

Ordered Completion for First-Order Logic Programs on Finite Structures

Vernon Asuncion

School of Computing & Mathematics
University of Western Sydney

Fangzhen Lin

Department of Computer Science
Hong Kong University of Sci. & Tech.

Yan Zhang and Yi Zhou

School of Computing & Mathematics
University of Western Sydney

Abstract

In this paper, we propose a translation from normal first-order logic programs under the answer set semantics to first-order theories on finite structures. Specifically, we introduce ordered completions which are modifications of Clark's completions with some extra predicates added to keep track of the derivation order, and show that on finite structures, classical models of the ordered-completion of a normal logic program correspond exactly to the answer sets (stable models) of the logic program.

Introduction

This work is about translating logic programs under the answer set semantics (Gelfond & Lifschitz 1988) to first-order logic. Viewed in the context of formalizing the semantics of logic programs in classical logic, work in this direction goes back to that of Clark (1978) who gave us what is now called Clark's completion semantics, on which our work, like almost all other work in this direction, is based.

In terms of the answer set semantics, Clark's completion semantics is too weak in the sense that not all models of Clark's completion are answer sets, unless the programs are "tight" (Fages 1994). Various ways to remedy this have been proposed, particularly in the propositional case (logic programs without variables) given the recent interest in Answer Set Programming (ASP) and the prospect of using SAT solvers to compute answer sets. This paper considers first-order logic programs, and the prospect of capturing the answer sets of these programs in first-order logic.

A crucial consideration in work of this kind is whether extra symbols (in the propositional case) or predicates (in the first-order case) can be used. For propositional logic programs, Ben-Eliyahu and Dechter's translation (1994) is polynomial in space but uses $O(n^2)$ extra variables, while Lin and Zhao's translation (2004) using loop formulas is exponential in the worst case but does not use any extra variables. Chen *et al.* (2006) extended loops and loop formulas to first-order case and showed that for finite domains, the answer sets of a first-order normal logic program can be captured by its completion and all its first-order loop formulas. However, in general, a program may have an infinite number of loops and loop formulas. But this seems to be the best

that one can hope for if no extra predicates are used: it is well-known that transitive closure, which can be easily written as a first-order logic program, cannot be defined by any finite first-order theories on finite structures (Kolaitis 1990).

However, the situation is different if we introduce extra predicates. Our main technical result of this paper is that by using some additional predicates that keep track of the derivation order from bodies to heads in a program, we can modify Clark's completion into what we call *ordered completion* that captures exactly the answer set semantics on finite structures. While our emphasis in this paper is first-order logic programs, we nonetheless report some preliminary experimental results of using our ordered completion to compute answer sets of a ground logic program.

Preliminaries

We assume a finite first-order language without function symbols but with equality. Given such a language, the notions of *terms*, *atoms*, *formulas* and *sentences* are defined as usual. In particular, an atom is called an *equality atom* if it is of the form $t_1 = t_2$, and a *proper atom* otherwise.

A normal logic program is a finite set of *rules* of the following form

$$\alpha \leftarrow \beta_1, \dots, \beta_k, \text{not } \gamma_1, \dots, \text{not } \gamma_l, \quad (1)$$

where α is a proper atom, and $\beta_i, (1 \leq i \leq k), \gamma_j, (1 \leq j \leq l)$ are atoms. We call a variable in a rule a *local* variable if it occurs in the body but not the head of the rule.

Given a program Π , a predicate is called *intentional* if it occurs in the head of a rule in Π , and *extensional* otherwise. The *signature* of Π contains all intentional predicates, extensional predicates and constants occurring in Π .

For convenience and without loss of generality, in the following we assume that programs are *normalized* in the sense that for each intentional predicate P , there is a tuple \vec{x} of distinct variables matching the arity of P such that for each rule, if its head mentions P , then the head must be $P(\vec{x})$. So the rules of P in the program can be enumerated as:

$$P(\vec{x}) \leftarrow \text{Body}_1, \dots, P(\vec{x}) \leftarrow \text{Body}_k.$$

Clark's completion

Our following definition of Clark's completion is standard except that we do not make completions for extensional predicates.

Given a program Π , and a predicate P in it, *Clark's Completion* of P in Π is the following first-order sentence (Clark 1978):

$$\forall \vec{x} (P(\vec{x}) \leftrightarrow \bigvee_{1 \leq i \leq k} \exists \vec{y}_i \widehat{Body}_i), \quad (2)$$

where

- $P(\vec{x}) \leftarrow Body_1, \dots, P(\vec{x}) \leftarrow Body_k$ are all the rules whose heads mention the predicate P (recall that we assume a program is normalized);
- \vec{y}_i is the tuple of local variables in $P(\vec{x}) \leftarrow Body_i$;
- \widehat{Body}_i is the conjunction of elements in $Body_i$ by simultaneously replacing the occurrences of not by \neg .

Clark's Completion (*completion* for short if clear from the context) of Π , denoted by $Comp(\Pi)$, is then the set of Clark's completions of all intentional predicates in Π .

Example 1 [Transitive Closure (TC)] The following normal logic program TC computes the paths of a given graph:

$$\begin{aligned} S(x, y) &\leftarrow E(x, y) \\ S(x, y) &\leftarrow E(x, z), S(z, y), \end{aligned}$$

where E is the only extensional predicate of TC, representing the edges of a graph, and S is the only intentional predicate of TC. Ideally, the intentional predicate computes the transitive closure (i.e., all the paths) of a given graph. The Clark's Completion of TC is the following first-order sentence:

$$\forall xy (S(x, y) \leftrightarrow (E(x, y) \vee \exists z E(x, z) \wedge S(z, y))).$$

The answer set (stable model) semantics

The stable model semantics for normal propositional programs was proposed by Gelfond and Lifschitz (1988), and later extended to become answer set semantics for propositional programs that can have classical negation, constraints, disjunctions, and other operators. Due to space limitation, we assume familiarity with the answer set semantics for propositional logic programs.

The *answer set semantics* (or *stable model semantics*) for first-order normal logic programs *without extensional predicates* have been well studied as well. There are some different characterizations, for instance, in terms of grounding (Gelfond & Lifschitz 1988), in terms of loop formulas (Chen *et al.* 2006), in terms of circumscription (Lin & Zhou 2007), in terms of modified circumscription (Ferraris, Lee, & Lifschitz 2007), and in terms of first-order equilibrium logic (Pearce & Valverde 2004). It has been shown that all the above definitions coincide on finite structures (Ferraris, Lee, & Lifschitz 2007; Lin & Zhou 2007; Lee & Meng 2008).

Accounting for extensional predicates is straightforward (see e.g. (Chen *et al.* 2006)). Assuming one knows the answer set semantics of ground logic programs, the easiest way to define answer set semantics for a first-order logic program is by grounding, which is what we will do here. But one subtlety is the unique names assumption: whether distinct constants are interpreted differently. Here we do not need it, so we will not assume it.

Given a program Π , and a first-order structure \mathcal{M} of the signature used in Π , we use the interpretations of \mathcal{M} on the constants and extensional predicates to ground Π .

Definition 1 *The grounding of a program Π on a structure \mathcal{M} , written $\Pi_{\mathcal{M}}$ below, is the union of the following three sets:*

1. *The set of all instances of the rules in Π under \mathcal{M} , here an instance of a rule under \mathcal{M} is the result of replacing all constants in the rule by their interpretations in \mathcal{M} , and all variables in the rule by some domain objects in \mathcal{M} ;*
2. $EQ_{\mathcal{M}} = \{u = u \mid u \text{ is a domain object in } \mathcal{M}\}$;
3. $Ext_{\mathcal{M}} = \{Q(\vec{u}) \mid Q \text{ is an extensional predicate and } \vec{u} \in Q^{\mathcal{M}}\}$, here $Q^{\mathcal{M}}$ is the interpretation of Q in \mathcal{M} .

We now have the following definition:

Definition 2 *Let Π be a normal logic program and \mathcal{M} a structure. We say that \mathcal{M} is a stable model or an answer set of Π if the following set*

$$EQ_{\mathcal{M}} \cup Ext_{\mathcal{M}} \cup Int_{\mathcal{M}}$$

is an answer set of $\Pi_{\mathcal{M}}$ in the propositional case, where $Int_{\mathcal{M}}$ is the following set

$$\{P(\vec{u}) \mid P \text{ is an intentional predicate, and } \vec{u} \in P^{\mathcal{M}}\}.$$

Ordered Completion

It is well-known that Clark's completion does not fully capture the answer set semantics because of the cycles. For instance, the following program

$$\begin{aligned} p &\leftarrow q \\ q &\leftarrow p \end{aligned}$$

has one answer set $\{\}$, but its completion completion $p \leftrightarrow q$ has two models $\{p, q\}$ and $\{\}$. Here, we propose a modification of Clark's completion to address this issue. The main technical property of our new translation is that for each finite first-order logic program, our translation yields a finite first-order theory that captures exactly the finite stable models of the program. The ideas behind our translation can be best illustrated by simple propositional programs. Consider the program mentioned above. We introduce four extra symbols $T_{pq}, T_{pp}, T_{qq}, T_{qp}$ (read, e.g. T_{pq} as from p to q), and translate this program into the following theory

$$\begin{aligned} (p \rightarrow q) \wedge (q \rightarrow p), \\ q \rightarrow (p \wedge T_{pq} \wedge \neg T_{qp}), \\ p \rightarrow (q \wedge T_{qp} \wedge \neg T_{pq}), \\ T_{pq} \wedge T_{qp} \rightarrow T_{pp}, \\ T_{qp} \wedge T_{pq} \rightarrow T_{qq}. \end{aligned}$$

The first sentence is the direct encoding of the two rules. The second one is similar to Clark's completion for q except that we add T_{pq} and $\neg T_{qp}$: for q to be true, p must be true and that it must be that p is used to derive q but not the other way around. The third sentence is similar, and the last two sentences are about the transitivity of the T atoms. It can be seen that in all models of the above sentences, both p and q must be false.

Definition of ordered completion

In general, let Π be a first-order normal logic program, and Ω_Π its set of intentional predicates. For each pair of predicates (P, Q) (might be the same) in Ω_Π , we introduce a new predicate T_{PQ} , called the *comparison predicate*, whose arity is the sum of the arities of P and Q . The intuitive meaning of $T_{PQ}(\vec{x}, \vec{y})$, read as from $P(\vec{x})$ to $Q(\vec{y})$, is that there is a derivation path from $P(\vec{x})$ to $Q(\vec{y})$.

Definition 3 Let Π be a normal logic program. The ordered completion of Π , denoted by $OC(\Pi)$, is the set of following sentences:

- For each intentional predicate P , the following sentences:

$$\forall \vec{x} \left(\bigvee_{1 \leq i \leq k} \exists \vec{y}_i \widehat{Body}_i \rightarrow P(\vec{x}) \right), \quad (3)$$

$$\forall \vec{x} \left(P(\vec{x}) \rightarrow \bigvee_{1 \leq i \leq k} \exists \vec{y}_i (\widehat{Body}_i \wedge \bigwedge_{Q(\vec{z}) \in Pos_i, Q \in \Omega_\Pi} (T_{QP}(\vec{z}, \vec{x}) \wedge \neg T_{PQ}(\vec{x}, \vec{z}))) \right) \quad (4)$$

where we have borrowed the notations used in the definition of Clark's completion, and further assume that Pos_i is the positive part of $Body_i$, and $Q(\vec{z})$ ranges over all the intentional atoms in the positive part of $Body_i$;

- For each triple of intentional predicates P, Q , and R (two or all of them can be the same predicate) the following sentence:

$$\bigwedge_{P, Q, R \in \Omega_\Pi} \forall \vec{x} \vec{y} \vec{z} (T_{PQ}(\vec{x}, \vec{y}) \wedge T_{QR}(\vec{y}, \vec{z}) \rightarrow T_{PR}(\vec{x}, \vec{z})), \quad (5)$$

In the following, we use $MComp(\Pi)$ to denote the set of the formulas (3) and (4), and $Trans(\Pi)$ the set of formulas (5). So $OC(\Pi) = MComp(\Pi) \cup Trans(\Pi)$.

Clearly, for finite programs, $OC(\Pi)$ is finite, and the predicates occurring in $OC(\Pi)$ are all the predicates occurring in Π together with all the comparison predicates $\{T_{PQ} \mid P, Q \in \Omega_\Pi\}$.

Notice that Clark's completion of a predicate consists of two parts:

$$\begin{aligned} & \forall \vec{x} \left(\bigvee_{1 \leq i \leq k} \exists \vec{y}_i \widehat{Body}_i \rightarrow P(\vec{x}) \right) \\ & \forall \vec{x} \left(P(\vec{x}) \rightarrow \bigvee_{1 \leq i \leq k} \exists \vec{y}_i \widehat{Body}_i \right). \end{aligned}$$

Thus the difference between $MComp(\Pi)$ and $Comp(\Pi)$ is that the former introduces some assertions on the comparison predicates, which intuitively mean that there exist derivation paths from the intentional atoms in the body to head but not the other way around (see Equation (4)). In addition, $Trans(\Pi)$ simply means that the comparison predicates satisfy "transitivity".

Proposition 1 Let Π be a normal logic program. Then, $OC(\Pi)$ introduces m^2 new predicates whose arities are no more than $2s$, and the size of $OC(\Pi)$ is $O(s \times m^3 + s \times n)$, where m is the number of intentional predicates of Π , s the maximal arity of the intentional predicates of Π and n the length of Π .

Example 2 [Transitive Closure continued] Recall the Transitive Closure program TC presented in Example 1. In this case, since the only intentional predicate is S , we only need to introduce one additional predicate T_{SS} , whose arity is 4. The ordered completion of TC consists of the following sentences:

$$\begin{aligned} & \forall xy (E(x, y) \vee \exists z (E(x, z) \wedge S(z, y))) \rightarrow S(x, y), \\ & \forall xy S(x, y) \rightarrow (E(x, y) \vee \exists z (E(x, z) \wedge S(z, y) \\ & \quad \wedge T_{SS}(z, y, x, y) \wedge \neg T_{SS}(x, y, z, y))), \\ & \forall xyuvwz T_{SS}(x, y, u, v) \wedge T_{SS}(u, v, z, w) \\ & \quad \rightarrow T_{SS}(x, y, z, w). \end{aligned}$$

Intuitively, one can understand $T_{SS}(x, y, u, v)$ to mean that $S(x, y)$ is used to establish $S(u, v)$. So the second sentence means that for $S(x, y)$ to be true, either $E(x, y)$ (the base case), or inductively, for some z , $E(x, z)$, $S(z, y)$, and $S(z, y)$ is used to establish $S(x, y)$ and not the other way around.

To see how these axioms work, consider the graph in Figure 1 with four vertices a, b, c, d , with E representing the edge relation: $E(a, b), E(b, a), E(a, c), E(c, d)$.

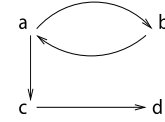


Figure 1: An example graph

Clearly, if there is a path from x to y , then $S(x, y)$ (by the first sentence above). We want to show that if there is no path from x to y , then $\neg S(x, y)$. Consider $S(d, a)$. If it is true, then since $\neg E(d, a)$, there must be an x such that

$$E(d, x) \wedge S(x, a) \wedge T_{SS}(x, a, d, a) \wedge \neg T_{SS}(d, a, x, a).$$

This is false as there is no edge going out of d .

Now consider $S(c, a)$. If it is true, then there must be an x such that

$$E(c, x) \wedge S(x, a) \wedge T_{SS}(x, a, c, a) \wedge \neg T_{SS}(c, a, x, a).$$

So x must be d , and

$$S(d, a) \wedge T_{SS}(d, b, a, b) \wedge \neg T_{SS}(a, b, d, b).$$

However, as shown above, $S(d, a)$ is false.

The main theorem

In this section, we prove the following main theorem.

Theorem 1 Let Π be a normal logic program whose signature is σ , and \mathcal{A} a finite σ -structure. Then, \mathcal{A} is an answer

set of Π if and only if there exists a model \mathcal{M} of $OC(\Pi)$ such that \mathcal{A} is the reduct¹ of \mathcal{M} on σ .

Proof:(sketch) First we show that every finite answer set \mathcal{A} of Π can be expanded to a model of $OC(\Pi)$. Construct a finite structure \mathcal{M} by expanding \mathcal{A} with the following interpretations on T_{PQ} for each pair (P, Q) of intentional predicates in Π :

$T_{PQ}(\vec{a}, \vec{b})$ iff there exists a path from $Q(\vec{b})$ to $P(\vec{a})$ in the dependency graph (see the definition in (Lin & Zhao 2003)) of the ground program $\Pi_{\mathcal{A}}$.

where \vec{a} and \vec{b} are two tuples of elements in the domain of \mathcal{A} that match the arities of P and Q respectively. It can be proved that \mathcal{M} is a model of $OC(\Pi)$.

Now we prove that the reduct of any finite model \mathcal{M} of the ordered completion of Π on σ must be an answer set of Π . Clearly, $\mathcal{M} \uparrow \sigma$ is a model of $Comp(\Pi)$. Hence, according to the loop formula characterization of answer set semantics in the propositional case (Lin & Zhao 2003), it suffices to show that for all loops L of the ground program $\Pi_{\mathcal{M} \uparrow \sigma}$, the set of ground atoms $EQ_{\mathcal{M} \uparrow \sigma} \cup Ext_{\mathcal{M} \uparrow \sigma} \cup Int_{\mathcal{M} \uparrow \sigma}$ is a model of its loop formula.

Otherwise, since \mathcal{M} is a model of $MComp(\Pi)$, we can get a sequence of ground atoms $P_0(\vec{a}_0), P_1(\vec{a}_1), P_2(\vec{a}_2), \dots$, such that for all i , $P_i(\vec{a}_i) \in L$, $\vec{a}_i \in P_i^{\mathcal{M}}$, $T_{P_{i+1}P_i}(\vec{a}_{i+1}, \vec{a}_i)$ holds in \mathcal{M} , and $T_{P_i P_{i+1}}(\vec{a}_i, \vec{a}_{i+1})$ does not hold in \mathcal{M} . Hence, for all $k < l$, $T_{P_l P_k}(\vec{a}_l, \vec{a}_k)$ holds in \mathcal{M} but $T_{P_k P_l}(\vec{a}_k, \vec{a}_l)$ does not hold since T_{PQ} satisfy transitivity for all pairs of intentional predicates. However, since \mathcal{M} is finite, there exist $k < l$ such that $P_k(\vec{a}_k) = P_l(\vec{a}_l)$, a contradiction. ■

Normal logic program with constraints

Recall that we have required the head of a rule to be a proper atom. If we allow the head to be empty, then we have so-called *constraints*:

$$\leftarrow \beta_1, \dots, \beta_k, \text{not } \gamma_1, \dots, \text{not } \gamma_l, \quad (6)$$

where $\beta_i, (1 \leq i \leq k), \gamma_j, (1 \leq j \leq l)$ are atoms. A model is said to satisfy the above constraint if it satisfies the corresponding sentence:

$$\forall \vec{y} \neg(\beta_1 \wedge \dots \wedge \beta_k \wedge \neg \gamma_1 \wedge \dots \wedge \neg \gamma_l),$$

where \vec{y} is the tuple of all variables occurring in (6). In the following, if c is a constraint of form (6), then we use \hat{c} to denote its corresponding formula above.

A normal logic program with constraints is then a finite set of rules and constraints. The answer set semantics can be extended to normal logic programs with constraints: a model is an answer set if it is an answer set of the set of the rules in the program and satisfies all the constraints in the program.

¹A σ -structure is said to be a *reduct* of a σ' -structure ($\sigma \subseteq \sigma'$) \mathcal{M} on σ , denoted by $\mathcal{M} \uparrow \sigma$, if it is the structure obtained from \mathcal{M} by removing the interpretations of the symbols in $\sigma' \setminus \sigma$ (Ebbinghaus & Flum 1995).

Both Clark's completion and our ordered completion can be extended straightforwardly to normal logic programs with constraints: one simply adds the sentences corresponding to the constraints to the respective completions.

Proposition 2 *Let Π be a normal logic program whose signature is σ , C a set of constraints, and \mathcal{A} a finite σ -structure. Then, \mathcal{A} is an answer set of $\Pi \cup C$ iff there exists a model \mathcal{M} of $OC(\Pi) \cup \{\hat{c} \mid c \in C\}$, such that \mathcal{A} is the reduct of \mathcal{M} on σ .*

Ordered completion on maximal predicate loops (strongly connected components)

In our definition of ordered completions, we introduce a comparison predicate between each pair of predicates. This is not necessary. We only need to do so for pairs of predicates that belong to a same loop in the predicate dependency graph of the program.

Formally, the *predicate dependency graph* of a first-order program Π is a finite graph $PG_{\Pi} = \langle V, E \rangle$, where V is the set of all intentional predicates of Π and $(P, Q) \in E$ iff there is a rule whose head mentions P and whose positive body mentions Q .

Maximal predicate loops are then strongly connected components of PG_{Π} . Ordered completions on maximal predicate loops are the same as ordered completions except that the comparison predicates T_{PQ} are defined only when P and Q belong to a same maximal predicate loop. More precisely, the ordered completion of Π on maximal predicate loops, denoted by $OC^*(\Pi)$, is of the similar form as the ordered completion of Π (see Definition 3), except that

- $Q(\vec{z})$ in (4) ranges over all the intentional atoms in the positive part of $Body_i$ such that for some maximal predicate loop L , both P and Q are in L .
- P, Q and R in (5) are intentional predicates such that for some maximal predicate loop L , P, Q , and R are all in L .

The following proposition is a refinement of the main theorem.

Proposition 3 *Let Π be a normal logic program whose signature is σ , and \mathcal{A} a finite σ -structure. Then, \mathcal{A} is an answer set of Π if and only if there exists a model \mathcal{M} of $OC^*(\Pi)$ such that \mathcal{A} is the reduct of \mathcal{M} on σ .*

In many cases, restricting comparison predicates on maximal predicate loops results in a much smaller ordered completion.

Example 3 [Hamiltonian Circuit (HC)] Consider the following normal program HC with constraints for computing Hamiltonian circuits of a graph:

$$\begin{aligned} hc(x, y) &\leftarrow arc(x, y), \text{not } otherroute(x, y), \\ otherroute(x, y) &\leftarrow arc(x, y), arc(x, z), hc(x, z), y \neq z, \\ otherroute(x, y) &\leftarrow arc(x, y), arc(z, y), hc(z, y), x \neq z, \\ reached(y) &\leftarrow arc(x, y), hc(x, y), reached(x), \text{not } init(x), \\ reached(y) &\leftarrow arc(x, y), hc(x, y), init(x), \\ &\leftarrow vertex(x), \text{not } reached(x). \end{aligned}$$

This program has three intentional predicates: hc , $otherroute$ and $reached$. According to the original version of ordered completion (see Definition 3), we need to

introduce 9 comparison predicates, and the maximal arity is 4.

However, by using maximal predicate loops, only one extra predicate is needed since HC has only one maximal predicate loop, namely $\{reached\}$. The only comparison predicate needed is $T_{RR}(x, y)$, which is binary. Hence, the new form of the ordered completion of HC is the following set of sentences:

$$\begin{aligned}
& \forall xy(hc(x, y) \leftrightarrow arc(x, y) \wedge \neg otherroute(x, y)), \\
& \forall xy(otherroute(x, y) \leftrightarrow \\
& \quad \exists z(arc(x, y) \wedge arc(x, z) \wedge hc(x, z) \wedge y \neq z) \vee \\
& \quad \exists z(arc(x, y) \wedge arc(z, y) \wedge hc(z, y) \wedge x \neq z)), \\
& \forall y((\exists x(arc(x, y) \wedge hc(x, y) \wedge reached(x) \wedge \neg init(x)) \vee \\
& \quad \exists x(arc(x, y) \wedge hc(x, y) \wedge init(x))) \rightarrow reached(y)), \\
& \forall y(reached(y) \rightarrow (\exists x(arc(x, y) \wedge hc(x, y) \wedge init(x)) \vee \\
& \quad \exists x(arc(x, y) \wedge hc(x, y) \wedge reached(x) \wedge \neg init(x) \wedge \\
& \quad \quad T_{RR}(x, y) \wedge \neg T_{RR}(y, x)))), \\
& \forall x(\neg(vertex(x) \wedge \neg reached(x))), \\
& \forall xyz(T_{RR}(x, y) \wedge T_{RR}(y, z) \rightarrow T_{RR}(x, z)).
\end{aligned}$$

Arbitrary structures

Finally, we want to emphasize that the correspondence between classical first-order models of our ordered completions and stable models of a logic program holds only on finite structures. In general, the result doesn't hold on arbitrary structures. For instance, on arbitrary structures, the transitive closure program cannot be captured by a first-order sentence with or without using new predicates.

Related Work and Discussions

The only other translations from first-order logic programs under answer set semantics to first-order logic are based on loop formulas (Chen *et al.* 2006; Lee & Meng 2008). As mentioned earlier, the main difference between these translations and ours is that ours results in a finite first-order theory but uses extra predicates while the ones based on loop formulas do not use any extra predicates but in general result in an infinite first-order theory.

As we also mentioned, the basic intuitions behind almost all of the current translations from logic programs with answer set semantics to classical logic are similar. The main differences are in the ways these intuitions are formalized. In the following, we briefly review some of the closely related ones in the propositional case.

Propositional case

Ordered completion can be viewed as a propositional translation from normal logic programs to propositional theories by treating each propositional atom as a 0-ary predicate. Several proposals in this direction have been proposed (Ben-Eliyahu & Dechter 1994; Lin & Zhao 2003; Janhunen 2004; Lin & Zhao 2004; Niemelä 2008). Ben-Eliyahu and Dechter (1994) assigned an index (or level numbering) $\#x$ to each propositional atom x , and added the assertions $\#x < \#y$ to the Clark's completion for each pair (x, y) , where x is the

head of a rule and y is in the positive body of the rule. Janhunen (2004) proposed another similar translation and implemented an ASP solver called "lp2atomic|lp2sat". More recently, Niemelä (2008) proposed to capture the level mapping in difference logic, and based on it, designed an ASP solver called lp2diff using an SMT solver that integrates SAT with a difference logic module.

The main difference between Ben-Eliyahu and Dechter's translation and ours is that we use the comparison atoms T_{xy} instead of indices $\#x$. In fact, $T_{xy} \wedge \neg T_{yx}$ in ordered completion plays the role as $\#x < \#y$ in Ben-Eliyahu and Dechter's translation. Although they look similar in the Clark's completion part, the ways to encode indices and comparison atoms in classical propositional logic are very different. As the difference to Niemelä's work, which formalized the notion of groundedness using the build-in predicate " $<$ " in difference logic, we did it by introducing some new (comparison) predicates in classical logic. This difference shows up in the implementations as well: while Niemelä uses SMT, we use SAT.

Another translation, also sharing the basic idea of comparing stages, is due to Lin and Zhao (2003). However, they first translate a program equivalently to a tight program, and then use the Clark's completion of the new program to capture the original one. Also, the loop formula approach in the propositional case (Lin & Zhao 2004) can be regarded as a translation from logic programs to propositional theories. Different from the ones mentioned above, the loop formula approach requires no extra atoms but may be exponential. Based on this idea, two solvers called "assat" (Lin & Zhao 2004) and "cmodels" (Lierler & Maratea 2004) are implemented by adding loop formulas incrementally.

Fixed-point logic and Datalog

Another related work (Kolaitis 1990) is in the area of finite model theory and fixed-point logic. Although fixed-point logic and normal logic programming are not comparable, they have a common fragment, namely Datalog. Kolaitis (1990) showed that every fixed-point query is conjunctive definable on finite structures. That is, given any fixed-point query Q , there exists another fixed-point query Q' such that the conjunctive query (Q, Q') is implicit definable on finite structures. As a consequence, every datalog query is also conjunctive definable on finite structures. From this result, although tedious, one can actually derive a translation from datalog to first-order sentences using some new predicates not in the signatures of the original datalog programs.

We will not go into details comparing our translation and the one derived from Kolaitis' result since our focus here is on normal logic programs. Suffice to say here that the two are different in many ways, not the least is that ours is based on Clark's completion in the sense that some additional conditions are added to the necessary parts of intentional predicates, while the one derived from Kolaitis' result is not. We mention this work because Kolaitis' result did play an important role early on in our work. We speculated that if it is possible to translate datalog programs to first-order sentences using some new predicates, then it must also be possible for normal logic programs, and that if this is true, then

it must be doable by modifying Clark’s completion. As it happened, this turned out to be the case.

Some Experimental Results

While our interest is mainly on first-order normal logic programs, and the possibility of constructing a first-order ASP solver, as an easy exercise, we implemented a prototype propositional ASP solver, called `asp2sat`, using our ordered completion. Our preliminary experimental results seem to indicate that while not as good as `clasp`², it is quite competitive with other SAT or SMT-based ones.

Table 1 contains some runtime data on Niemelä’s Hamiltonian Circuit program with the particular instances taken from the `assat` website³. The following solvers were compared: our `asp2sat` using either `zchaff` (3.12)⁴ or `minisat` (2.0)⁵; `lp2atomic` (1.12)⁶ with `minisat` and `lp2diff` (1.10)⁷ with `z3`; `cmodels` (3.79)⁸ with `zchaff`, and `clasp`. In the table, “y” (“n”) means that the corresponding graph has a (no) Hamiltonian Circuit. For each instance, we record the average time for 5 runs in seconds. We set the timeout threshold as 900 seconds, which is denoted by “-” in the table.

graph		asp2sat +zchaff	asp2sat +minisat	lp2atomic +minisat	lp2diff +z3	cmodels	clasp
nv60a356	y	1.04	0.35	0.27	0.13	3.01	0.02
nv60a526	y	3.21	3.11	1.06	0.21	1.82	0.09
nv60a554	y	7.33	16.06	1.95	0.26	0.39	0.04
nv70a396	y	0.43	0.95	0.55	0.16	7.26	0.04
nv70a428	y	1.52	2.08	0.67	0.20	0.68	0.03
nv70a511	y	1.20	15.57	1.52	0.28	2.27	0.04
nv70a549	y	3.05	17.12	1.91	0.24	0.33	0.04
nv70a571	y	4.35	5.35	5.52	0.24	0.33	0.04
2xp30.1	y	4.04	4.97	2.60	28.83	8.44	0.01
2xp30.3	y	2.56	8.40	580.93	2.29	17.31	0.04
2xp30.4	n	17.56	18.66	-	198.58	217.23	8.33
4xp20	n	0.00	0.09	118.27	2.82	0.14	0.00
4xp20.1	n	0.52	0.19	113.12	2.55	0.65	0.08
4xp20.2	y	1.40	1.26	126.51	0.28	0.90	0.02
4xp20.3	n	6.10	0.62	122.50	3.36	1.38	0.01

Table 1: Experimental results

Conclusion

The main contribution of this paper is to introduce a notion of ordered completion that captures the answer set semantics of normal logic programs on finite structures (See Theorem 1). Interestingly, this result fails if infinite structures are allowed. Theorem 1 is also surprising in the sense that many logic programs cannot be captured by first-order sentences without extra predicates (e.g. TC in Example 1).

²<http://www.cs.uni-potsdam.de/clasp/>

³<http://assat.cs.ust.hk/Assat-2.0/hc-2.0.html>

⁴<http://www.princeton.edu/~chaff/zchaff.html>

⁵<http://www.minisat.se/>

⁶<http://www.tcs.hut.fi/Software/lp2sat/>

⁷<http://www.tcs.hut.fi/Software/lp2diff/>

⁸<http://www.cs.utexas.edu/~tag/cmodels/>

Theorem 1 is important from both a theoretical and a practical point of view. To the best of our knowledge, our translation from first-order normal logic programs to first-order sentences is the first such one. It is also worth noting that this translation, when instantiated in the propositional case in an obvious way, yields an ASP solver that is competitive with other ASP solvers based on SAT and SMT solvers. More interestingly, we are looking at the possibility of using Theorem 1 to construct a first-order ASP solver. For us, this is the most important future direction of this work.

References

- Ben-Eliyahu, R., and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.* 12(1-2):53–87.
- Chen, Y.; Lin, F.; Wang, Y.; and Zhang, M. 2006. First-order loop formulas for normal logic programs. In *KR’06*, 298–307.
- Clark, K. L. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logics and Databases*. New York: Plenum Press. 293–322.
- Ebbinghaus, H. D., and Flum, J. 1995. *Finite Model Theory*. Springer-Verlag.
- Fages, F. 1994. Consistency of Clark’s completion and existence of stable of stable models. *Journal of Methods of Logic in Computer Science* 1:51–60.
- Ferraris, P.; Lee, J.; and Lifschitz, V. 2007. A new perspective on stable models. In *IJCAI’07*, 372–379.
- Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *ICLP’88*, 1070–1080.
- Janhunen, T. 2004. Representing normal programs with clauses. In *ECAI’04*, 358–362.
- Kolaitis, P. G. 1990. Implicit definability on finite structures and unambiguous computations (preliminary report). In *LICS’90*, 168–180.
- Lee, J., and Meng, Y. 2008. On loop formulas with variables. In *KR’08*, 444–453.
- Lierler, Y., and Maratea, M. 2004. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *LP-NMR’04*, 346–350.
- Lin, F., and Zhao, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *IJCAI’03*, 853–858.
- Lin, F., and Zhao, Y. 2004. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.* 157(1-2):115–137.
- Lin, F., and Zhou, Y. 2007. From answer set logic programming to circumscription via logic of GK. In *IJCAI’07*, 441–446.
- Niemelä, I. 2008. Stable models and difference logic. *Ann. Math. Artif. Intell.* 53(1-4):313–329.
- Pearce, D., and Valverde, A. 2004. Towards a first order equilibrium logic for nonmonotonic reasoning. In *JELIA’04*, 147–160.