_{n+1} \wedge \cdots \wedge \neg\gamma_k \rightarrow \alpha_1 \vee \cdots \vee \alpha_m;$$

by r^*, we denote the universal closure of

$$\beta^*_{m+1} \wedge \cdots \wedge \beta^*_n \wedge \neg\gamma_{n+1} \wedge \cdots \wedge \neg\gamma_k \rightarrow \alpha^*_1 \vee \cdots \vee \alpha^*_m;$$

where $\alpha_i^* = Q_j^*(\overrightarrow{x})$ if $\alpha_i = Q_j(\overrightarrow{x}), 1 \leq j \leq n$ and

$$\beta_i^* = \begin{cases} Q^*(\overrightarrow{t}) & \text{if } \beta_i = Q(\overrightarrow{t}) \text{ and } Q \in \tau_{int}(\Pi) \\ \beta_i & \text{otherwise.} \end{cases}$$

By $\widehat{\Pi}$, we denote the first-order sentence $\bigwedge_{r\in\Pi} \hat{r}$; by Π^*, we denote the first-order sentence $\bigwedge_{r\in\Pi} r^*$.

Let Π be a program. We use $SM(\Pi)$ to denote the following second-order sentence:

$$\widehat{\Pi} \wedge \neg\exists\tau_{int}(\Pi)^*(\tau_{int}(\Pi)^* < \tau_{int}(\Pi) \wedge \Pi^*),$$

where $\tau_{int}(\Pi)^* < \tau_{int}(\Pi)$ is the abbreviation of the formula

$$\bigwedge_{1\leq i\leq n} \forall\overrightarrow{x}(Q_i^*(\overrightarrow{x}) \rightarrow Q_i(\overrightarrow{x})) \wedge \neg \bigwedge_{1\leq i\leq n} \forall\overrightarrow{x}(Q_i(\overrightarrow{x}) \rightarrow Q_i^*(\overrightarrow{x})).$$

Definition 1 *A structure $\mathcal{M}$ of $\tau(\Pi)$ is called an answer set (or a* stable model*) of Π if it is a model of $SM(\Pi)$.*

We use $AS(\Pi)$ to denote the set of answer sets of a program Π. Two programs are said to be *equivalent* if they have the same set of answer sets. Note that the second-order sentence $SM(\Pi)$ presented here is slightly different from the one in [Ferraris *et al.*, 2011], mainly because we do not consider nested expressions and quantifiers in this paper. Nevertheless, they are equivalent when restricted into first-order disjunction logic programs.

Now we turn into the progression semantics. Let S be a set and $\sigma = \{S_1, \ldots, S_t, \ldots\}$ a collection of subsets of S. A subset $H \subseteq S$ is a *hitting set* of σ if for all i, $H \cap S_i \neq \emptyset$. Furthermore, H is said to be a *minimal hitting set* of σ if H is a hitting set of σ and there is no $H' \subset H$ such that H' is also a hitting set of σ. Let Π be a program and $\mathcal{M}$ a structure of $\tau(\Pi)$. An *evolution sequence* of Π based on $\mathcal{M}$, denoted by $\sigma_{\mathcal{M}}(\Pi)$, is a sequence $\sigma^0_{\mathcal{M}}(\Pi), \sigma^1_{\mathcal{M}}(\Pi), \ldots, \sigma^t_{\mathcal{M}}(\Pi), \ldots$ of structures of $\tau(\Pi)$ such that

- $\sigma^0_{\mathcal{M}}(\Pi) = \mathcal{M}|\tau_{ext}(\Pi) \cup \mathcal{E}|\tau_{int}(\Pi)$, where $\mathcal{E}$ is the empty structure in which all predicates are interpreted as empty;
- $\sigma^{t+1}_{\mathcal{M}}(\Pi) = \sigma^t_{\mathcal{M}}(\Pi) \cup \mathcal{H}^t$, where $\mathcal{H}^t \subseteq \mathcal{M}$ and $\mathcal{H}^t$ is a minimal hitting set of the collection of

  $$Head(r)\eta,$$

  where there exists a rule r and an assignment η such that $Head(r)\eta \cap \sigma^t_M(\Pi) = \emptyset$, $\sigma^t_{\mathcal{M}}(\Pi) \models Pos(r)\eta$ and $\mathcal{M} \models Neg(r)\eta$.

An evolution sequence starts with the extensional database. In each step, the structure is expanded by a collection of ground atoms that is a minimal hitting set of the heads of ground rules applicable at the current step. Here, a ground rule is applicable at stage t if its head is not satisfied by the current structure $\sigma^t_{\mathcal{M}}(\Pi)$ in the evolution sequence, its positive body is satisfied by $\sigma^t_{\mathcal{M}}(\Pi)$ and its negative body is satisfied by the original structure $\mathcal{M}$ itself. Notice that at a certain stage, there could be many minimal hitting sets. In this sense, there could be many evolution sequences of a program Π based on a candidate structure $\mathcal{M}$. Nevertheless, if the program is normal, then there is only one minimal hitting set, which is the collection of all heads of ground rules applicable at the current stage. Hence, for normal logic programs, the progression yields a unique evolution sequence.

Definition 2 (Progression semantics) *Let Π be a program and $\mathcal{M}$ a structure of $\tau(\Pi)$. $\mathcal{M}$ is called a* stable model *(or an* answer set*) of Π iff there exists at least one evolution sequence, and for all evolution sequence σ of Π based on $\mathcal{M}$, $\sigma^\infty_{\mathcal{M}}(\Pi) = \mathcal{M}$, where $\sigma^\infty_{\mathcal{M}}(\Pi) = \bigcup_{t\geq 0} \sigma^t_{\mathcal{M}}(\Pi)$.*

The progression semantics coincides with the translation stable model semantics [Zhou and Zhang, 2011].

Example 1 [Originated from Example 2 in [Eiter *et al.*, 1997]] Consider the following program $3 - \texttt{color}$

$$\begin{array}{ll} R(x); G(x); B(x) \leftarrow, & NC \leftarrow E(x,y), R(x), R(y), \\ NC \leftarrow E(x,y), G(x), G(y), & NC \leftarrow E(x,y), B(x), B(y), \\ NC; C \leftarrow . & \end{array}$$

This disjunctive program defines the 3-colorability of a graph, whose edges are represented by $E(x,y)$. It can be checked that a stable model of $3 - \texttt{color}$ is corresponding to a 3-coloring solution of the graph, where $R(x)$, $G(x)$ and $B(x)$ mean that the node x should be colored by red, green and blue respectively, and NC means that the graph is not 3-colored.

If we replace the last rule with the following rules

$$\begin{array}{ll} R(x) \leftarrow NC, & G(x) \leftarrow NC, \\ B(x) \leftarrow NC, & NC \leftarrow \mathsf{not}\, NC. \end{array}$$

Then, the new program $\texttt{non} - 3 - \texttt{color}$ defines non-3-colorability of the graph. That is, a graph is non-3-colorable iff $\texttt{non} - 3 - \texttt{color}$ has a stable model, which must contain NC. As we shall show later in the paper, while the program $3 - \texttt{color}$ can be converted into a normal one, the program $\texttt{non} - 3 - \texttt{color}$ cannot.

3 NLP<DLP

The same as in classic first-order logic, stable models/answer sets of programs are standard first-order structures. Then, a fundamental problem arises what are the expressive power of different first-order ASP formalisms. The expressiveness problem is one of the most fundamental problems both in the logic community (particularly in model theory and finite model theory) and in the knowledge representation and reasoning community. Moreover, this problem is not only theoretically important but also practically relevant. If one formalism can be expressed by another, then we can use solvers for the latter to solve the former.

In this section, we study the expressive power between first-order DLP and NLP. Clearly, NLP is a subclass of DLP. The question is the other way around whether DLP can be expressed by NLP, more precisely, for any given disjunctive program, whether there always exists an equivalent normal program, i.e., having the same set of answer sets.

If we consider the propositional case, the answer is yes. Eiter et al. [1994] showed that, from the class of answer sets of a propositional disjunctive program, one can always construct a corresponding normal program. Nevertheless, this translation is purely semantical as the construction needs to know all answer sets in advance. Recently, Zhou [2014] proposed a syntactic translation. More precisely, by adding sufficiently enough unfolded rules into a disjunctive program, the shifted program, which is a normal one, must have the same answer sets.

Now we show that this is not the case for the first-order case. That is, there exists a first-order disjunctive program whose answer sets cannot be captured by any normal program over the same vocabulary.

Theorem 1 *NLP $<$ DLP.*

Proof: *sketch* We show that the program $\texttt{non} - 3 - \texttt{color}$ presented in Example 1 cannot be encoded into NLP over the same vocabulary. Assume that Π_0 is equivalent to $\texttt{non} - 3 - \texttt{color}$. Then, given a graph represented by the only extensional predicate E, if it is 3-colorable, then Π_0 has no answer sets, else if it is non-3-colorable, then Π_0 has exactly one answer set, whose intensional database contains NC and the full interpretations of R, G and B. The following two lemmas are needed in our proof.

Lemma 1 *Suppose that $G_1, \ldots, G_n$ are non-3-colorable graphs. There exists a non-3-colorable graph G such that for any G_i, $1 \leq i \leq n$, G_i is not a subgraph of G.*

Proof: By the Hajós construction [Jensen and Toft, 2011], there are infinite number of 4-critical graphs, which are non-3-colorable graphs but any subgraph is 3-colorable. □

Lemma 2 *Let $G_1 = \langle V_1, E_1\rangle$ and $G_2 = \langle V_2, E_2\rangle$ be two 3-colorable graphs such that $V_1 \cap V_2$ is a singleton a. Then, the union $G_1 \cup G_2 = \langle V_1 \cup V_2, E_1 \cup E_2)\rangle$ is 3-colorable as well.*

Step 1: The program obtained from Π_0 by adding the following rules is also equivalent to `non-3-color`:

$$\begin{array}{ll} NC \leftarrow \mathsf{not}\, NC, & R(x) \leftarrow \mathsf{not}\, R(x), \\ G(x) \leftarrow \mathsf{not}\, G(x), & B(x) \leftarrow \mathsf{not}\, B(x). \end{array}$$

We denote this program by Π_1.

Step 2: Let Π_2 be the program obtained from Π_1 by deleting all rules not from the above ones but whose negative parts mention some intensional predicates. Then, Π_2 is also equivalent to `non-3-color`.

A stable model $\mathcal{M}$ of a program containing the rules above must have full interpretations on all intensional predicates. For a ground rule r whose negative part mentions some intensional predicates, its negative part will never be satisfied by $\mathcal{M}$. Hence, $\mathcal{M} \models \hat{r}$. Therefore, $\mathcal{M}$ is a model of a program $\widehat{\Pi_1}$ iff $\mathcal{M}$ is a model of $\widehat{\Pi_1 \backslash \{r\}}$.

Step 3: Consider the rules in Π_2. Other than the above added rules in Step 1, Π_2 only has rules in which the negative bodies mention no intensional predicates. Let r be a ground rule of some rule in Π_2. Notice that the extensional ground atoms in the positive body of r form a graph whose nodes are the mentioned domain elements and whose edges are specified by the positive extensional ground atoms in the body of r. In this case, we say that this is the graph *represented* by r. Moreover, we say that r represents a non-3-colorable graph if the graph represented by r is non-3-colorable. Otherwise, we say that r represents a 3-colorable graph.

By Lemma 1, there exists a sufficiently large graph G_0, which is non-3-colorable and any non-3-colorable graph represented by some ground rule in Π_2 is not a subgraph of G_0. However, the structure G_0^+, which takes all edges in G_0 for the extensional predicate E and the full interpretations for all intensional predicates, is an answer set of Π_2 since G_0 is non-3-colorable. Note that for any ground rule r that represents a non-3-colorable graph, r will not be used in the progression of Π_2 based on G_0^+. In other words, all ground rules applicable in the progression of Π_2 based on G_0^+ must represent some 3-colorable graph.

Now we consider the earliest stage deriving NC in the progression of Π_2 based on G_0^+. Suppose that NC is justified by such a ground rule r_1 whose positive body contains some intensional ground atoms of the form $R(a)$ (or $G(a)$, $B(a)$) and r_1 represents a 3-colorable graph G_1. Then, those ground atoms in the positive body of r_1 will be justified by other ground rules earlier in the progression, and those ground rules represent 3-colorable graphs as well. Furthermore, the ground atoms in those rules will be justified even earlier. Then, this forms a tree, where each node is a ground atom and each edge forms a justification/dependency relationship. Nevertheless, the depth of this tree is bounded since NC is justified at a finite step in the progression. We construct a graph G_{NC} associated with this tree. Started from G_1, if one ground atom $R(a)$ in r_1 is justified by another rule r_2 with a G_2 to represent its body, then we consider the union of $G_1 \cup G_2'$, where G_2' is G_2 with all nodes except node a renamed to new constants. By Lemma 2, $G_1 \cup G_2'$ is 3-colorable as well. Hence, we obtain a finite graph G_{NC} to justify NC. By induction on the structures and Lemma 2, G_{NC} is 3-colorable since all G_i's in the rules are 3-colorable.

Similarly, we can obtain a 3-colorable graph $G_{R(a)}$ ($G_{G(a)}$ and $G_{B(a)}$ respectively) to justify an intensional ground atom $R(a)$ ($G(a)$ and $B(a)$ respectively). Finally, we construct a graph G recursively as follows. The construction starts from G_{NC}. At each step, for each node b in the current graph, we construct a graph $G_{R(b)}$ ($G_{G(b)}$ and $G_{B(b)}$ respectively), which is obtained from $G_{R(a)}$ ($G_{G(a)}$ and $G_{B(a)}$ respectively) by renaming the node a to b and other nodes to new constants. Then, we take the union of the current graph and all such newly constructed graphs together. On one side, G is a 3-colorable graph by Lemma 2. On the other side, G^+, the structure expanded form G by full interpretations on all intensional predicates, is a stable model since all intensional ground atoms can be justified, a contradiction. $\Box$

Theorem 1 is only concerned with the expressive power over the same vocabulary, which is the standard concept of expressiveness from a (finite) model theory point of view. Nevertheless, from a knowledge representation and reasoning point of view, auxiliary predicates are often allowed when talking about translations from one KR formalism to another. Using auxiliary predicates is a quite common technique in the ASP community. Normally, to encode a problem in ASP, one needs to use auxiliary predicates, e.g. the predicate $otherroute$ in the well known Hamiltonian circuit program [Niemelä, 1999].

A problem arises whether Theorem 1 still holds if auxiliary predicates are allowed. On finite structures, this is not possible providing some general assumptions in the complexity theory. As shown in the literature, while normal logic programming under the stable model semantics exactly captures the complexity class NP [Saccà, 1997; Schlipf, 1995; Marek and Remmel, 2003; Dantsin *et al.*, 2001], disjunctive logic programming under the stable model semantics exactly captures the complexity class Σ_2^P [Eiter *et al.*, 1997; Dantsin *et al.*, 2001]. Hence, on finite structures, NLP $=^*$ DLP if and only if $\mathtt{NP} = \Sigma_2^{\mathtt{P}}$ if and only if $\mathtt{NP} = \mathtt{coNP}$.

4 A Partial Translation via Unfolding and Shifting

The previous section shows that, unlike the propositional case, first-order DLP is strictly more expressive than first-order NLP, even if auxiliary predicates are allowed. Hence, it seems impossible to develop a faithful translation $\mathcal{T}$ from first-order DLP to NLP in the sense that for any disjunctive program Π, $\mathcal{T}(\Pi)$ is a normal program and $AS(\Pi) = AS(\mathcal{T}(\Pi)|\tau(\Pi))$.

In order to still utilizing NLP solvers for implementing first-order DLP, we consider to weaken the conditions. We investigate two approaches in this direction. One is to consider partially correct translations and another is to consider NLP definable subclasses of first-order DLP.

In this section, we propose a sound but incomplete translation via unfolding [Brass and Dix, 1999; Sakama and Seki, 1997] and shifting [Gelfond *et al.*, 1991; Ben-Eliyahu and Dechter, 1994]. Unfolding and shifting are two classical operators in the literature that have been widely used. Interestingly, it is only till recently to observe that these two operators can be used to fully translate propositional disjunctive programs to normal ones [Zhou, 2014]. More precisely, by unfolding sufficiently enough rules in any given propositional disjunctive program, its shifted program, which is a normal one, must have the same answer sets.

Nevertheless, according to Theorem 1, this result cannot be lifted into the first-order case. It remains an interesting question why this is the case. Let us recall the proof ideas in the propositional case, which can be divided into three assertions. First, adding any unfolded rule into a program does not affect the answer sets. Second, for programs closed under unfolding, i.e., any unfolded rule must be in the program itself, the shifted program has the same answer sets. Finally, by adding sufficiently enough unfolded rules, a propositional disjunctive program must be closed under unfolding. In this section, we shall show that, while the first two assertions remain true in the first-order case, the last one is not.

Let r_1 and r_2 be two rules, $P(\overrightarrow{t})$ an atom in $Head(r_1)$

and η a substitution such that $P(\overrightarrow{t})\eta \in Pos(r_2)$. Then, the rule obtained by *unfolding* $P(\overrightarrow{t})$ on r_1 and r_2 with respect to η, denoted by $\texttt{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta)$ is a new rule whose head is $(Head(r_1)\backslash P(\overrightarrow{t}))\eta \cup Head(r_2)$ and whose body is $Body(r_1)\eta \cup (Body(r_2)\backslash P(\overrightarrow{t})\eta)$. Let r be a rule of the form (1). The result of *shifting* r, denoted by $\texttt{Shift}(r)$, is the set of following rules $\{\alpha_i \leftarrow Body(r), \text{not}\,\{Head(r)\backslash\alpha_i\}\}$, where $1 \leq i \leq m$ and $\text{not}\,\{Head(r)\backslash\alpha_i\}$ is the set $\{\text{not}\,\alpha \mid \alpha \in Head(r), \alpha \neq \alpha_i\}$.

Example 2 Consider the following program:

$$P(x); Q(x) \leftarrow, \quad P(x) \leftarrow Q(x), \quad Q(x) \leftarrow P(x).$$

Clearly, it has a unique answer set, which contains full interpretations of both P and Q. Shifting the program, the first rule will be replaced by

$$P(x) \leftarrow \text{not}\, Q(x), \qquad Q(x) \leftarrow \text{not}\, P(x).$$

The shifted program has no answer sets. Nevertheless, if we add the unfolded rule (i.e., $P(x)$) of the first two rules w.r.t. the substitution from $\vec{x}$ to itself into the program, then the shifted program has the same answer sets.

Theorem 2 *Let* Π *be a first-order disjunctive program. Let* Π' *be a program obtained from* Π *by iteratively adding some unfolded rules. Then,* $AS(\texttt{Shift}(\Pi')) \subseteq AS(\Pi)$.

Proof: *sketch* This assertion follows from a) for all programs Π and all unfolded rules $\texttt{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta)$, $AS(\Pi) = AS(\Pi \cup \texttt{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta))$; and b) for all programs Π, $AS(\texttt{Shift}(\Pi)) \subseteq AS(\Pi)$. For a), we show that $\models SM(\Pi) \equiv SM(\Pi \cup \texttt{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta))$, which follows from the fact that $\models \widehat{\Pi} \equiv \widehat{\Pi \cup \texttt{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta)}$ and $\tau_{int}(\Pi)^* < \tau_{int}(\Pi) \models \widehat{\Pi^*} \equiv \widehat{(\Pi \cup \texttt{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta))^*}$. For b), we are able to show that $\models SM(\texttt{Shift}(\Pi)) \rightarrow SM(\Pi)$. $\Box$

Theorem 3 *Let* Π *be a first-order disjunctive program closed under unfolding. Then,* $AS(\texttt{Shift}(\Pi)) = AS(\Pi)$.

Proof: *sketch* It suffices to show that $AS(\Pi) \subseteq AS(\texttt{Shift}(\Pi))$. If Π is closed under unfolding, then for any answer set $\mathcal{M}$ of Π, there exists a unique evolution sequence σ such that $\sigma^1_{\mathcal{M}}(\Pi) = \mathcal{M} = \sigma^\infty_{\mathcal{M}}(\Pi)$. Then, it can be checked that for $\texttt{Shift}(\Pi)$, $\sigma^1_{\mathcal{M}}(\texttt{Shift}(\Pi)) = \sigma^1_{\mathcal{M}}(\Pi) = \mathcal{M} = \sigma^\infty_{\mathcal{M}}(\texttt{Shift}(\Pi))$. Hence, $\mathcal{M}$ is an answer set of $\texttt{Shift}(\Pi)$ as well. $\Box$

Theorem 2 shows that one can still use unfolding and shifting to convert a first-order disjunctive program into a normal one. However, this is just a partially correct translation. In case that the program is closed under unfolding by adding a number of unfolded rules, Theorem 3 guarantees that the converse holds as well. Nevertheless, unlike the propositional case that the program will eventually be closed under unfolding since the number of propositional rules are finite, for first-order programs, unfolding may never end.

Example 3 Consider again the program $\texttt{non}-\texttt{3}-\texttt{color}$ in Example 1. It can be shown that for any non-3-colorable graph G, the rule $NC \leftarrow Body_G$ can be obtained from $\texttt{non}-\texttt{3}-\texttt{color}$ by iteratively unfolding some rules. By Lemma 1, there are infinitely many non-3-colorable graphs. Hence, for the $\texttt{non}-\texttt{3}-\texttt{color}$ program, no matter how many unfolded rules are added, it is not closed under unfolding. Therefore, the converse of Theorem 2 does not hold for $\texttt{non}-\texttt{3}-\texttt{color}$ as there are always bigger non-3-colorable graphs that cannot be covered by the unfolded rules.

Theorem 2 enables us to develop a sound yet incomplete first-order disjunctive ASP solver by calling an NLP solver. The basic idea is to perform the unfolding operation for some rules in a given disjunctive program and then perform the shifting operation. Calling an NLP solver for the shifted program, if there exists an answer set, then, guaranteed by Theorem 2, it must be an answer set of the original disjunctive program. In case that no answer sets is found, one can obtain some information in the solving process. Based on which, one can construct new rules to be unfolded and continue the above processes. As shown in Example 3, this process might not end in the first-order case so that the procedure might be incomplete. The key issue is how to deliberately select those rules to be unfolded based on the information obtained when failing to find an answer set for the shifted program. We leave this as one of our focuses for future investigations.

5 NLP Definable Subclasses

In this section, we consider to identify some NLP definable subclasses for DLP. Of course, the class NLP itself is a trivial case. Also, all propositional disjunctive programs form another class [Eiter *et al.*, 2004; Zhou, 2014]. Pointed out by Theorem 3, the class of all disjunctive programs closed under unfolding is an NLP definable class as well.

In the literature, particularly for propositional disjunctive programs, a well known class that has attracted much attention is so-called *head-cycle-free* programs [Ben-Eliyahu and Dechter, 1994]. It is shown that, in the propositional case, all head-cycle-free programs can be encoded in NLP, in particular, by its shifted program. One can lift this into the first-order case. Let Π be a disjunctive program. The *predicate dependency graph* of Π, is a directed graph whose nodes are intensional predicates in Π, and there is an edge from a predicate P to another predicate Q iff there exists $r \in \Pi$ such that $Head(r)$ mentions P and $Body(r)$ mentions Q. A *loop* L of the program Π is a cycle in the predicate dependency graph going through all nodes in L. Then, Π is said to be *head-cycle-free* if there is no loop L and rule r such that L contains two or more occurrences of predicates mentioned in $Head(r)$. Clearly, all NLP programs are head-cycle-free.

Example 4 The program $\texttt{3}-\texttt{color}$ presented in Example 1 is head-cycle-free, while the program $\texttt{non}-\texttt{3}-\texttt{color}$ is not since $\{NC, R, G, B\}$ forms a loop that contains 3 predicates in the first disjunctive rule. The program presented in Example 2 is not head-cycle-free either.

It is shown that, in the propositional case, all head-cycle-free disjunctive programs are equivalent to their shifted programs [Ben-Eliyahu and Dechter, 1994]. This result can be lifted into the first-order case by using a similar technique. Nevertheless, head-cycle-free is a sufficient but not necessary condition for NLP definability. For instance, the program presented in Example 2 is not head-cycle-free but NLP definable. In this section, we consider whether we can characterize NLP definability for first-order disjunctive programs, similar to the notion of boundedness to characterize first-order definability for datalog programs [Ajtai and Gurevich, 1994] and NLP programs [Zhang and Zhou, 2010]. Recall the notion of boundedness for normal logic programs [Zhang and Zhou, 2010]. A normal program Π is said to be *bounded* if there exists a natural number k such that for all answer sets $\mathcal{M}$ of Π, the progression of Π based on $\mathcal{M}$ will end within k-steps. On arbitrary structures, a normal program is first-order definable if and only if it is bounded [Zhang and Zhou, 2010].

Analogously, we can extend the notion of boundedness for disjunctive programs. Naturally, a disjunctive program Π is said to be *bounded* if there exists a natural number k such that for all answer sets $\mathcal{M}$ of Π, the progression of Π based on $\mathcal{M}$ will end within k-steps. However, boundedness itself alone can capture neither first-order definability nor NLP definability for disjunctive programs. For instance, the $\mathtt{non}-\mathtt{3}-\mathtt{color}$ program presented in Example 1 is bounded within 2 steps, but it is neither first-order definable nor NLP definable over the same vocabulary.

Let us consider the differences between first-order disjunctive programs and normal programs from a progression semantics point of view. It can be observed that a key point is that, for normal programs, there is a unique evolution sequence, while for disjunctive programs, there could be many. In this sense, the number of (nondeterministic) choices in progression indeed makes a big difference. The following theorem states that if a disjunctive program only has a bounded number of choices, i.e., different evolution sequences, then it can be encoded in NLP. Let Π be a first-order disjunctive program. If there exists a number k such that for all answer sets $\mathcal{M}$ of Π, $|ES(\Pi, \mathcal{M})| < k$, where $ES(\Pi, \mathcal{M})$ is the set of all evolution sequences of Π based on $\mathcal{M}$, then we say that Π is *choice-bounded*.

Theorem 4 *Choice-bounded programs are NLP definable.*

Proof: *sketch* We first can show that if $|ES(\Pi, \mathcal{M})| = 1$ for all answer sets $\mathcal{M}$ of Π, then $AS(\Pi) = AS(\mathtt{Shift}(\Pi))$. If there is only one evolution sequence for $\mathcal{M}$ on Π, then it must coincide with the unique evolution sequence of $\mathtt{Shift}(\Pi)$ based on $\mathcal{M}$. Then, $\sigma^{\infty}_{\mathcal{M}}(\mathtt{Shift}(\Pi)) = \sigma^{\infty}_{\mathcal{M}}(\Pi) = \mathcal{M}$. Next, we show that unfolding can always reduce the number of choices. This is because by unfolding some rules in the program, one can obtain more heads in each progression step so that the number of choices are reduced. Finally, by Theorem 2, this assertion holds. □

In fact, many NLP definable classes mentioned previously belong to this class. Clearly, all NLP programs are choice-bounded with a bound to be 1. All programs closed under unfolding are choice-bounded as well. Also, all propositional disjunctive programs are choice bounded since the number of propositional variables in a program is finite. This also provides an alternative proof why all propositional disjunctive programs can be translated into normal ones. However, head-cycle-free programs are not necessarily choice-bounded.

Example 5 Consider the following program:

$$P(x); \overline{P}(x) \leftarrow, \quad Q(x); \overline{Q}(x) \leftarrow, \quad P(x) \leftarrow Q(y),$$
$$Q(x) \leftarrow P(y), \quad \overline{P}(x) \leftarrow \overline{Q}(y), \quad \overline{Q}(x) \leftarrow \overline{P}(y).$$

This program is head-cycle-free as the two predicate loops $\{P, Q\}$ and $\{\overline{P}, \overline{Q}\}$ do not go through two or more predicates in the heads of any rule. On the other side, this program is not choice-bounded as one can have an arbitrary guess of P (or Q) at the first step of progression.

Hence, head-cycle-free and choice-bounded are two disjoint NLP definable subclasses. Interestingly, both of them are highly related to the notion of unfolding and shifting. Head-cycle-free programs are equivalent to their shifted programs, while choice bounded programs can be converted to normal programs via unfolding and shifting. In addition, for first-order definability of NLP programs, unfolding also plays a key role. That is, a normal logic program is first-order definable iff it is bounded by a number k iff it is equivalent to its completion on the program by unfolding all possible rules k times [Zhang and Zhou, 2010]. Based on the evidences, we end up with our discussions with a conjecture.

Conjecture 1 *On arbitrary structures, a disjunctive program is NLP definable iff it is equivalent to the shifted program of this program expanded with some unfolded rules.*

6 Conclusion

In this paper, we have studied the expressive power of first-order DLP and first-order NLP. Interestingly, unlike the propositional case, first-order DLP is strictly more expressive than NLP (see Theorem 1). Despite these negative results, we are still able to take advantage of NLP solvers for solving first-order DLP by further weakening the conditions. For instance, we have obtained a partially correct translation from first-order DLP to NLP via unfolding and shifting (see Theorem 2). This enables us to develop a sound but incomplete DLP solver, which is one of our focuses in our future work. Also, we have identified some NLP definable subclasses of DLP, including head-cycle-free programs and choice-bounded programs (see Theorem 4). We conjecture that, on arbitrary structures, a disjunctive program is NLP definable iff this can be done via unfolding and shifting. We leave this problem open to our future investigations.

References

[Ajtai and Gurevich, 1994] Miklós Ajtai and Yuri Gurevich. Datalog vs first-order logic. *J. Comput. Syst. Sci.*, 49(3):562–588, 1994.

[Asuncion *et al.*, 2012a] Vernon Asuncion, Fangzhen Lin, Yan Zhang, and Yi Zhou. Ordered completion for first-order logic programs on finite structures. *Artif. Intell.*, 177-179:1–24, 2012.

[Asuncion *et al.*, 2012b] Vernon Asuncion, Yan Zhang, and Yi Zhou. Ordered completion for logic programs with aggregates. In *AAAI-2012*, pages 691–697, 2012.

[Baral, 2003] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[Bartholomew and Lee, 2013] Michael Bartholomew and Joohyung Lee. Functional stable model semantics and answer set programming modulo theories. In *IJCAI 2013*, 2013.

[Bartholomew *et al.*, 2011] Michael Bartholomew, Joohyung Lee, and Yunsong Meng. First-order extension of the FLP stable model semantics via modified circumscription. In *IJCAI-2011*, pages 724–730, 2011.

[Ben-Eliyahu and Dechter, 1994] Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994.

[Brass and Dix, 1999] Stefan Brass and Jürgen Dix. Semantics of (disjunctive) logic programs based on partial evaluation. *J. Log. Program.*, 40(1):1–46, 1999.

[Chen *et al.*, 2006] Yin Chen, Fangzhen Lin, Yisong Wang, and Mingyi Zhang. First-order loop formulas for normal logic programs. In *KR'06*, pages 298–307, 2006.

[Dantsin *et al.*, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[Denecker *et al.*, 2012] Marc Denecker, Yuliya Lierler, Miroslaw Truszczynski, and Joost Vennekens. A tarskian informal semantics for answer set programming. In *ICLP 2012*, pages 277–289, 2012.

[Eiter *et al.*, 1994] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Expressive power and complexity of disjunctive datalog under the stable model semantics. In *IS/KI*, pages 83–103, 1994.

[Eiter *et al.*, 1997] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[Eiter *et al.*, 2004] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. On eliminating disjunctions in stable logic programming. In *KR*, pages 447–458, 2004.

[Eiter, 2007] Thomas Eiter. Answer set programming for the semantic web. In *ICLP 2007*, pages 23–26, 2007.

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

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP'88*, pages 1070–1080. MIT Press, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[Gelfond *et al.*, 1991] Michael Gelfond, Halina Przymusinska, Vladimir Lifschitz, and Miroslaw Truszczynski. Disjective defaults. In *KR'91*, pages 230–237, 1991.

[Jensen and Toft, 2011] T.R. Jensen and B. Toft. *Graph Coloring Problems*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 2011.

[Lee and Meng, 2011] Joohyung Lee and Yunsong Meng. First-order stable model semantics and first-order loop formulas. *J. Artif. Intell. Res. (JAIR)*, 42:125–180, 2011.

[Lifschitz, 2012] Vladimir Lifschitz. Logic programs with intensional functions. In *KR 2012*, 2012.

[Lin and Zhou, 2011] Fangzhen Lin and Yi Zhou. From answer set logic programming to circumscription via logic of GK. *Artif. Intell.*, 175(1):264–277, 2011.

[Marek and Remmel, 2003] V. Wiktor Marek and Jeffrey B. Remmel. On the expressibility of stable logic programming. *TPLP*, 3(4-5):551–567, 2003.

[Marek and Truszczynski, 1999] Victor W. Marek and Miroslaw Truszczynski. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. and AI*, 25(3-4):241–273, 1999.

[Pearce and Valverde, 2004] David Pearce and Agustín Valverde. Towards a first order equilibrium logic for nonmonotonic reasoning. In *JELIA'2004*, pages 147–160, 2004.

[Saccà, 1997] Domenico Saccà. The expressive powers of stable models for bound and unbound DATALOG queries. *J. Comput. Syst. Sci.*, 54(3):441–464, 1997.

[Sakama and Seki, 1997] Chiaki Sakama and Hirohisa Seki. Partial deduction in disjunctive logic programming. *J. Log. Program.*, 32(3):229–245, 1997.

[Schlipf, 1995] John S. Schlipf. The expressive powers of the logic programming semantics. *J. Comput. Syst. Sci.*, 51(1):64–86, 1995.

[Shen *et al.*, 2014] Yi-Dong Shen, Kewen Wang, Thomas Eiter, Michael Fink, Christoph Redl, Thomas Krennwallner, and Jun Deng. FLP answer set semantics without circular justifications for general logic programs. *Artif. Intell.*, 213:1–41, 2014.

[Zhang and Zhou, 2010] Yan Zhang and Yi Zhou. On the progression semantics and boundedness of answer set programs. In *KR 2010*, 2010.

[Zhou and Zhang, 2011] Yi Zhou and Yan Zhang. Progression semantics for disjunctive logic programs. In *AAAI 2011*, 2011.

[Zhou, 2014] Yi Zhou. From disjunctive to normal logic programs via unfolding and shifting. In *ECAI 2014*, pages 1139–1140, 2014.