On Loop Formulas with Variables

Joohyung Lee and Yunsong Meng
Computer Science and Engineering
School of Computing and Informatics
Arizona State University, Tempe, USA
{joolee, Yunsong.Meng}@asu.edu

Abstract

Recently Ferraris, Lee and Lifschitz proposed a new definition of stable models that does not refer to grounding, which applies to the syntax of arbitrary first-order sentences. We show its relation to the idea of loop formulas with variables by Chen, Lin, Wang and Zhang, and generalize their loop formulas to disjunctive programs and to arbitrary first-order sentences. We also extend the syntax of logic programs to allow explicit quantifiers, and define its semantics as a subclass of the new language of stable models by Ferraris *et al.* Such programs inherit from the general language the ability to handle nonmonotonic reasoning under the stable model semantics even in the absence of the unique name and the domain closure assumptions, while yielding more succinct loop formulas than the general language due to the restricted syntax. We also show certain syntactic conditions under which query answering for an extended program can be reduced to entailment checking in first-order logic, providing a way to apply first-order theorem provers to reasoning about non-Herbrand stable models.

Introduction

The theorem on loop formulas showed that the stable models (answer sets) are the models of the logic program that satisfy all its loop formulas. This idea has turned out to be widely applicable in relating the stable model semantics (Gelfond and Lifschitz 1988) to propositional logic, which in turn allowed to use SAT solvers for computing answer sets. Since the original invention of loop formulas for nondisjunctive logic programs (Lin and Zhao 2004), the theorem has been extended to more general classes of logic programs, such as disjunctive programs (Lee and Lifschitz 2003), programs with classical negation and infinite programs (Lee 2005), arbitrary propositional formulas under the stable model semantics (Ferraris *et al.* 2006), and programs with aggregates (Liu and Truszczyński 2005). The theorem has also been applied to other nonmonotonic formalisms, such as nonmonotonic causal theories (Lee 2004) and McCarthy's circumscription (Lee and Lin 2006). The notion of a loop has been further refined by "elementary sets" (Gebser *et al.* 2006).

However, most work has been restricted to the propositional case. Variables contained in a program are first eliminated by grounding—the process which replaces every variable with every object constant—and then loop formulas are computed from the ground program. As a result, loop formulas were defined as formulas in propositional logic.

Chen *et al.*'s definition of loop formulas [2006] is different in that loop formulas are obtained from the original program without converting to the ground program, so that variables remain. However, since the underlying semantics of logic programs refers to grounding, such a loop formula was understood as a schema for the set of propositional loop formulas.

Recently there emerged a generalization of the stable model semantics that does not refer to grounding (Ferraris *et al.* 2007). The semantics turns a first-order sentence into a second-order sentence using the "stable model operator" SM, similar to the use of the "circumscription operator" CIRC (McCarthy 1980). Logic programs are understood as a special class of first-order sentences under the stable model semantics. Unlike the traditional stable model semantics, the new language has quantifiers with genuine object variables and the notion of first-order models instead of Herbrand models. Consequently, as in classical logic, it has no built-in unique name and domain closure assumptions.

In this paper, we study the relationship between first-order loop formulas from (Chen *et al.* 2006) and the new definition of stable models from (Ferraris *et al.* 2007). We also extend the definition of first-order loop formulas from (Chen *et al.* 2006) to disjunctive programs and to arbitrary first-order sentences, and present certain conditions under which the new second-order definition of stable models can be turned into formulas in first-order logic in the form of loop formulas.

The studied relationship helps extend the syntax of logic programs by allowing explicit quantifiers, which will be useful in overcoming the difficulties of traditional answer set programs in reasoning about the existence (or absence) of unnamed objects. We define the semantics of extended programs as a subclass of the new language of stable models from (Ferraris *et al.* 2007). Such programs inherit from the general language the ability to handle nonmonotonic reasoning under the stable model semantics even in the absence of the unique name and the domain closure assumptions. On

the other hand, extended programs yield succinct loop formulas due to the restricted syntax so that it is feasible to apply first-order theorem provers as computational engines.

Imagine an insurance policy considering "a person is eligible for a discount plan if he or she has a spouse and has no record of accident." This can be represented by the following program Π_1 that contains explicit existential quantifiers.

$$GotMarried(x, y) \leftarrow Spouse(x, y)$$
$$Spouse(x, y) \leftarrow GotMarried(x, y), not\ Divorced(x, y)$$
$$\exists w\ Discount(x, w) \leftarrow Spouse(x, y), not\ \exists z\ Accident(x, z)$$

We will say that a program Π entails a query F (under the stable model semantics) if every stable model of Π satisfies F. For example,

- Π_1 entails each of $\neg\exists xy\ Spouse(x, y)$ and $\neg\exists xy\ Discount(x, y)$.

- Π_1 conjoined with $\Pi_2 = \{\exists y\ GotMarried(marge, y)\}$, no more entails $\neg\exists xw\ Discount(x, w)$, but entails each of $\exists xw\ Discount(x, w)$ and $\forall x(Discount(x, plan1) \rightarrow x = marge)$.

- Π_1 conjoined with

  $\Pi_3 = \{Spouse(homer, marge), \exists z\ Accident(homer, z)\}$

  entails $\neg\exists w\ Discount(homer, w)$.

For the reasoning of this kind, we need the notion of non-Herbrand models since the names of discount plans, spouses and accident records may be unknown. However, answer sets are defined as a special class of Herbrand models. Instead, we will show how reasoning about non-Herbrand stable models can be represented by extended programs and can be computed using loop formulas with variables. This provides a way to apply first-order theorem provers to reasoning about non-Herbrand stable models.

The paper is organized as follows. In the next section we review the new definition of stable models from (Ferraris *et al.* 2007). Then we review first-order loop formulas from (Chen *et al.* 2006) and extend the result to disjunctive programs and to arbitrary sentences. We compare the new definition of stable models with first-order loop formulas and show certain conditions under which the former can be reduced to the latter. Given these results we give the notion of extended programs with explicit quantifiers and show how query answering for extended programs can be reduced to entailment checking in first-order logic.

Review of the New Stable Model Semantics

Let $\mathbf{p}$ be a list of distinct predicate constants $p_1, \ldots, p_n$, and let $\mathbf{u}$ be a list of distinct predicate variables $u_1, \ldots, u_n$ of the same length as $\mathbf{p}$. By $\mathbf{u} = \mathbf{p}$ we denote the conjunction of the formulas $\forall\mathbf{x}(u_i(\mathbf{x}) \leftrightarrow p_i(\mathbf{x}))$, where $\mathbf{x}$ is a list of distinct object variables of the same arity as the length of p_i, for all $i = 1, \ldots n$. By $\mathbf{u} \leq \mathbf{p}$ we denote the conjunction of formulas $\forall\mathbf{x}(u_i(\mathbf{x}) \rightarrow p_i(\mathbf{x}))$ for all $i = 1, \ldots n$, and $\mathbf{u} < \mathbf{p}$ stands for $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{u} = \mathbf{p})$.

For any first-order sentence F, $\text{SM}[F]$ stands for the second-order sentence

$$F \wedge \neg\exists\mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where $\mathbf{p}$ is the list $p_1, \ldots, p_n$ of all predicate constants occurring in F, $\mathbf{u}$ is a list $u_1, \ldots, u_n$ of distinct predicate variables of the same length as $\mathbf{p}$, and $F^*(\mathbf{u})$ is defined recursively:

- $p_i(t_1, \ldots, t_m)^* = u_i(t_1, \ldots, t_m)$;

- $(t_1 = t_2)^* = (t_1 = t_2)$;

- $\perp^* = \perp$;

- $(F \wedge G)^* = F^* \wedge G^*$;

- $(F \vee G)^* = F^* \vee G^*$;

- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;

- $(\forall x F)^* = \forall x F^*$;

- $(\exists x F)^* = \exists x F^*$.

(There is no clause for negation here, because we treat $\neg F$ as shorthand for $F \rightarrow \perp$.) According to (Ferraris *et al.* 2007), an interpretation of the signature $\sigma(F)$ consisting of the object, function and predicate constants occurring in F is a *stable model* of F if it satisfies $\text{SM}[F]$.

The terms "stable model" and "answer set" are often used in the literature interchangeably. In the context of the new language of stable models, it is convenient to distinguish between them as follows: By an *answer set* of a first-order sentence F that contains at least one object constant we will understand an Herbrand[1] interpretation of $\sigma(F)$ that satisfies $\text{SM}[F]$.

Logic programs are viewed as alternative notation for first-order sentences of special kinds (called the FOL-representation) by

- replacing every comma by $\wedge$, every semi-colon by $\vee$, and every *not* by $\neg$;

- turning every rule *Head $\leftarrow$ Body* into a formula by rewriting it as the implication *Body $\rightarrow$ Head*, and

- forming the conjunction of the universal closures of these formulas.

Example 1 *For program Π that contains three rules*

$$p(a)$$
$$q(b)$$
$$r(x) \leftarrow p(x), not\ q(x)$$

the FOL-representation F of Π is

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x)) \qquad (1)$$

and $\text{SM}[F]$ *is*

$$p(a) \wedge q(b) \wedge \forall x((p(x) \wedge \neg q(x)) \rightarrow r(x))$$
$$\wedge\neg\exists uvw(((u, v, w) < (p, q, r)) \wedge u(a) \wedge v(b)$$
$$\wedge\forall x(((u(x) \wedge (\neg v(x) \wedge \neg q(x))) \rightarrow w(x))$$
$$\wedge((p(x) \wedge \neg q(x)) \rightarrow r(x)))),$$

which is equivalent to first-order sentence

$$\forall x(p(x) \leftrightarrow x = a) \wedge \forall x(q(x) \leftrightarrow x = b) \qquad (2)$$
$$\wedge\forall x(r(x) \leftrightarrow (p(x) \wedge \neg q(x)))$$

---

[1] Recall that an *Herbrand interpretation* of a signature σ (containing at least one object constant) is an interpretation of σ such that its universe is the set of all ground terms of σ, and every ground term represents itself. An Herbrand interpretation can be identified with the set of ground atoms to which it assigns the value *true*.

(see (Ferraris et al. *2007), Example 3). The stable models of F are any first-order models of (1) whose signature is $\sigma(F)$. On the other hand F has only one answer set: $\{p(a),\ q(b),\ r(a)\}$.*

We call a formula *negative* if every occurrence of every predicate constant in it belongs to the antecedent of an implication. For instance, any formula of the form $\neg F$ is negative, because this expression is shorthand for $F \rightarrow \bot$.

First-Order Loop Formulas

Review of Loop Formulas from (Chen *et al.* 2006)

We reformulate the definition of a first-order loop formula for a nondisjunctive program from (Chen *et al.* 2006).

Let Π be a nondisjunctive program that has no function constants of positive arity, consisting of a finite number of rules of the form

$$A \leftarrow B, N \tag{3}$$

where A is an atom, and B is a set of atoms, and N is a negative formula.

We will say that Π is in *normal form* if, for all rules (3) in Π, no object constants occur in A. It is clear that every program can be turned into normal form using equality. Let's assume that Π is in normal form.

Let $\sigma(\Pi)$ be the signature consisting of object and predicate constants occurring in Π. Given a finite set Y of non-equality atoms of $\sigma(\Pi)$, we first rename variables in Π so that no variables occur in Y. The *(first-order) external support formula* of Y for Π, denoted by $FES_\Pi(Y)$, is the disjunction of

$$\bigvee_{\theta : A\theta \in Y} \exists \mathbf{z} \left(B\theta \wedge N\theta \wedge \bigwedge_{\substack{p(\mathbf{t}) \in B\theta \\ p(\mathbf{t}') \in Y}} (\mathbf{t} \neq \mathbf{t}') \right) \tag{4}$$

for all rules (3) in Π where θ is a substitution that maps variables in A to terms occurring in Y, and $\mathbf{z}$ is the list of all variables that occur in

$$A\theta \leftarrow B\theta, N\theta$$

but not in Y. [2]

The *(first-order) loop formula* of Y, denoted by $FLF_\Pi(Y)$, is the universal closure of

$$\bigwedge Y \rightarrow FES_\Pi(Y). \tag{5}$$

If Π is a propositional program, for any nonempty finite set Y of propositional atoms, $FLF_\Pi(Y)$ is equivalent to conjunctive loop formulas defined in (Ferraris *et al.* 2006), which we will denote by $LF_\Pi(Y)$.

The definition of a *(first-order) loop* is as follows. We say that $p(\mathbf{t})$ *depends on* $q(\mathbf{t}')$ in Π if Π has a rule (3) such that $p(\mathbf{t})$ is A and $q(\mathbf{t}')$ is in B. The *(first-order) dependency graph* of Π is an infinite directed graph (V, E) such that

- V is a set of non-equality atoms formed from $\sigma(\Pi)$, along with an infinite supply of variables;

- $(p(\mathbf{t})\theta, q(\mathbf{t}')\theta)$ is in E if $p(\mathbf{t})$ depends on $q(\mathbf{t}')$ in Π and θ is a substitution that maps variables in $\mathbf{t}$ and $\mathbf{t}'$ to object constants and variables occurring in V.

A nonempty finite subset L of V is called a *(first-order) loop* of Π if the subgraph of the first-order dependency graph of Π induced by L is strongly connected.

Example 2 *Let Π be the following program:*

$$\begin{aligned}
p(x) &\leftarrow q(x) \\
q(y) &\leftarrow p(y) \\
p(x) &\leftarrow not\ r(x).
\end{aligned} \tag{6}$$

The following sets are first-order loops: $Y_1 = \{p(z)\}$, $Y_2 = \{q(z)\}$, $Y_3 = \{r(z)\}$, $Y_4 = \{p(z), q(z)\}$. Their loop formulas are

$$\begin{aligned}
FLF_\Pi(Y_1) &= \forall z(p(z) \rightarrow (q(z) \vee \neg r(z))); \\
FLF_\Pi(Y_2) &= \forall z(q(z) \rightarrow p(z)); \\
FLF_\Pi(Y_3) &= \forall z(r(z) \rightarrow \bot); \\
FLF_\Pi(Y_4) &= \forall z(p(z) \wedge q(z) \rightarrow \\
&\quad (q(z) \wedge z \neq z) \vee (p(z) \wedge z \neq z) \vee \neg r(z)).
\end{aligned}$$

Example 3 *Let Π be the one-rule program*

$$p(x) \leftarrow p(y). \tag{7}$$

Its first-order loops are $Y_k = \{p(x_1), \ldots, p(x_k)\}$ where $k > 0$. Formula $FLF_\Pi(Y_k)$ is

$$\begin{aligned}
\forall x_1 &\ldots x_k(p(x_1) \wedge \ldots \wedge p(x_k) \\
&\rightarrow \exists y(p(y) \wedge (y \neq x_1) \wedge \ldots \wedge (y \neq x_k))).
\end{aligned} \tag{8}$$

Definition 1 (Grounding a program) *For any nondisjunctive program Π we denote by $Ground(\Pi)$ the ground instance of Π, that is the program obtained from Π by replacing every occurrence of object variables with every object constant occurring in Π, and then replacing equality $a = b$ with $\top$ or $\bot$ depending on whether a is the same symbol as b.*

Given a program Π, let $(\sigma(\Pi))^g$ be a propositional signature consisting of all the ground atoms of $\sigma(\Pi)$. An Herbrand model of $\sigma(\Pi)$ can be identified with a corresponding propositional model of $(\sigma(\Pi))^g$.

The following is a reformulation of Theorem 1 from (Chen *et al.* 2006).

Proposition 1 *Let Π be a nondisjunctive program in normal form, and let I be an Herbrand model of Π whose signature is $\sigma(\Pi)$. The following conditions are equivalent to each other:*

(a) I is an answer set of Π;

(b) I is an Herbrand model of $\{FLF_\Pi(Y) : Y \text{ is a nonempty finite set of atoms of } \sigma(\Pi)\}$;

(c) I is an Herbrand model of $\{FLF_\Pi(Y) : Y \text{ is a first-order loop of } \Pi\}$;

(d) I is a (propositional) model of $\{LF_{Ground(\Pi)}(Y) : Y \text{ is a nonempty (finite) set of ground atoms of } (\sigma(\Pi))^g\}$;

(e) I is a (propositional) model of $\{LF_{Ground(\Pi)}(Y) : Y \text{ is a loop of } Ground(\Pi)\} \cup \{\neg p : p \text{ is an atom in } (\sigma(\Pi))^g \text{ not occurring in } Ground(\Pi)\}$.

The sets of first-order loop formulas considered in conditions (b), (c) above have obvious redundancy. For instance, the loop formula of $\{p(x)\}$ is equivalent to the loop formula of $\{p(y)\}$; the loop formula of $\{p(x), p(y)\}$ entails the loop formula of $\{p(z)\}$. Following (Chen *et al.* 2006), given two sets of atoms Y_1, Y_2 not containing equality, we say that Y_1 *subsumes* Y_2 if there is a substitution θ that maps variables in Y_1 to terms so that $Y_1\theta = Y_2$. We say that Y_1 and Y_2 are *equivalent* if they subsume each other.

Proposition 2 *(Chen* et al. *2006, Proposition 7) Given two loops Y_1 and Y_2, if Y_1 subsumes Y_2, then $FLF_\Pi(Y_1)$ entails $FLF_\Pi(Y_2)$.*

Therefore in condition (c) from Proposition 1, it is sufficient to consider a set Γ of loops such that for every loop L of Π, there is a loop L' in Γ that subsumes L. Chen *et al.* [2006] called this set of loops *complete*. In Example 2, set $\{Y_1, Y_2, Y_3, Y_4\}$ is a finite complete set of loops of program (6). Program (7) in Example 3 has no finite complete set of loops.

In condition (c) of Proposition 1, instead of the first-order loops of the given program, one may consider the first-order loops of any strongly equivalent program, including a program that is not in normal form. This sometimes yields a smaller number of loop formulas to consider. For example, the ground loops of program

$$p(a) \leftarrow p(b)$$
$$p(b) \leftarrow p(c) \tag{9}$$

are $\{p(a)\}$, $\{p(b)\}$, $\{p(c)\}$, all of which are subsumed by $\{p(x)\}$. Thus it is sufficient to consider the loop formula of $\{p(x)\}$:

$$\forall x(p(x) \rightarrow ((x = a) \wedge p(b) \wedge (x \neq b))$$
$$\vee((x = b) \wedge p(c) \wedge (x \neq c))). \tag{10}$$

On the other hand, the ground loops of its normal form

$$p(x) \leftarrow x{=}a,\ p(b)$$
$$p(x) \leftarrow x{=}b,\ p(c)$$

contain $\{p(b), p(c)\}$ in addition to the singleton ground loops.

Extension to Disjunctive Programs

A disjunctive program consists of a finite number of rules of the form

$$A \leftarrow B, N \tag{11}$$

where A, B are sets of atoms, and N is a negative formula. As in the nondisjunctive case we assume that there are no function constants of positive arity. Similar to above, a program is in *normal form* if, for all rules (11) in Π, no object constants occur in A. We assume that Π is in normal form.

Given a finite set Y of non-equality atoms of $\sigma(\Pi)$, we first rename variables in Π so that no variables occur in Y. The *(first-order) external support formula* of Y for Π, denoted by $FES_\Pi(Y)$, is the disjunction of

$$\bigvee_{\theta: A\theta \cap Y \neq \emptyset} \exists \mathbf{z} \left(B\theta \wedge N\theta \wedge \bigwedge_{\substack{p(\mathbf{t}) \in B\theta \\ p(\mathbf{t}') \in Y}} (\mathbf{t} \neq \mathbf{t}') \right.$$
$$\left. \wedge \neg \left(\bigvee_{p(\mathbf{t}) \in A\theta} \left(p(\mathbf{t}) \wedge \bigwedge_{p(\mathbf{t}') \in Y} \mathbf{t} \neq \mathbf{t}' \right) \right) \right) \tag{12}$$

for all rules (11) in Π where θ is a substitution that maps variables in A to terms occurring in Y or to themselves, and $\mathbf{z}$ is the list of all variables that occur in

$$A\theta \leftarrow B\theta, N\theta$$

but not in Y. The *(first-order) loop formula* of Y for Π, denoted by $FLF_\Pi(Y)$, is the universal closure of (5). Clearly, (12) is equivalent to (4) when Π is nondisjunctive.

Similar to the nondisjunctive case, we say that $p(\mathbf{t})$ *depends on* $q(\mathbf{t}')$ in Π if there is a rule (11) in Π such that $p(\mathbf{t})$ is in A and $q(\mathbf{t}')$ is in B. The notions of grounding, a dependency graph and a first-order loop are extended to disjunctive programs in a straightforward way. Propositions 1 and 2 can be extended to disjunctive programs with these extended notions.

Example 4 *Let Π be the following program $p(x, y) \vee p(y, z) \leftarrow q(x)$ and let $Y = \{p(u, v)\}$. Formula $FLF_\Pi(Y)$ is the universal closure of*

$$p(u, v) \rightarrow \quad \exists z(q(u) \wedge \neg(p(v, z) \wedge ((v, z) \neq (u, v))))$$
$$\vee \exists x(q(x) \wedge \neg(p(x, u) \wedge ((x, u) \neq (u, v)))).$$

Extension to Arbitrary Sentences

First-order loop formulas can even be extended to arbitrary sentences under the stable model semantics (Ferraris *et al.* 2007).

As in (Ferraris *et al.* 2006), it will be easier to discuss the result with a formula whose *negation* is similar to *FES*. We define formula $NFES_F(Y)$ (*"Negation" of FES*) as follows, where F is a first-order formula and Y is a finite set of atoms not containing equality. The reader familiar with (Ferraris *et al.* 2006) will notice that this is a generalization of the notion *NES* from that paper to first-order formulas.

We assume that no variables in Y occur in F by renaming bound variables in F.

- $NFES_{p_i(\mathbf{t})}(Y) = p_i(\mathbf{t}) \wedge \bigwedge_{p_i(\mathbf{t}') \in Y} \mathbf{t} \neq \mathbf{t}'$;
- $NFES_{t_1 = t_2}(Y) = (t_1 = t_2)$;
- $NFES_\perp(Y) = \perp$;
- $NFES_{F \wedge G}(Y) = NFES_F(Y) \wedge NFES_G(Y)$;
- $NFES_{F \vee G}(Y) = NFES_F(Y) \vee NFES_G(Y)$;
- $NFES_{F \rightarrow G}(Y) = (NFES_F(Y) \rightarrow NFES_G(Y)) \wedge (F \rightarrow G)$;
- $NFES_{\forall x G}(Y) = \forall x NFES_G(Y)$;
- $NFES_{\exists x G}(Y) = \exists x NFES_G(Y)$.

The (first-order) loop formula of Y for sentence F, denoted by $FLF_F(Y)$, is the universal closure of

$$\bigwedge Y \rightarrow \neg NFES_F(Y). \tag{13}$$

It is not difficult to check that for any propositional formula F and any nonempty finite set Y of propositional atoms, $FLF_F(Y)$ is equivalent to $LF_F(Y)$, where LF denotes loop formula for a propositional formula as defined in (Ferraris *et al.* 2006).

This notion of a loop formula is a generalization of a loop formula for a disjunctive program in view of the following lemma.

Lemma 1 *Let Π be a disjunctive program in normal form, F the FOL-representation of Π, and Y a finite set of atoms not containing equality. Formula $NFES_F(Y)$ is equivalent to $\neg FES_\Pi(Y)$ under the assumption Π.*

To define a first-order dependency graph of F, we need a few notions. Recall that an occurrence of a formula G in a formula F is *positive* if the number of implications in F containing that occurrence in the antecedent is even; it is *strictly positive* if that number is 0. We will call a formula in *rectified form* if it has no variables that are both bound and free, and the quantifiers are followed by pairwise distinct variables. Any formula can be turned into rectified form by renaming bound variables.

Let F be a formula in rectified form. We say that an atom $p(\mathbf{t})$ *weakly depends on* an atom $q(\mathbf{t}')$ in an implication $G \to H$ if

- $p(\mathbf{t})$ has a strictly positive occurrence in H, and

- $q(\mathbf{t}')$ has a positive occurrence in G that does not belong to any occurrence of a negative formula in G.

We say that $p(\mathbf{t})$ *depends on* $q(\mathbf{t}')$ in F if $p(\mathbf{t})$ weakly depends on $q(\mathbf{t}')$ in an implication that has a strictly positive occurrence in F.

The definition of a first-order dependency graph for a nondisjunctive program is extended to F in a straightforward way using this extended notion of dependency between two atoms. A loop is also defined similarly.

Definition 2 (Grounding a sentence) *For any sentence F that has no function constants of positive arity, $Ground(F)$ is defined recursively. If F is an atom $p(\mathbf{t})$ then $Ground(F)$ is F. If F is an equality $a = b$ then $Ground(F)$ is $\top$ or $\bot$ depending on whether a is the same symbol as b. The function $Ground$ commutes with all propositional connectives; quantifiers turn into finite conjunctions and disjunctions over all object constants occurring in F.*

Proposition 1 remains correct even after replacing "a nondisjunctive program in normal form" in the statement with "a sentence in rectified form that contains no function constants of positive arity," and using the extended notions accordingly. Proposition 2 can be extended to arbitrary sentences as well.

Loop Formulas in Second-Order Logic

SM and Loop Formulas

Let F be a first-order formula, let $p_1, \ldots, p_n$ be the list of all predicate constants occurring in F, and let $\mathbf{u}$ and $\mathbf{v}$ be lists of predicate variables corresponding to $p_1, \ldots, p_n$.

We define $NES_F(\mathbf{u})$ recursively as follows, which is similar to $NFES$ above but contains second-order variables as its argument.

- $NES_{p_i(\mathbf{t})}(\mathbf{u}) = p_i(\mathbf{t}) \wedge \neg u_i(\mathbf{t})$;

- $NES_{t_1=t_2}(\mathbf{u}) = (t_1=t_2)$;

- $NES_\bot(\mathbf{u}) = \bot$;

- $NES_{F \wedge G}(\mathbf{u}) = NES_F(\mathbf{u}) \wedge NES_G(\mathbf{u})$;

- $NES_{F \vee G}(\mathbf{u}) = NES_F(\mathbf{u}) \vee NES_G(\mathbf{u})$;

- $NES_{F \to G}(\mathbf{u}) = (NES_F(\mathbf{u}) \to NES_G(\mathbf{u})) \wedge (F \to G)$;

- $NES_{\forall x F}(\mathbf{u}) = \forall x NES_F(\mathbf{u})$;

- $NES_{\exists x F}(\mathbf{u}) = \exists x NES_F(\mathbf{u})$.

By $Nonempty(\mathbf{u})$ we denote the formula

$$\exists \mathbf{x}^1 u_1(\mathbf{x}^1) \vee \cdots \vee \exists \mathbf{x}^n u_n(\mathbf{x}^n).$$

$SM[F]$ can be written in the style of "loop formulas" in the following way.

Proposition 3 *For any sentence F, $SM[F]$ is equivalent to*

$$F \wedge \forall \mathbf{u}((\mathbf{u} \leq \mathbf{p}) \wedge Nonempty(\mathbf{u}) \to \neg NES_F(\mathbf{u})). \quad (14)$$

Second-Order Characterization of Loops

The notion of a loop can be incorporated into the second-order definition of stable models as follows.

Given a sentence F in rectified form, by $E_F(\mathbf{v}, \mathbf{u})$ we denote

$$\bigvee_{\substack{(p_i(\mathbf{t}), p_j(\mathbf{t}')) \ : \\ p_i(\mathbf{t}) \text{ depends on } p_j(\mathbf{t}') \text{ in } F}} \exists \mathbf{z}(v_i(\mathbf{t}) \wedge u_j(\mathbf{t}') \wedge \neg v_j(\mathbf{t}'))$$

where $\mathbf{z}$ is the list of all object variables in $\mathbf{t}$ and $\mathbf{t}'$. By $SC_F(\mathbf{u})$ we denote the second-order sentence

$$Nonempty(\mathbf{u}) \wedge \forall \mathbf{v}((\mathbf{v} < \mathbf{u}) \wedge Nonempty(\mathbf{v}) \to E_F(\mathbf{v}, \mathbf{u})). \quad (15)$$

Formula (15) represents the concept of a loop without referring to the notion of a dependency graph explicitly, based on the following observation. Consider a finite propositional program Π. A set U of atoms is a loop of Π iff for every nonempty proper subset V of U, there is an edge from an atom in V to an atom in $U \setminus V$ in the dependency graph of Π (Gebser *et al.* 2006). To see the relation in the first-order case, we first define a dependency graph and a loop that are relative to a given interpretation. Let F be a sentence in rectified form and let I be an interpretation of F. The *dependency graph of F w.r.t. I* is an infinite directed graph (V, E) where

- V is the set of all atoms of the form $p_i(\vec{\xi^*})$ where $\vec{\xi^*}$ is a list of object names, [3] and

- $(p_i(\vec{\xi^*}), p_j(\vec{\eta^*}))$ is in E if there are $p_i(\mathbf{t})$, $p_j(\mathbf{t}')$ such that $p_i(\mathbf{t})$ depends on $p_j(\mathbf{t}')$ in F and there is a mapping θ from variables in $\mathbf{t}$ and $\mathbf{t}'$ to object names such that $(\mathbf{t}\theta)^I = \vec{\xi}$, and $(\mathbf{t}'\theta)^I = \vec{\eta}$.

We call a nonempty subset L of V a *loop* of F w.r.t. I if the subgraph of the dependency graph of F w.r.t. I that is induced by L is strongly connected. [4] The following lemma describes the relation between formula (15) and a loop w.r.t. I.

---

[3] Each element ξ of the universe $|I|$ has a corresponding *object name*, which is an object constant not from the given signature See (Lifschitz *et al.* 2008) for details.

[4] Note that unlike first-order loops defined earlier we don't restrict L to be finite. There the assumption was required to be able to write a loop formula.

Lemma 2 *Let F be a first-order sentence in rectified form, I an interpretation of F and $\mathbf{q}$ a list of predicate names* [5] *corresponding to $\mathbf{p}$. $I \models SC_F(\mathbf{q})$ iff*

$$Y = \{p_i(\vec{\xi^*}) : q_i^I(\vec{\xi})=\text{TRUE where } \vec{\xi} \text{ is a list of object names}\}$$

is a loop of F w.r.t. I.

One may expect that, similar to the equivalence between conditions (b) and (c) from Proposition 1, formula (14) is equivalent to the following formula:

$$F \wedge \forall \mathbf{u}((\mathbf{u} \leq \mathbf{p}) \wedge SC_F(\mathbf{u}) \to \neg NES_F(\mathbf{u})). \qquad (16)$$

However, this is not the case as shown in the following example.

Example 5 *Let F be the FOL-representation of program Π :*

$$p(x,y) \leftarrow q(x,z)$$
$$q(x,z) \leftarrow p(y,z).$$

Consider interpretation I whose universe is the set of all nonnegative integers such that

$$p^I(m,n) = \begin{cases} \text{TRUE} & \text{if } m = n, \\ \text{FALSE} & \text{otherwise;} \end{cases}$$

$$q^I(m,n) = \begin{cases} \text{TRUE} & \text{if } n = m+1, \\ \text{FALSE} & \text{otherwise;} \end{cases}$$

One can check that I is not a stable model of F, but satisfies (16).

This mismatch is similar to the observation from (Lee 2005) that the external support of all loops does not ensure the stability of the model if the program is allowed to be infinite. Consider the following infinite program:

$$p_i \leftarrow p_{i+1} \qquad (i > 0). \qquad (17)$$

The only loops are singletons, and their loop formulas are satisfied by the model $\{p_1, p_2, \ldots\}$ of (17), which is not stable. To check the stability, not only we need to check every loop is externally supported, but also need to check that $\{p_1, p_2, \ldots\}$ is "externally supported." Example 5 shows that the mismatch can occur even if the program is finite once it is allowed to contain variables. What distinguishes $\{p_1, p_2, \ldots\}$ from loops is that, for every loop in $\{p_1, p_2, \ldots\}$, there is an outgoing edge in the dependency graph. Taking this into account, we define $Loop_F(\mathbf{u})$ as

$$SC_F(\mathbf{u}) \vee (Nonempty(\mathbf{u})$$
$$\wedge \forall \mathbf{v}((\mathbf{v} \leq \mathbf{u}) \wedge SC_F(\mathbf{v}) \to E_F(\mathbf{v}, \mathbf{u}))). \qquad (18)$$

Given a dependency graph of F w.r.t. I, we say that a nonempty set Y of vertices is *unbounded* w.r.t. I if, for every subset Z of Y that induces a strongly connected subgraph, there is an edge from a vertex in Z to a vertex in $Y \setminus Z$. For instance, for the interpretation I in Example 5,

$$\{p(0^*, 0^*), q(0^*, 1^*), p(1^*, 1^*), q(1^*, 2^*), \ldots, \}$$

is an unbounded set w.r.t. I.

The following lemma describes the relation between the second disjunctive term of (18) with unbounded sets.

---

[5]Like object names, for every $n > 0$, each subset of $|I|^n$ has a name, which is an n-ary predicate constant not from the given signature.

Lemma 3 *Let F be a first-order sentence in rectified form, I an interpretation, and $\mathbf{q}$ a list of predicate names corresponding to $\mathbf{p}$.*

$$I \models Nonempty(\mathbf{q}) \wedge \forall \mathbf{v}((\mathbf{v} \leq \mathbf{q}) \wedge SC_F(\mathbf{v}) \to E_F(\mathbf{v}, \mathbf{q}))$$

iff

$$Y = \{p_i(\vec{\xi^*}) : q_i^I(\vec{\xi})=\text{TRUE where } \vec{\xi} \text{ is a list of object names}\}$$

is an unbounded set of F w.r.t. I.

An *extended loop* of F w.r.t. I is a loop or an unbounded set of F w.r.t. I. Clearly $I \models (18)$ iff

$$Y = \{p_i(\vec{\xi^*}) : q_i^I(\vec{\xi})=\text{TRUE where } \vec{\xi} \text{ is a list of object names}\}$$

is an extended loop of F w.r.t. I.

The following proposition shows that the formula obtained from (16) by replacing $SC_F(\mathbf{u})$ with $Loop_F(\mathbf{u})$ is equivalent to SM$[F]$.

Proposition 3′ *For any sentence F in rectified form, the following second-order sentences are equivalent to each other:*

(a) SM$[F]$;

(b) $F \wedge \forall \mathbf{u}((\mathbf{u} \leq \mathbf{p}) \wedge Nonempty(\mathbf{u}) \to \neg NES_F(\mathbf{u}))$;

(c) $F \wedge \forall \mathbf{u}((\mathbf{u} \leq \mathbf{p}) \wedge Loop_F(\mathbf{u}) \to \neg NES_F(\mathbf{u}))$.

(See appendix A for an example.)

Proposition 3′ is essentially a generalization of the main theorem from (Ferraris *et al.* 2006) to first-order sentences. If F is a propositional formula, then for any subset Y of $\mathbf{p}$, by $\overrightarrow{Y}$ we denote the tuple $(Y_1, \ldots, Y_n)$, where

$$Y_i = \begin{cases} \top, & \text{if } p_i \in Y; \\ \bot, & \text{otherwise.} \end{cases}$$

Corollary 1 *(Ferraris* et al. *2006, Theorem 2) For any propositional formula F, the following conditions are equivalent to each other under the assumption F.*

(a) SM$[F]$;

(b) The conjunction of $\bigwedge Y \to \neg NES_F(\overrightarrow{Y})$ for all nonempty sets Y of atoms occurring in F;

(c) The conjunction of $\bigwedge Y \to \neg NES_F(\overrightarrow{Y})$ for all loops Y of F.

Several other propositions in this paper are derived from Proposition 3′.

Between SM and First-Order Loop Formulas

In general, SM$[F]$ is not reducible to any first-order sentence, even in the absence of function constants of positive arity. As in circumscription, transitive closure can be represented using SM, while it cannot be done by any set of first-order formulas, even if that set is allowed to be infinite.[6] However, if the universe consists of finite elements, then the following holds. We will say that F is in *normal form* if no object constants occur in a strictly positive occurrence of atoms in F.

---

[6]Vladimir Lifschitz, personal communication.

Proposition 4 *For any sentence F and any model I of F whose universe is finite, the following conditions are equivalent:*

(a) I satisfies $\mathrm{SM}[F]$;

(b) for every nonempty finite set Y of atoms formed from predicate constants in $\sigma(F)$ and an infinite supply of variables, I satisfies $FLF_F(Y)$.

If F is in rectified and normal form that has no function constants of positive arity, the following condition is also equivalent to each of (a) and (b):

(c) for every first-order loop Y of F, I satisfies $FLF_F(Y)$.

Unlike Proposition 1 in which loops can be found from any strongly equivalent program, condition (c) requires that loops be found from a normal form. This is related to the fact that Proposition 4 considers non-Herbrand stable models as well, which may not satisfy the unique name assumption. For instance, recall that program (9) has singleton loops only, which are subsumed by $\{p(x)\}$. Consider an interpretation I such that $|I| = \{1,2\}$ and $a^I = c^I = 1$, $b^I = 2$, $p^I(m) = \mathrm{TRUE}$ for $m = 1,2$. I is a non-Herbrand model which is not stable, but it satisfies (10), the loop formula of $\{p(x)\}$.

The proof of the equivalence between (a) and (c) uses the following lemma.

Lemma 4 *Let F be a sentence in rectified and normal form that contains no function constants of positive arity, and let I be an interpretation. If there is no infinite extended loop of F w.r.t. I, then $I \models \mathrm{SM}[F]$ iff, for every first-order loop Y of F, $I \models FLF_F(Y)$.*

Without the finite universe assumption, Proposition 4 would be incorrect, as shown in Example 5. For another example, consider program (7) with an interpretation I with an infinite universe such that p is identically true. I does not satisfy $\mathrm{SM}[F]$, but satisfies F and $FLF_F(Y)$ for any finite set Y of atoms.

In view of Proposition 2, if the size of the universe $|I|$ is known, as with the answer sets (whose universe is the Herbrand universe of $\sigma(F)$), it is sufficient to consider at most $2^{|\mathbf{p}|} - 1$ loop formulas where $\mathbf{p}$ is set of all predicate constants occurring in the sentence. Each loop formula is for set $Y_{\mathbf{q}}$ corresponding to a nonempty subset $\mathbf{q}$ of $\mathbf{p}$, defined as $Y_{\mathbf{q}} = \{p(\mathbf{x_1}), \ldots, p(\mathbf{x_{|I|^n}}) : p \in \mathbf{q}\}$ where n is the arity of p. For instance, for program (7), if the size of the universe is known to be 3, it is sufficient to consider only one loop formula (8) where $k = 3$.

In the next section we consider certain classes of sentences for which $\mathrm{SM}[F]$ is equivalent to a first-order sentence without the finite universe assumption.

Reducibility to first-order formulas

Finite complete set of first-order loops

Proposition 8 from (Ferraris *et al.* 2007) shows that $\mathrm{SM}[F]$ can be reduced to a first-order sentence if F is "tight", i.e., F has no "nontrivial" predicate loops. (Predicate loops are defined similar to first-order loops, but from a "predicate dependency graph" (Ferraris *et al.* 2007), which does not take

into account "pointwise dependency.") We further generalize this result using the notion of finite complete set of loops.

Let F be a sentence in rectified form that contains no function constants of positive arity. Theorem 2 from (Chen *et al.* 2006) provides a syntactic condition under which a nondisjunctive program has a finite complete set of loops, which can be extended to disjunctive programs and arbitrary sentences in a straightforward way.

The following proposition tells that if F has a finite complete set of loops, then $\mathrm{SM}[F]$ can be equivalently rewritten as a first-order sentence.

Proposition 5 *Let F be a sentence in rectified and normal form that contains no function constants of positive arity. If F has a finite complete set Γ of first-order loops, then $\mathrm{SM}[F]$ is equivalent to the conjunction of F with the set of loop formulas for all loops in Γ.*

This proposition generalizes Proposition 8 from (Ferraris *et al.* 2007). Clearly, every tight sentence has a finite complete set of first-order loops.

The proof of Proposition 5 follows from Lemma 4 and the following lemma.

Lemma 5 *Let F be a sentence in rectified and normal form that contains no function constants of positive arity. If F has a finite complete set of loops, then there is no infinite extended loop of F w.r.t any interpretation.*

Proposition 5 would go wrong if we replace "a finite complete set of loops" in the statement with "a finite number of predicate loops." Obviously any sentence F contains a finite number of predicate constants, so that this condition is trivial. In view of intranslatability of SM to first-order sentences, this fact tells that the more refined notion of first-order loops is essential for this proposition to hold.

For nondisjunctive program Π, Proposition 9 from (Chen *et al.* 2006) shows that if every variable in the head occurs in the body, then Π has a finite complete set of loops. However, this does not hold once Π is allowed to be disjunctive. For instance,

$$p(x,y) \leftarrow q(x), r(y)$$
$$q(x) \vee r(y) \leftarrow p(x,y)$$

has no finite complete set of loops.

Safe formulas

A disjunctive program Π is called *safe* if, for each rule (11) of Π, every variable occurring in the rule occurs in B as well. (Lee *et al.* 2008) generalized this notion to sentences, showing that for any safe sentence, its Herbrand stable models are not affected by "irrelevant" object constants that do not occur in the program. We will show that this notion is also related to reducing $\mathrm{SM}[F]$ to a first-order sentence.

We review the notion of safety from (Lee *et al.* 2008).[7] We assume that there are no function constants of positive arity. As a preliminary step, we assign to every formula F in rectified form a set $RV(F)$ of its *restricted variables*, as follows:

---

[7]The definition here is slightly weaker and applies to arbitrary sentences, unlike the one in (Lee *et al.* 2008) that refers to prenex form.

- For an atom F,
  - if F is an equality between two variables then $RV(F) = \emptyset$;
  - otherwise, $RV(F)$ is the set of all variables occurring in F;
- $RV(\bot) = \emptyset$;
- $RV(F \wedge G) = RV(F) \cup RV(G)$;
- $RV(F \vee G) = RV(F) \cap RV(G)$;
- $RV(F \rightarrow G) = \emptyset$;
- $RV(QvF) = RV(F) \setminus \{v\}$ where $Q \in \{\forall, \exists\}$.

We say that a variable x is *unsafe* in F if there is an occurrence of x in F that is not in any of

- $\forall x, \exists x$, and

- any subformula $G \rightarrow H$ of F such that $x \in RV(G)$.

By U_F we denote the formula

$$\bigwedge_{p \in \mathbf{p}} \forall \mathbf{x} \Big(p(\mathbf{x}) \rightarrow \bigwedge_{x \in \mathbf{x}} \bigvee_{c \in C} x = c \Big)$$

where C is the set of all object constants occurring in F, and $\mathbf{x}$ is a list of distinct object variables whose length is the same as the arity of p.

The following proposition tells that for a safe sentence F, formula $\mathrm{SM}[F]$ can be equivalently rewritten as a first-order sentence.

Proposition 6 *Let F be a sentence in rectified form that has no function constants of positive arity. If F has no unsafe variables, then $\mathrm{SM}[F]$ is equivalent to the conjunction of F, U_F and a finite number of loop formulas.*

We note that the syntactic conditions in Propositions 5 and 6 do not entail each other. For instance, $\forall x \, (q(x) \wedge p(y) \rightarrow p(x))$ has no unsafe variables, but has no finite complete set of first-order loops, while $\forall x \, p(x)$ has a finite complete set of loops $\{\{p(x)\}\}$, but has an unsafe variable x.

Safety is usually imposed on input programs for answer set solvers, but it could be somewhat restricted in first-order reasoning which is not confined to generating Herbrand stable models. For instance, the example program in the introduction (identified as a sentence) has an unsafe variable w (but has a finite complete set of loops).

Programs with Explicit Quantifiers

In the following we extend the syntax of logic programs by allowing explicit quantifiers. As in answer set programs, the syntax uses the intuitive if-then form, but allows explicit quantifiers. An *extended rule* is of the form

$$H \leftarrow G \qquad (19)$$

where G and H are formulas with no function constants of positive arity such that every occurrence of an implication in G and H is in a negative formula. An extended program is a finite set of extended rules. The semantics of an extended program is defined by identifying the program with $\mathrm{SM}[F]$

where F is a conjunction of the universal closure of implications that correspond to the rules (FOL-representation). An example of an extended program is given in the introduction.

Let Π be an extended program. Given a nonempty finite set Y of non-equality atoms of $\sigma(\Pi)$, we first rename variables in Π so that no variables occur in Y. Formula $EFES_\Pi(Y)$ ("*Extended FES*") is defined as the disjunction of

$$\exists \mathbf{z}(NFES_G(Y) \wedge \neg NFES_H(Y)) \qquad (20)$$

for all rules (19) where H contains a strictly positive occurrence of a predicate constant that belongs to Y, and $\mathbf{z}$ is the list of all free variables in the rule that do not occur in Y.

The loop formula of Y for Π is the universal closure of

$$\bigwedge Y \rightarrow EFES_\Pi(Y). \qquad (21)$$

The following proposition tells that (21) is a generalization of the definition of a loop formula for a disjunctive program and is equivalent to the definition of a loop formula (13) for an arbitrary sentence.

Proposition 7 *Let Π be an extended program, F the FOL-representation of Π, and Y a finite set of atoms not containing equality. Under the assumption Π, formula $EFES_\Pi(Y)$ is equivalent to $\neg NFES_F(Y)$. If Π is a disjunctive program, then $EFES_\Pi(Y)$ is also equivalent to $FES_\Pi(Y)$ under the assumption Π.*

While the size of (13) is exponential to the size of F in the worst case, (21) can be equivalently written in a linear size due to the following lemma.

Lemma 6 *For any negative formula F and any finite set Y of non-equality atoms, $NFES_F(Y)$ is equivalent to F.*

For instance, for $F = (p(x) \rightarrow \bot) \rightarrow \bot$ and $Y = \{p(a)\}$, formula $NFES_F(Y)$ is

$$[(((p(x) \wedge x \neq a) \rightarrow \bot) \wedge (p(x) \rightarrow \bot)) \rightarrow \bot] \wedge [(p(x) \rightarrow \bot) \rightarrow \bot],$$

which is equivalent to F.

A finite set Γ of sentences *entails* a sentence F under the stable model semantics (symbolically, $\Gamma \models_{\mathrm{SM}} F$), if every stable model of Γ satisfies F.

If $\mathrm{SM}[F]$ can be reduced to a first-order sentence, as allowed in Propositions 5 and 6, clearly, the following holds.

$$\Gamma \models_{\mathrm{SM}} F \text{ iff } \Gamma \cup \Delta \models F$$

where Δ is the set of first-order loop formulas required. This fact allows us to use first-order theorem provers to reason about query entailment under the stable model semantics.

Example 6 *Consider the insurance policy example in the introduction, which has the following finite complete set of loops: $\{Divorced(u, v)\}$, $\{Accident(u, v)\}$, $\{Discount(u, v)\}$, $\{GotMarried(u, v)\}$, $\{Spouse(u, v)\}$ and $\{GotMarried(u, v), Spouse(u, v)\}$. Their loop formulas for*

$\Pi_1 \cup \Pi_2$ *are equivalent to the universal closure of*

$$Div(u,v) \rightarrow \bot$$

$$Acc(u,v) \rightarrow \bot$$

$$Dis(u,v) \rightarrow \exists xy[Spo(x,y) \wedge \neg\exists z Acc(x,z) \\ \wedge \neg(\exists w(Dis(x,w) \wedge (x,w) \neq (u,v))))]$$

$$Mar(u,v) \rightarrow \\ \exists xy[Spo(x,y) \wedge \neg(Mar(x,y) \wedge (x,y) \neq (u,v))] \\ \vee \neg\exists y[Mar(marge,y) \wedge (marge,y) \neq (u,v)]$$

$$Spo(u,v) \rightarrow \\ \exists xy[Mar(x,y) \wedge \neg Div(x,y) \\ \wedge\neg(Spo(x,y) \wedge (x,y) \neq (u,v))]$$

$$Mar(u,v) \wedge Spo(u,v) \rightarrow \\ \exists xy[(Spo(x,y) \wedge (x,y) \neq (u,v)) \\ \wedge\neg(Mar(x,y) \wedge (x,y) \neq (u,v))] \\ \vee \neg\exists y[Mar(marge,y) \wedge (marge,y) \neq (u,v)] \\ \vee \exists xy[(Mar(x,y) \wedge (x,y) \neq (u,v)) \wedge \neg Div(x,y) \\ \wedge\neg(Spo(x,y) \wedge (x,y) \neq (u,v))].$$

These loop formulas, conjoined with the FOL-representation of $\Pi_1 \cup \Pi_2$, entail under first-order logic each of $\exists xw\, Dis(x,w)$ and $\forall x(Dis(x,plan1) \rightarrow x = marge)$. We verified the answers using a first-order theorem prover Vampire [8].

Conclusion

Our main contributions are as follows.

- We extended loop formulas with variables from (Chen *et al.* 2006) to disjunctive programs and to arbitrary first-order sentences and showed their relations to the new language of stable models from (Ferraris *et al.* 2007).

- We presented certain syntactic conditions under which the language of stable models from (Ferraris *et al.* 2007) can be reduced to first-order logic, which allows to use first-order theorem provers to reason about stable models.

- We defined the notion of an extended program which allows closed-world reasoning under the stable model semantics even in the absence of the unique name and the domain closure assumptions. We provided a computational method for extended programs by means of loop formulas.

The use of first-order theorem provers for the stable model semantics was already investigated in (Sabuncu and Alpaslan 2007), but their results are limited in several ways. They considered nondisjunctive logic programs with "trivial" loops only, in which case the stable model semantics is equivalent to the completion semantics (Clark 1978); their notion of models were limited to Herbrand models.

SAT-based answer set solvers may also benefit from loop formulas with variables. Instead of finding propositional loop formulas one by one from the ground program, one

may consider a set of formulas in a batch which are obtained from grounding first-order loop formulas. Whether it leads to computational efficiency needs empirical evaluation.

References

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

Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 47:79–101, 2006.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 372–379, 2007.

Martin Gebser, Joohyung Lee, and Yuliya Lierler. Elementary sets for logic programs. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 244–249, 2006.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *Proceedings of International Conference on Logic Programming (ICLP)*, pages 451–465, 2003.

Joohyung Lee and Fangzhen Lin. Loop formulas for circumscription. *Artificial Intelligence*, 170(2):160–185, 2006.

Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. A reductive semantics for counting and choice in answer set programming. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2008. To appear.

Joohyung Lee. Nondefinite vs. definite causal theories. In *Proc. 7th Int'l Conference on Logic Programming and Nonmonotonic Reasoning*, pages 141–153, 2004.

Joohyung Lee. A model-theoretic counterpart of loop formulas. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 503–508, 2005.

Vladimir Lifschitz, Leora Morgenstern, and David Plaisted. Knowledge representation and classical logic. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2008.

---

[8]`http://www.vampire.fm` .

Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157:115–137, 2004.

Lengning Liu and Mirosław Truszczyński. Properties of programs with monotone and convex constraints. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 701–706, 2005.

John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39,171–172, 1980.

Orkunt Sabuncu and Ferda N. Alpaslan. Computing answer sets using model generation theorem provers. In Working Notes of Answer Set Programming (ASP) 2007, 2007.

Appendix. Additional Examples

Consider program (6) from Example 2:

$$p(x) \leftarrow q(x)$$
$$q(y) \leftarrow p(y)$$
$$p(x) \leftarrow not\ r(x).$$

Let F be the FOL-representation of Π:

$$\forall xy((q(x) \rightarrow p(x)) \wedge (p(y) \rightarrow q(y)) \wedge (\neg r(x) \rightarrow p(x))).$$

Below we use the following fact to simplify the formulas.

Lemma 7 *For any negative formula F, the formula*

$$NES_F(\mathbf{u}) \leftrightarrow F$$

is logically valid.

1. SM$[F]$ is equivalent to

$$F \wedge \neg \exists u_1 u_2 u_3((u_1, u_2, u_3) < (p, q, r))$$
$$\wedge \forall xy((u_2(x) \rightarrow u_1(x))$$
$$\wedge (u_1(y) \rightarrow u_2(y))$$
$$\wedge (\neg r(x) \rightarrow u_1(x))).$$

2. Formula in Proposition 3′ (b):

$$F \wedge \forall \mathbf{u}(\mathbf{u} \le \mathbf{p} \wedge Nonempty(\mathbf{u}) \rightarrow \neg NES_F(\mathbf{u}))$$

is equivalent to

$$F \wedge \forall u_1 u_2 u_3((u_1, u_2, u_3) \le (p, q, r)$$
$$\wedge (\exists x\ u_1(x) \vee \exists x\ u_2(x) \vee \exists x\ u_3(x))$$
$$\rightarrow \neg \forall xy([q(x) \wedge \neg u_2(x) \rightarrow p(x) \wedge \neg u_1(x)]$$
$$\wedge [p(y) \wedge \neg u_1(y) \rightarrow q(y) \wedge \neg u_2(y)]$$
$$\wedge [\neg r(x) \rightarrow p(x) \wedge \neg u_1(x)])).$$
$$(22)$$

3. Formula in Proposition 3′ (c): Similar to (22) except that

$$\exists x\ u_1(x) \vee \exists x\ u_2(x) \vee \exists x\ u_3(x)$$

in (22) is replaced with $Loop_F(\mathbf{u})$, which is

$$SC_F(\mathbf{u}) \vee [(\exists x\ u_1(x) \vee \exists x\ u_2(x) \vee \exists x\ u_3(x))$$
$$\wedge \forall v_1 v_2 v_3(((v_1, v_2, v_3) \le (u_1, u_2, u_3)) \wedge SC_F(\mathbf{v})$$
$$\rightarrow (\exists x(v_1(x) \wedge u_2(x) \wedge \neg v_2(x))$$
$$\vee \exists y(v_2(y) \wedge u_1(y) \wedge \neg v_1(y))))],$$

where $SC_F(\mathbf{u})$ is

$$(\exists x\ u_1(x) \vee \exists x\ u_2(x) \vee \exists x\ u_3(x))$$
$$\wedge \forall v_1 v_2 v_3(((\exists x\ v_1(x) \vee \exists x\ v_2(x) \vee \exists x\ v_3(x))$$
$$\wedge (v_1, v_2, v_3) < (u_1, u_2, u_3))$$
$$\rightarrow (\exists x(v_1(x) \wedge u_2(x) \wedge \neg v_2(x))$$
$$\vee \exists y(v_2(y) \wedge u_1(y) \wedge \neg v_1(y)))).$$

Remark: Proposition 3′ tells that each of the formulas in **1**, **2**, **3** are equivalent to each other.

4. First-Order Loop Formula for Sentence F (Using NFES) : Let $Y_1 = \{p(z)\}, Y_2 = \{q(z)\}$, $Y_3 = \{r(z)\}$, $Y_4 = \{p(z), q(z)\}$. Set $\{Y_1, Y_2, Y_3, Y_4\}$ is a complete set of loops.

Under the assumption F,

- $FLF_F(Y_1)$ is equivalent to the universal closure of

$$p(z) \rightarrow \neg \forall xy([q(x) \rightarrow p(x) \wedge x \ne z]$$
$$\wedge [p(y) \wedge y \ne z \rightarrow q(y)]$$
$$\wedge [\neg r(x) \rightarrow p(x) \wedge x \ne z]).$$

- $FLF_F(Y_2)$ is equivalent to the universal closure of

$$q(z) \rightarrow \neg \forall xy([q(x) \wedge x \ne z \rightarrow p(x)]$$
$$\wedge [p(y) \rightarrow q(y) \wedge y \ne z]).$$

- $FLF_F(Y_3)$ is equivalent to the universal closure of

$$r(z) \rightarrow \bot.$$

- $FLF_F(Y_4)$ is equivalent to the universal closure of

$$p(z) \wedge q(z) \rightarrow$$
$$\neg \forall xy([q(x) \wedge x \ne z \rightarrow p(x) \wedge x \ne z]$$
$$\wedge [p(y) \wedge y \ne z \rightarrow q(y) \wedge y \ne z]$$
$$\wedge [\neg r(x) \rightarrow p(x) \wedge x \ne z]).$$

5. First-Order Loop Formula for Nondisjunctive Program (Using FES): See Example 2.

6. First-Order Loop Formula when Π is understood as an extended program (Using EFES) : Consider the same Y_i as before.

Under the assumption Π,

- $FLF_\Pi(Y_1)$ is equivalent to the universal closure of

$$p(z) \rightarrow (\exists x(q(x) \wedge \neg(p(x) \wedge x \ne z))$$
$$\vee \exists x(\neg r(x) \wedge \neg(p(x) \wedge x \ne z))).$$

- $FLF_\Pi(Y_2)$ is equivalent to the universal closure of

$$q(z) \rightarrow \exists y(p(y) \wedge \neg(q(y) \wedge y \ne z)).$$

- $FLF_\Pi(Y_3)$ is equivalent to the universal closure of

$$r(z) \rightarrow \bot.$$

- $FLF_\Pi(Y_4)$ is equivalent to the universal closure of

$$(p(z) \wedge q(z)) \rightarrow (\exists x((q(x) \wedge x \ne z) \wedge \neg(p(x) \wedge x \ne z))$$
$$\vee \exists y((p(y) \wedge y \ne z) \wedge \neg(q(y) \wedge y \ne z))$$
$$\vee \exists x(\neg r(x) \wedge \neg(p(x) \wedge x \ne z))).$$

Remark: Proposition 7 tells that the sets of formulas in each of **4**, **5**, **6** are equivalent to each other, under the assumption F. In view of Proposition 5, each set conjoined with F is equivalent to each of the formulas in **1**, **2**, **3**.