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 NP \neq \text{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 Truszcynski, 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 Herbrand 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 relevance 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 \text{NP} = \text{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-bound programs. We conjecture that, on arbitrary structures, a disjunctive program can be translated into a normal one if and only if this can be done via unfolding and shifting.
Given a program $\Pi$. 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 $\Pi$ of the form (1), by $\bar{r}$, 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 $\alpha_i^* = Q_j^*(\overline{t})$ if $\alpha_i = Q_j(\overline{t})$, $1 \leq j \leq n$ and
\[ \beta_i^* = \left\{ \begin{array}{ll} Q^*(\overline{t}) & \text{if } \beta_i = Q(\overline{t}) \text{ and } Q \in \tau_{int}(\Pi) \\ \beta_i & \text{otherwise.} \end{array} \right. \]
By $\hat{\Pi}$, we denote the first-order sentence $\bigwedge_{r \in \Pi} \bar{r}$; by $\Pi^*$, we denote the first-order sentence $\bigwedge_{r \in \Pi} r^*$.

Let $\Pi$ be a program. We use $SM(\Pi)$ to denote the following second-order sentence:
\[ \hat{\Pi} \wedge \neg \exists \tau_{int}(\Pi)^*(\tau_{int}(\Pi)^* < \tau_{int}(\Pi) \wedge \Pi^*), \]
where $\tau_{int}(\Pi)^* < \tau_{int}(\Pi)$ is the abbreviation of the formula
\[ \bigwedge_{1 \leq i \leq n} \forall \overline{x}(Q_i^*(\overline{x}) \rightarrow Q_i(\overline{x})) \wedge \neg \bigwedge_{1 \leq i \leq n} \forall \overline{x}(Q_i(\overline{x}) \rightarrow Q_i^*(\overline{x})). \]

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

We use $AS(\Pi)$ to denote the set of answer sets of a program $\Pi$. 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 $\sigma = \{S_1, \ldots, S_i\}$ a collection of subsets of $S$. A subset $H \subseteq S$ is a hitting set of $\sigma$ if for all $i$, $H \cap S_i \neq \emptyset$. Furthermore, $H$ is said to be a minimal hitting set of $\sigma$ if $H'$ is a hitting set of $\sigma$ and there is no $H' \subset H$ such that $H'$ is also a hitting set of $\sigma$. Let $\Pi$ be a program and $M$ a structure of $\tau(\Pi)$. An evolution sequence of $\Pi$ based on $M$, denoted by $\sigma_M(\Pi)$, is a sequence $\sigma^0_M(\Pi), \sigma^1_M(\Pi), \ldots, \sigma^k_M(\Pi), \ldots$ of structures of $\tau(\Pi)$ such that
\[ \sigma^0_M(\Pi) = M|\tau_{ext}(\Pi) \cup E|\tau_{int}(\Pi), \]
where $E$ is the empty structure in which all predicates are interpreted as empty;
\[ \sigma^{k+1}_M(\Pi) = \sigma^k_M(\Pi) \cup H^k, \]
where $H^k \subseteq M$ and $H^k$ is a minimal hitting set of the collection of $Head(r)\eta$,
where there exists a rule $r$ and an assignment $\eta$ such that $Head(r)\eta \cap \sigma^k_M(\Pi) = \emptyset$, $\sigma^k_M(\Pi) \models Pos(r)\eta$ and $M \models Neg(r)\eta$.

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...
the progression yields a unique evolution sequence. Hence, for normal logic programs, the progression yields a unique evolution sequence.

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

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-color
\[
R(x); G(x); B(x) \leftarrow, \quad NC \leftarrow E(x, y), R(x), R(y), \quad NC \leftarrow E(x, y), G(x), G(y), \quad NC \leftarrow E(x, y), B(x), B(y),
\]
\[
NC; C \leftarrow.
\]

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-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
\[
R(x) \leftarrow NC, \quad G(x) \leftarrow NC, \quad B(x) \leftarrow NC, \quad NC \leftarrow not NC.
\]

Then, the new program non-3-color defines non-3-colorability of the graph. That is, a graph is non-3-colorable iff non-3-color has a stable model, which must contain \( NC \). As we shall show later in the paper, while the program 3-color can be converted into a normal one, the program non-3-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 non-3-color presented in Example 1 cannot be encoded into NLP over the same vocabulary. Assume that \( \Pi_0 \) is equivalent to non-3-color. Then, given a graph represented by the only extensional predicate \( E \), if it is 3-colorable, then \( \Pi_0 \) has no answer sets, else if it is non-3-colorable, then \( \Pi_0 \) 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 \( V_1 \cap V_2 = \{\} \). 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 \( \Pi_0 \) by adding the following rules is also equivalent to non-3-color:
\[
NC \leftarrow not NC, \quad R(x) \leftarrow not R(x), \quad G(x) \leftarrow not G(x), \quad B(x) \leftarrow not B(x).
\]

We denote this program by \( \Pi_1 \).

**Step 2:** Let \( \Pi_2 \) be the program obtained from \( \Pi_1 \) by deleting all rules not from the above ones but whose negative parts mention some intensional predicates. Then, \( \Pi_2 \) 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, \( \mathcal{M} \models \lnot r \). Therefore, \( \mathcal{M} \) is a model of a program \( \Pi_{\mathcal{M}} \) iff \( \Pi_{\mathcal{M}} \) is a model of \( \Pi_1 \setminus \{r\} \).

**Step 3:** Consider the rules in \( \Pi_2 \). Other than the above added rules in Step 1, \( \Pi_2 \) only has rules in which the negative bodies
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 $\Pi_2$ 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 ground atoms of the form $\Pi$ defined by such a ground rule $\Pi$, is an answer set of $\Pi_2$ 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 $\Pi_2$ based on $G_0^+$. In other words, all ground rules applicable in the progression of $\Pi_2$ based on $G_0^+$ must represent some 3-colorable graph.

Now we consider the earliest stage deriving $NC$ in the progression of $\Pi_2$ 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 $G_1$’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. □

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 $\Sigma^P_2$ [Eiter et al., 1997; Dantsin et al., 2001]. Hence, on finite structures, NLP $\not\equiv$ DLP if and only if $NP = \Sigma^P_2$ and if only if $NP = coNP$.

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 $T$ from first-order DLP to NLP in the sense that for any disjunctive program $\Pi$, $T(\Pi)$ is a normal program and $AS(\Pi) = AS(T(\Pi)|\tau(\Pi))$.

In order to still utilize 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, $\overline{P(\overline{t})}$ an atom in $Head(r_1)$
and $\eta$ 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 $\eta$, denoted by $\text{Unfold}(r_1, r_2, P(\overrightarrow{t})\eta)$, is a new rule whose head is $(\text{Head}(r_1) \setminus P(\overrightarrow{t}))\eta \cup \text{Head}(r_2)$ and whose body is $\text{Body}(r_1)\eta \cup (\text{Body}(r_2) \setminus P(\overrightarrow{t})\eta)$. Let $r$ be a rule of the form \((1)\). The result of shifting $r$, denoted by $\text{Shift}(r)$, is the set of following rules \(\{\alpha_i \leftarrow \text{Body}(r), \text{not} \{\text{Head}(r)\setminus \alpha_i\}\}\), where \(1 \leq i \leq m\) and not \(\{\text{Head}(r)\setminus \alpha_i\}\) is the set \(\{\alpha \mid \alpha \in \text{Head}(r), \alpha \neq \alpha_i\}\).

**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

\[
P(x) \leftarrow \text{not} Q(x), \quad Q(x) \leftarrow \text{not} P(x).
\]

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 $\overrightarrow{x}$ to itself into the program, then the shifted program has the same answer sets.

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

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

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

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

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 is finite, for first-order programs, unfolding may never end.

**Example 3** Consider again the program\(\text{non} - 3 - \text{color}\) in Example 1. It can be shown that for any non-3-colorable graph $G$, the rule $NC \leftarrow \text{Body}_G$ can be obtained from $\text{non} - 3 - \text{color}$ by iteratively unfolding some rules. By Lemma 1, there are infinitely many non-3-colorable graphs. Hence, for the $\text{non} - 3 - \text{color}$ 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 $\text{non} - 3 - \text{color}$ 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 are 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 $\Pi$ be a disjunctive program. The predicate dependency graph of $\Pi$, is a directed graph whose nodes are intensional predicates in $\Pi$, and there is an edge from a predicate $P$ to another predicate $Q$ iff there exists $r \in \Pi$ such that $\text{Head}(r)$ mentions $P$ and $\text{Body}(r)$ mentions $Q$. A loop $L$ of the program $\Pi$ is a cycle in the predicate dependency graph going through all nodes in $L$. Then, $\Pi$ 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 $\text{Head}(r)$. Clearly, all NLP programs are head-cycle-free.

**Example 4** The program $\text{non} - 3 - \text{color}$ presented in Example 1 is head-cycle-free, while the program $\text{non} - 3 - \text{color}$ 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 first-order definable programs can be translated into normal ones. However, head-cycle-free programs are not necessarily choice-bounded.

Example 5 Consider the following program:

\[
\begin{align*}
P(x) &\leftarrow \overline{P}(x), \\
Q(x) &\leftarrow \overline{Q}(x) \leftarrow P(x) \leftarrow Q(y), \\
Q(x) &\leftarrow P(y), \\
\overline{P}(x) &\leftarrow \overline{Q}(y), \\
\overline{Q}(x) &\leftarrow \overline{P}(y).
\end{align*}
\]

This program is head-cycle-free as the two predicate loops \{P, Q\} and \{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 \(k\) if 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


