Ordered Completion for Logic Programs with Aggregates

Vernon Asuncion, Yan Zhang and Yi Zhou
Intelligent Systems Laboratory
School of Computing, Engineering and Mathematics
University of Western Sydney, NSW, Australia

Abstract

In this paper, we show that first-order logic programs with monotone aggregates under the stable model semantics can be captured in classical first-order logic. More precisely, we extend the notion of ordered completion for logic programs with a large variety of aggregates so that every stable model of a program with aggregates corresponds to a classical model of its enhanced ordered completion, and vice versa.

Introduction

In the last three decades, Answer Set Programming (ASP) has emerged as a predominant declarative programming paradigm in the area of knowledge representation and logic programming (Baral 2003). One of the main focuses of recent advances in ASP is first-order answer set programming (Ferraris, Lee, and Lifschitz 2011; Lin and Zhou 2011), which aims to characterize the answer set or stable model semantics of logic programs directly on a first-order level. This is significantly different from the traditional approach by grounding (Gelfond and Lifschitz 1988), which is essentially propositional.

The stable model semantics of first-order logic programs is defined in second-order logic (Ferraris, Lee, and Lifschitz 2011; Lin and Zhou 2011). Interestingly, Asuncion et al. (2012) recently proposed a notion of ordered completion (a first-order sentence) for normal logic programs, and showed that the stable models of a normal logic program are exactly corresponding to the models of its ordered completion on finite structures. This work is not only theoretically interesting but also practically relevant as it initiates a new direction of ASP solver. A first implementation (Asuncion et al. 2012) shows that this new direction is promising as it performs surprisingly good for the Hamiltonian Circuit program on big instances.

However, this work is still rather restricted, at least from a practical point of view, because it cannot handle aggregates - a very important building block in ASP that is widely used in many applications. The reason why aggregates are crucial in answer set solving is twofold. First, it can simplify the representation task. For many applications, one can write a simpler and more elegant logic program by using aggregates, for instance, the job scheduling program (Pontelli, Son, and Elkabani 2004). More importantly, it can improve the efficiency of ASP solving (Gebser et al. 2009). Normally, the program using aggregates can be solved much faster (Faber et al. 2008).

Incorporating aggregates in ordered completion is a challenging task. In the propositional case, the complexity of normal logic programs with arbitrary aggregates is beyond the class of NP (Ferraris 2011). This possibly suggests that, most likely, normal logic programs with arbitrary aggregates cannot be captured in first-order logic. Hence, we are mainly focused on (anti)monotone aggregates. Even for this case, the task is still very complicated as aggregate atoms, on one hand, can express some features of existential quantifiers, and on the other hand, contribute to the loops (Chen et al. 2006; Lee and Meng 2009) of the program.

In this paper, we show that first-order normal logic programs with a large variety of (anti)monotone aggregates (covering the types of aggregates in most benchmark programs) can indeed be captured in first-order logic. More precisely, we extend the notion of ordered completion for these kind of programs so that every stable model of a program is corresponding to a classical model of its enhanced ordered completion, and vice versa. Our proof technique is significantly different from the original proof for ordered completion in (Asuncion et al. 2012), thus also sheds new insights in first-order ASP (with aggregates).

Preliminaries

We first consider a first-order language without functions but with equality =. A signature contains a set of constants and a set of predicates. A structure $A$ of a signature is a tuple

$(Dom(A), c_1^A, \ldots, c_l^A, P_1^A, \ldots, P_m^A)$,

where $c_i^A, 1 \leq i \leq l$, and $P_j^A, 1 \leq j \leq m$ are interpretations for constant $c_i$ and predicate $P_j$ respectively. We call a structure finite if $Dom(A)$ is finite.

Aggregates

Now we introduce the aggregate component. For this purpose, we enhance our first-order language with a background theory for numbers, similar to Satisfiability Modular Theories (SMT) (Nieuwenhuis, Oliveras, and Tinelli 1999).
An aggregate atom \( \delta \) is of the form
\[
\text{OP}(v : \exists w Bd(\delta)) \leq N, \tag{1}
\]
where:
- \( \text{OP} \in \{\text{CARD, SUM, PROD, MIN, MAX}\} \) is an aggregate function for cardinality, sum, product, minimum, and maximum respectively;
- \( Bd(\delta) \) (the body of \( \delta \)) is of the form
\[
Q_1(y_1) \land \cdots \land Q_s(y_s) \land \neg R_1(z_1) \land \cdots \land \neg R_t(z_t), \tag{2}
\]
where each \( Q_i(y_i) \) (\( 1 \leq i \leq s \)), \( R_j(z_j) \) (\( 1 \leq j \leq t \)) are atoms in the first-order language;
- \( v \) and \( w \) are distinctive tuples of variables mentioned in (2), and \( v \cap w = \emptyset \);
- \( \leq \in \{<, \leq, =, \geq, >\} \) is a comparison operator on numbers;
- finally \( N \) is a number.

For convenience, we use \( \text{Pos}(\delta) \) and \( \text{Neg}(\delta) \) to denote the sets \( \{Q_1(y_1), \ldots, Q_s(y_s)\} \) and \( \{R_1(z_1), \ldots, R_t(z_t)\} \) respectively.

For an aggregate atom \( \delta \) of the form (1), let \( y \) be the set of free variables in it, i.e., the variables in \( \delta \) but not in either \( v \) or \( w \). Let \( A \) be a structure and \( a \) a set of domain elements that matches \( y, A \models \delta[y/a] \) if and only if:

1. The multiset\(^1\)
\[
M = \{\{a[1] \mid A \models Bd(\delta)|ywv\abc, b \in X, c \in Y\}\}
\]
is in the domain of \( \text{OP} \), where \( X \) and \( Y \) are the sets of domain tuples that match \( w \) and \( v \) respectively;

2. \( \text{OP}(M) \leq N \).

Based on this, the satisfaction relation between structures (along with assignments on variables) and formulas (with aggregates) is defined recursively as usual in classical logic.

An aggregate function \( \text{OP} \) with respect to a comparison operator \( \leq \) and number \( N \) is said to be: **monotone** if \( M_1, M_2 \in \text{Dom}(\text{OP}) \) (i.e., in the domain of \( \text{OP} \)), \( M_1 \leq M_2 \), and \( \text{OP}(M_1) \leq N \) imply that \( \text{OP}(M_2) \leq N \); and **anti-monotone** if \( M_1, M_2 \in \text{Dom}(\text{OP}) \), \( M_2 \leq M_1 \), and \( \text{OP}(M_1) \leq N \) imply that \( \text{OP}(M_2) \leq N \).

### Stable Models Semantics for Logic Programs

In this paper, we consider normal logic programs with aggregates mentioned above. A **rule** \( r \) is of the form
\[
\beta_1 \land \cdots \land \beta_n \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \rightarrow \alpha, \tag{3}
\]
where \( \alpha \) is an atom \( P(x) \); \( \beta_i \) (\( 1 \leq i \leq n \)) and \( \gamma_j \) (\( 1 \leq j \leq m \)) are either an equality atom \( t_1 = t_2 \), or an atom \( P(t) \), or an aggregate atom of the form (1). In the literature, this form is also written as
\[
\alpha \leftarrow \beta_1, \ldots, \beta_n, \text{not } \gamma_1, \ldots, \text{not } \gamma_m.
\]

For convenience, we use \( \text{Head}(r) \) and \( \text{Body}(r) \) to denote \( \alpha \) and \( \beta_1 \land \cdots \land \beta_n \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \) respectively.

A program is a finite set of rules. The signature of a program \( II \), denoted by \( \tau(\Pi) \), consists of all constants and predicates occurred in \( II \). A predicate in a program is said to be **intensional** if it occurs in some head of the program, and **extensional** otherwise. We use \( \mathcal{P}_{\text{int}}(\Pi) \) to denote the set of all intensional predicates of \( \Pi \).

**Example 1** Consider the following program \( II \) with a simple aggregate atom
\[
\text{CARD}(x : P(x)) = 2 \rightarrow Q(y), \tag{4}
\]
\[
Q(x) \rightarrow P(x), \tag{5}
\]
\[
R(x) \rightarrow P(x). \tag{6}
\]
Here, \( P \) and \( Q \) are intensional while \( R \) is extensional. Rule (4) states that if \( P(x) \) holds exactly for two elements, then we have \( Q(y) \).

The stable model semantics of programs (with aggregates) is defined by a second-order sentence. Let \( \Pi \) be a program and \( \mathcal{P}_{\text{int}}(\Pi) = \{P_1, \ldots, P_n\} \). Let \( U = \{U_1, \ldots, U_n\} \) be a set of new predicates such that each \( U_i, 1 \leq i \leq n \), matches the arity of \( P_i \). Let \( p = P(x) \) be an atom. By \( p^* \), we denote \( U_i(x) \) if \( P = P_i \), and \( P(x) \) itself if \( P \in \mathcal{P}_{\text{int}}(\Pi) \).

Let \( \delta \) be an aggregate atom of form (1). By \( \delta^* \), we denote the formula
\[
(\text{OP}(v : \exists w Bd(\delta^*)) \leq N) \land (\text{OP}(v : \exists w Bd(\delta)) \leq N),
\]
where \( Bd(\delta)^* = Q_1^*(y_1) \land \cdots \land Q_s^*(y_s) \land \neg R_1(z_1) \land \cdots \land \neg R_t(z_t) \). Now given a rule \( r \) of the form (3), by \( r^* \), we denote the universal closure of the following formula
\[
\beta_1^* \land \cdots \land \beta_n^* \land \neg \gamma_1 \land \cdots \land \neg \gamma_m \rightarrow \alpha^*.
\]

Finally, by \( SM(\Pi)^2 \), we denote the following second-order sentence
\[
\bigwedge_{r \in \Pi} \bar{r} \land \neg \exists U(U < \mathcal{P}_{\text{int}}(\Pi) \land \bigwedge_{r \in \Pi} r^*), \tag{7}
\]
where \( \bar{r} \) is the universal closure of \( r \) and \( U < \mathcal{P}_{\text{int}}(\Pi) \) is the abbreviation of the formula
\[
\bigwedge_{1 \leq i \leq n} \forall x(U_i(x) \rightarrow P_i(x)) \land \bigwedge_{1 \leq i \leq n} (P_i(x) \rightarrow U_i(x)).
\]

**Definition 1 (stable model)** Let \( \Pi \) be a program. A \( \tau(\Pi) \)-structure \( A \) is said to be a stable model of \( \Pi \) if \( A \) is a model of \( SM(\Pi) \).

### Ordered Completion

Asuncion et al. (2012) showed that the stable models semantics for normal logic programs without aggregates can be captured in classical first-order logic on finite structures. More precisely, they defined the notion of ordered completion, which is a modification of Clark’s completion by adding the track of the derivation order, and showed that the finite stable models of a normal program corresponds exactly to the models of its ordered completion.

\(^{2}\)Notice that this definition is essentially equivalent to the first-order stable model semantics as defined in (Bartholomew, Lee, and Meng 2011) but restricted to normal programs for simplicity.
Let $\Pi$ be a program without aggregate atoms. The modified completion of $\Pi$, denoted by $M\text{Comp}(\Pi)$, is
\[
\hat{\Pi} = \bigwedge_{r \in \Pi} \forall x (P(x) \rightarrow \bigvee_{r \in \Pi} \exists y (\text{Body}(r) \land Pos(r) < P(x))) \land \bigwedge_{r \in \Pi} \forall x \forall y \forall z \exists \tau (\text{Head}(r) \rightarrow \bigwedge_{r \in \Pi} \exists y (\text{Body}(r) \land Pos(r) < P(x)))
\]
where $\hat{\Pi}$ is the enhanced ordered completion and $Pos(r) < P(x)$ is\(^3\)
\[
\bigwedge_{Q(y) \in \text{Pos}(r), Q \in \text{P}_\text{int}(\Pi)} (\leq_{PQ} (xy) \land \neg \leq_{PQ} (xy)),
\]
where $\leq_{PQ}$ and $\leq_{PQ}$ are new predicates called comparison predicates for keeping track of the derivation order. Finally, the ordered completion of $\Pi$, denoted by $OC(\Pi)$, is $M\text{Comp}(\Pi) \land \text{Trans}(\Pi)$, where $\text{Trans}(\Pi)$ is
\[
\bigwedge_{P,Q,R \in \text{P}_\text{int}(\Pi)} \forall xyz (\leq_{PQ} (xy) \land \leq_{PR} (yz) \rightarrow \leq_{PR} (xz)).
\]

**Theorem 1** (Asuncion et al. 2012) Let $\Pi$ be a normal logic program without aggregate atoms and $\mathcal{A}$ a finite $\tau(\Pi)$-structure. Then, $\mathcal{A}$ is a stable model of $\Pi$ if and only if $\mathcal{A}$ can be expanded to a model of $OC(\Pi)$.

**Ordered Completion for Programs with Aggregates**

From a theoretical point of view, ordered completion makes important progressions on understanding first-order answer set programming. Firstly, it shows that the stable model semantics is simply Clark’s completion plus derivation order. Secondly, it clarifies the relationship between first-order normal ASP and classical first-order logic. More precisely, every normal answer set program can be captured by a first-order sentence with some new predicates on first order structures. Surprisingly, this fails to be true on infinite structures or if no new predicates are allowed (Asuncion et al. 2012).

Ordered completion is not only theoretically interesting but also practically important. It initiates a new direction of ASP solver by first translating a normal logic program to its ordered completion, then working on finding a model of this first-order sentence (Asuncion et al. 2012). A first implementation shows that this new direction is promising as it performs surprisingly good on the Hamiltonian Circuit program (Niemelä 1999), especially on big instances.

However, ordered completion can hardly be used beyond the Hamiltonian Circuit program because it cannot handle aggregates - a very important building block in ASP that is widely used in many applications. As far as we are concerned, most benchmark programs contain aggregate constructs (Calimeri et al. 2011). Aggregates are crucial in answer set solving because on one hand, it can simplify the representation task, and on the other hand, it can improve the efficiency of ASP solving (Gebser et al. 2009).

In this paper, we consider to incorporate aggregates in ordered completion. However, this is a challenging task. As shown in (Asuncion et al. 2012), ordered completion cannot be extended for disjunctive programs because they are not in the same complexity level. Similarly, normal answer set programs with arbitrary aggregates has the same complexity level as disjunctive programs, which is beyond the complexity of naive normal programs. This suggests that, most likely, normal logic programs with arbitrary aggregates cannot be captured in first-order logic, thus not by ordered completion.

Hence, we are mainly focused on some special classes of aggregates, for instance, (anti)monotone aggregates. Even for this case, the task is still very complicated. One observation is that aggregate atoms can express some features of existential quantifiers, which is even more complicated than disjunctions in first-order ASP. For instance, recall the program specified in Example 1. The first rule, i.e., rule (4), is actually equivalent to the following rule $\exists x \exists z (\neg u \land \neg v \land u \land v \land \neg x \land \neg z)$, meaning that $P(x)$ holds exactly for 2 elements.

Another observation is that aggregate atoms that are not anti-monotone\(^4\) contribute to the first-order loops (Chen et al. 2006; Lee and Meng 2009) of the program.\(^5\) Again, consider the program in Example 1. If we ignore the aggregate atoms, then this program has no loops. But the stable models of the program cannot be captured by its Clark’s completion (Clark 1978). This means that the aggregate atom in rule (4) indeed plays a role to form a loop of the program.

Hence, the difficult part is in identifying the gap between the aggregates that can be incorporated into ordered completion and those that cannot. In this paper, we show that a large variety of (anti)monotone aggregates can indeed be incorporated in ordered completion, which covers the types of aggregates in most benchmark programs as far as we have checked. We extend the notion of ordered completion to normal logic programs with such kind of aggregates so that every stable model of a program corresponds to a classical model of its enhanced ordered completion, and vice versa.

Formally, the aggregate functions we considered in this paper are restricted as follows.

**Definition 2** For an aggregate atom of the form (1) where $OP \in \{\text{CARD, SUM, PROD, MIN, MAX}\}$,
- **CARD** is a function from multisets of domain tuples to $\mathbb{Z}^+$, and it is defined as $0$ on the empty multiset $\emptyset$;
- **SUM** is a function from multisets of $\mathbb{Z}^+$ to $\mathbb{Z}^+$, and it is also defined as $0$ on $\emptyset$;
- **MIN** and **MAX** are functions from multisets of $\mathbb{Z}$ to $\mathbb{Z}$, and are undefined for $\emptyset$;
- **PROD** is a function from multisets of $\mathbb{N}$ to $\mathbb{N}$, and is defined as $1$ on $\emptyset$.

Now, we define the ordered completion for normal programs with aggregate atoms of form (1) under the restrictions in Definition 2. For convenience, we use $\text{PosAgg}(r)$ to denote the aggregate atoms from $\text{Pos}(r)$; $\text{PosCardSumProd}(r)$ to denote the cardinality, sum, and
\[^3\text{Pos}(r)$ denotes the positive body atoms of $r$ as usual.\n\[^4\text{Aggregate atoms that can enforce a non-empty multiset.}\n\[^5\text{For space reasons, the readers are referred to (Chen et al. 2006) for the formal definitions about loops and some related notions.}\]
product aggregates from $\text{Pos}(r)$; and $\text{PosMinMax}(r)$ to denote the minimum and maximum aggregate atoms from $\text{Pos}(r)$ respectively.

**Definition 3 (ordered completion with aggregates)** Let $\Pi$ be a program with aggregate atoms of form (1) under the restrictions in Definition 2. The modified completion of $\Pi$, denoted by $\text{MComp}(\Pi)$, is

$$\Pi \land \forall x (\text{Pos}(r) \land \text{PosAgg}(r) \land r \in M \text{Head}(r) \Rightarrow \text{Pos}(r) < P(x) \land \text{PosAgg}(r) < P(x))$$

where:

- $\Pi = \bigwedge_{r \in \Pi} \hat{r}$;
- $\text{Pos}(r) < P(x)$ is

$$\bigwedge_{Q(y) \in \text{Pos}(r) \land \text{PosAgg}(r), \hat{Q} \in \text{Pos}(\Pi)} (\leq_{OPQ}(xy) \land \neg \leq_{OPQ}(xy))$$

- and $\text{PosAgg}(r) < P(x)$ is

$$\bigwedge_{\delta \in \text{PosCardSumProd}(r), \delta \in \{\leq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}, \geq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}\}} \text{Pos}(\delta) < P(x) \land \bigwedge_{\delta \in \text{PosMinMax}(r), \delta \in \{\leq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}, \geq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}\}} \text{Pos}(\delta) < P(x) \land \bigwedge_{\delta \in \text{PosMinMax}(r), \delta \in \{\leq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}, \geq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}\}} \text{Pos}(\delta) < P(x)$$

Finally, the ordered completion of $\Pi$, denoted by $\text{OC}(\Pi)$, is again $\text{MComp}(\Pi) \land \text{Trans}(\Pi)$.

Let us take a closer look at Definition 3. First of all, for non-aggregate atoms, we treat them the same way as in the original definition of ordered completion. For aggregate atoms, we distinguish two cases. For those atoms $\text{Op}(v : \exists w B_d(\delta)) \leq N$ where $\text{Op} \in \{\text{CARD}, \text{SUM}, \text{PROD}\}$ and $\delta \in \{\leq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}, \geq_{\leq_{\leq_{\geq_{\geq_{\geq}}}}}\}$ and those atoms negatively occurring in the rule, we do not need to pay extra attentions. However, for the rest of the positive aggregate atoms $\delta$, we need to enforce the comparison assertions, i.e., formula (10). This is because, for the latter kind of aggregate atoms, we need to keep track of the derivation order. However, in the former kind, this is not necessary because they do not enforce the condition of a non-empty multiset, i.e., since anti-monotone.

**Example 2 (Example 1 continued)** Consider again the program in Example 1. Then, $\text{OC}(\Pi)$ is

$$\forall y \text{CARD}(x : P(x)) = 2 \rightarrow Q(y))$$

$$\land \forall x (Q(x) \rightarrow P(x))$$

$$\land \forall x (Q(x) \rightarrow P(x)) \lor \forall x (R(x) \rightarrow P(x))$$

$$\land \forall y (Q(y) \rightarrow \text{CARD}(x : P(x)) = 2)$$

$$\land \text{Trans}(\Pi),$$

$$\land \text{Trans}(\Pi),$$

where $Q(x) < P(x)$ and $P(x) < Q(y)$ are the shorthands of $\leq_{OPQ}(x, x) \land \neg \leq_{OPQ}(x, y)$ and $\leq_{OPQ}(x, y) \land \neg \leq_{OPQ}(y, x)$ respectively. Notice that $\text{OC}(\Pi)$ can be represented in first-order logic as formula (11) is equivalent to

$$\forall y (\exists^2 x P(x) \rightarrow Q(y))$$

and formula (14) is equivalent to

$$\forall y (Q(y) \rightarrow \exists^2 x (P(x) \land P(x) < Q(y)))$$

Consider a structure $\hat{A}$ where $R_A^A = \emptyset$. Then, $\hat{A}$ is a stable model of $\Pi$ if $P_A^A = Q_A^A = \emptyset$. This in fact corresponds to a model of $\text{OC}(\Pi)$. Otherwise, suppose $P_A^A \neq \emptyset$, say $P_A^A = \{a, b\}$. Then since we have $P(a)$, by (13), $Q(a) \land Q(a) < P(a)$ holds. Then by (14), $\exists^2 x (P(x) \land P(x) < Q(a))$ must also hold. Then since $P_A^A = \{a, b\}$ exactly contains 2 elements, then $P(a) < Q(a)$ must hold as well. This contradicts to $\text{Trans}(\Pi)$ and $Q(a) < P(a)$.

Consider a structure $\hat{A}$ with $R_A^A = \{a, b\}$. Then, $\hat{A}$ is a stable model of $\Pi$ if $\text{Dom}(A)$ only contains two elements and $P_A^A = Q_A^A = \{a, b\}$. We consider the fact that corresponds to the model of $\text{OC}(\Pi)$ as well. On one hand, if $\text{Dom}(A)$ only contains two elements and $P_A^A = Q_A^A = \{a, b\}$, it can be easily extended to a model of $\text{OC}(\Pi)$ by forcing $\forall x y P(x) < Q(y)$ to be true. On the other hand, suppose $\text{Dom}(A)$ contains more than 2 elements. Then, $\text{OC}(\Pi)$ has no model. There are two cases:

**Case 1**: $P_A^A = \{a, b\}$. Then, by (11), $Q$ holds for all domain elements. By (12), $P$ holds for all domain elements as well, a contradiction.

**Case 2**: $P_A^A \neq \{a, b\}$. Then, by (12), $\{a, b\} \subset P_A^A$. By (13), $Q_A^A$ is not empty. Then by (14), $\exists^2 x P(x)$ holds, a contradiction.

In fact, it can be verified that a finite structure $\hat{A}$ is a stable model of $\Pi$ if it can be expanded to a model of $\text{OC}(\Pi)$.

In general, we have the following main theorem.

**Theorem 2 (main theorem)** Let $\Pi$ be a program with aggregate atoms of form (1) under the restrictions in Definition 2, and $A$ a finite $\tau(\Pi)$-structure. Then, $A$ is a stable model of $\Pi$ if and only if $\hat{A}$ can be expanded to a model of $\text{OC}(\Pi)$.

The proof of this theorem is rather technical and tedious. However, the basic ideas are simple. Here, the notion of externally supported set plays a crucial role. Roughly speaking, a set of grounded atoms is externally supported if there exists a grounded atom in it and an associated rule that supports this atom (i.e., this atom is the head of the grounded rule) and whose positive body could be satisfied by external grounded atoms (i.e., grounded atoms not in this set). Then, we show that a structure is a stable model of a program if and only if it is a model of the program and every subset of grounded atoms included in this structure is externally supported. Furthermore, it is also equivalent to the fact that this structure can be expanded to a model of the ordered completion of the program.

For this purpose, we first introduce the following notion. For a program $\Pi$ and structure $\hat{A}$ of signature $\sigma$ such that

$^6$Grounded atoms are of the form $P(a)$, where $P$ is a predicate and $a$ is a tuple of domain elements matching the arity of $P$. 694
\(\tau(\Pi) \subseteq \sigma\), by \([P_{\text{int}}(\Pi)]^A\), we denote the set of grounded atoms \(\{ P(a) \mid a \in P^A, P \in P_{\text{int}}(\Pi) \}\).

**Definition 4 (Externally supported set)** Let \(\Pi\) be a program with aggregate atoms of form (1) and \(A\) a structure of \(\sigma\) such that \(\tau(\Pi) \subseteq \sigma\). We say that a set \(S \subseteq [P_{\text{int}}(\Pi)]^A\) is externally supported (under \(A\) and \(\Pi\)) if there exist some \(P(a) \in S\) and rule \(\text{Body}(r) \rightarrow P(x) \in \Pi\) with local variables \(y_r\), such that for some assignment of the form \(x_y \mapsto a\).

1. \(A \models \text{Body}(r)[xy_r/ab]_r\);
2. \((\text{Pos}(r) \setminus \text{Pos}(\text{agg}))[xy_r/ab]_r \cap S = \emptyset\);
3. For all aggregate atom \(\delta \in \text{Pos}(\text{agg})\) of the form (1),
   \[
   \text{Op}(c_v[1] : A) = B(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset, N,\]
   where \(\alpha\) is the assignment of the form \(xy_r \mapsto ab\).

   By Definition 4, a set \(S \subseteq [P_{\text{int}}(\Pi)]^A\) is not externally supported if for all \(P(a) \in S\), the rule \(\text{Body}(r) \rightarrow P(x) \in \Pi\) with local variables \(y_r\), and assignments \(x_y \mapsto ab\) such that \(A \models \text{Body}(r)[xy_r/ab]\) (i.e., a “support” for \(P(a)\)), there exists some atom \(\beta \in \text{Pos}(r)\) such that

1. \(\beta = Q(y)\), where \(Q \in P_{\text{int}}(\Pi)\) and \(Q(b) \in S\) is a further elaboration \(\beta[xy_r/ab]\), or
2. \(\beta\) is a aggregate atom \(\delta\) of the form (1) and
   \[
   \text{Op}(c_v[1] : A) = B(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset, N,\]
   does not hold, where \(\alpha\) is the assignment \(xy_r \mapsto ab\).

**Lemma 1** Let \(\Pi\) be a program with aggregate atoms of form (1) under the restrictions in Definition 2, and \(A\) a finite \(\tau(\Pi)\)-structure. Then, \(A\) is a stable model of \(\Pi\) iff \(A \models \hat{\Pi}\) and every \(S \subseteq [P_{\text{int}}(\Pi)]^A\) is externally supported.

**Proof:** (sketch) “\(\Rightarrow\)” Since \(A \models SM(\Pi)\), we have \(A \models \hat{\Pi}\). Now assume there exists a set \(S \subseteq [P_{\text{int}}(\Pi)]^A\) which is not externally supported. Let \(S_1\) denote the set of predicate symbols \(\{ U_i \mid 1 \leq i \leq n \}\). We construct a structure \(\mathcal{U}\) of \(\tau(\Pi)\) under \(S_1\) as follows:

- \(\mathcal{U} \models c^A\) for each constant \(c \in \tau(\Pi)\);
- \(\mathcal{U} \models P^A\) for each predicate \(P \in \tau(\Pi)\);
- \(U_i^A \models P_i^A \setminus \{ a \mid P_i(a) \in S \}\) for \(1 \leq i \leq n\).

Clearly, \(\mathcal{U} \models U \nless P_{\text{int}}(\Pi)\). Since \(A \models SM(\Pi)\), there exists a rule \(r\) such that \(\mathcal{U} \models B(\delta)[\alpha], \text{Pos}(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset, N\),

\[
\text{Op}(c_v[1] : A) = B(\delta)[\alpha], \text{Pos}(\delta)[\alpha] \cap S = \emptyset, N,\]

where \(\alpha\) is the assignment \(xy_r \mapsto ab\). Now, it is important to mention that although our work only considers a certain subclass of aggregates, it is indeed powerful as it covers the types of aggregates used in most applications. As far as we have checked, almost all aggregates used in benchmark normal programs (Calimeri et al. 2011) belong to our class. According to Theorem 2, those programs can all be captured by their ordered completions.

In addition, our proof technique for Theorem 2, e.g., the concept of external support, is significantly different from the original proof for ordered completion in (Asuncion et al. 2012). We believe this will shed new insights in first-order ASP (with aggregates) as well.
Ordered Completion in FOL

Theorem 2 shows that normal answer set programs with a large variety of aggregates can be captured by their ordered completions. However, the ordered completion defined in Definition 3 is not exactly in first-order logic as it contains aggregate atoms. A natural question arises whether this can be further translated in classical first-order logic. We answer it positively in this section. That is, we show that programs with aggregates functions restricted in Definition 2 can be expressed in classical first-order logic with theory on numbers.

In fact, what we need to do is to translate an aggregate atom of form (1) into classical first-order logic. For the aggregate functions MIN and MAX, this is relatively simple.

**Definition 5** Let \( \delta \) be an aggregate atom of the form (1) where OP \( \in \) \{MIN, MAX\}, and \( Lt(\delta) \) be as in \( \text{Bd}(\delta) \) but where \( Lt(\delta) \) can mention the comparison atoms. Then by \( \delta^{FO} \), we denote the following formula\(^8\)

\[
\exists \text{vw} \, Lt(\delta) \land \Phi_2 \land \\
\forall v_1 \{ \forall v_2 [\exists \text{vw} \, Lt(\delta)[v_1/v_1] \land \exists \text{vw} \, Lt(\delta)[v_1/v_2] \\
\land v_1 \neq v_2 \rightarrow (v_1 \lor v_2) \rightarrow v_1 \leq N \},
\]

where \( \circ \) is \(< \) if \( \text{OP} = \text{MIN} \) and \( > \) if \( \text{OP} = \text{MAX} \), and \( \Phi_2 = \forall \text{vw} \, (\text{Lt}(\delta) \rightarrow (v_1 \leq 0 \lor v_1 \geq 0)) \), i.e., states that the (satisfying) \( v^i \) are integers.

However, for aggregate atoms where \( \text{OP} \in \{\text{CARD, SUM, PROD}\} \), it is more complicated. Here, for space reasons, we only present the translation for aggregate atoms of the form (1) where \( \text{OP} = \text{SUM} \) and \( \leq = \leq \). The others can be done in a similar manner.

**Definition 6** Let \( \delta \) be the aggregate atom

\[
\text{SUM}(v : \exists \text{vw} \, Lt(\delta)) \leq N.
\]

By \( \delta^{FO} \), we denote the following formula\(^9\)

\[
\exists \text{vw} \, Lt(\delta) \rightarrow \exists v_1 w_1 n_1 \ldots v_N w_N n_N \\
\{ \bigwedge_{1 \leq i \leq N} Lt(\delta)[\text{vw}/v_1 w_1] \land \bigwedge_{1 \leq i \leq N} (n_i = v_1[1] \land n_i = 0) \\
\land \bigwedge_{1 \leq i,j \leq N} (v_i w_1 = v_j w_j \rightarrow (n_i = 0 \lor n_j = 0)) \\
\land \bigvee_{1 \leq i \leq N} (\text{vw} = v_1 w_1 \land v[1] = n_i) \\
\land \bigvee_{1 \leq i \leq N} (\text{vw} \rightarrow v[1] = 0) \land (n_1 + \cdots + n_N \leq N) \}
\]

Although Definition 6 seems a little complicated, the underlying idea is quite simple. We use \( n_i \) to simulate \( v_i[1] \) but only counting non-duplicated occurrences of \( v_i \). \( \exists \text{vw} \, Lt(\delta) \) in formula (19) is to cover the case where \( \text{SUM} \) is defined on \( \emptyset \). The \( \exists v_1 w_1 n_1 \ldots v_N w_N n_N \) part in formula (19) is for our counting. Formula (22) ensures that every satisfiable instance of \( \text{vw} \) is considered in our counting. Formula (20) forces \( n_i \) to be either \( v_1[1] \) (count this number) or 0 (not count it). Formula (21) means that for duplicated satisfiable instance \( \text{vw} \), we only count at most once. Finally, formula (23) ensures that the (satisfying) \( v[1] \)'s are in \( \mathbb{Z}^+ \) (i.e., must satisfy \( v[1] \geq 0 \)) and that the counting result is less than or equal to \( N \).

It should be noted that the \( \text{CARD} \) aggregates are already well known to be translatable into FOL via the \( \exists \neg \mathcal{F}(x) \) formulas (e.g., see (Lee, Lifschitz, and Palla 2008)).

**Theorem 3** For a program \( \Pi \) with aggregate atoms of form (1) under the restrictions in Definition 2. Let \( \text{OC}(\Pi)^{FO} \) be the formula obtained from \( \text{OC}(\Pi) \) by replacing every aggregate atom \( \delta \) with \( \delta^{FO} \). Then, a finite structure \( A \) is a stable model of \( \Pi \) if \( \text{A} \) can be expanded into a model of \( \text{OC}(\Pi)^{FO} \).

Theorem 3 enables us to use a first-order theorem prover or a SMT solver to compute the stable models of a program (with aggregates). This is our main motivation.

**Related Work and Discussions**

Aggregates are extensively studied in the literature (Bartholomew, Lee, and Meng 2011; Dao-Tran et al. 2009; Eiter et al. 2005; Faber, Pfeifer, and Leone 2011; Ferraris 2011; Son and Pontelli 2007). Although the syntactic form of aggregates is usually presented in a first-order language, its semantics is normally defined propositionally via grounding. There are several major approaches: Ferraris’ semantics (Ferraris 2011), the FLP semantics (Faber, Pfeifer, and Leone 2011) (later extended for arbitrary formulas by Truszczynski (2010)), and the SP semantics (Son and Pontelli 2007).

The Ferraris’ semantics and the FLP semantics (and its extension to arbitrary formulas by Truszczynski) are extended into first-order case (Ferraris, Lee, and Lifschitz 2011; Bartholomew, Lee, and Meng 2011). Our work captures the first one, also called the stable model semantics, in first-order logic. Certainly, it is interesting to consider whether the FLP semantics can be captured in first-order logic as well. We leave this to our future work. Nevertheless, it is worth mentioning that if the aggregate atoms only occurs in the positive bodies of rules and the bodies of these aggregates contain no negative atoms (this is actually the case in most benchmark programs), these semantics coincide.

Bartholomew et al. (2011) studied the aggregate semantics in first-order case via translating into second-order logic. This work is slightly different from ours. Syntactically, the former considers aggregate atoms as arbitrary formula while we only consider a special form, i.e., form (1). Semantically, the former combines the theory of first-order atoms and aggregate atoms into a unified one, while the latter defines them separately in the sense that the theory of aggregates is regarded as a background theory. The main reason is
for simplicity. It can be shown that they essentially coincide when restricted into the specific aggregate forms.

Our main results can also be extended for programs with other useful building blocks in ASP, e.g., choice rules and constraints, in the same way as in (Asuncion et al. 2012). However, for clarity and simplicity, we omit this in the paper.

Conclusion
In this paper, we showed that normal answer set programs with a large variety of aggregates (covering the aggregates used in most benchmark programs) can be captured by their ordered completions (see Theorem 2), which can be further translated into classical first-order logic (see Theorem 3). This work enables us to implement a new direction of ASP solver by firstly translating a program to its ordered completion, and then working on finding a model of this first-order sentence using, e.g., SAT or SMT solvers. We leave this promising task to our future work.

Acknowledgement
This publication was made possible by the support of an NPRP grant (NPRP 09-079-1-013) from the Qatar National Research Fund (QNRF). The statements made herein are solely the responsibility of the authors.

References


