On the Stable Model Semantics of First-Order Formulas with Aggregates

Paolo Ferraris[1] and **Vladimir Lifschitz**[2]

[1]Google, Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
[2]Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, USA

Abstract

The original definition of a stable model has been generalized to logic programs with aggregates. On the other hand, it was extended to first-order formulas using a syntactic transformation SM, similar to circumscription. In a recent paper, Lee and Meng combined these two ideas in a single framework by defining the operator SM for generalized formulas that may include, besides the usual syntactic features of first-order logic, symbols used in DLV-style aggregate expressions. In this note, we make the syntax proposed by Lee and Meng more uniform by allowing aggregate symbols to be used at any stage of the recursive process of building a formula from atoms, along with propositional connectives and quantifiers. We also generalize several useful properties of SM to formulas with aggregates, and investigate the relationship between the Lee-Meng semantics of aggregates and the semantics of counting adopted in the language RASPL-1.

Introduction

The stable model semantics, defined originally in (Gelfond and Lifschitz 1988), has been generalized in several ways. Stable models were defined, in particular, for logic programs with aggregates of various kinds (Niemelä *et al.* 1999; Faber *et al.* 2004; Ferraris 2005). This is an important extension because aggregates are widely used in the practice of answer set programming.

On the other hand, the concept of a stable model was extended to first-order formulas using a syntactic transformation SM that is similar to circumscription (Ferraris *et al.* 2007; 2010). Under this approach, logic programs are identified with first-order formulas of special syntactic forms. This version of the stable model semantics is of interest because it allows us to reason about logic programs using the well-known and well-developed apparatus of predicate logic; it is similar in this sense to Clark's completion semantics (Clark 1978).

More recently, Lee and Meng [2009] showed how to combine these two ideas in a single framework. This is achieved by defining the operator SM for generalized formulas that may include, besides the usual syntactic features of first-order logic, symbols used in DLV-style aggregate expressions.

In this note, we extend this line of work in several directions. First, we make the definition of an aggregate formula syntactically uniform by allowing aggregate symbols to be used at any stage of the recursive process of building a formula from atoms, along with propositional connectives and quantifiers. (In (Lee and Meng 2009) quantifiers cannot occur in the scope of aggregates, and aggregates cannot be nested.) Aggregates become now a special case of generalized quantifiers—a syntactic construct that plays an important role in applications of logic to computational semantics (Peters and Westerståhl 2006).

Second, we generalize the properties of SM established in (Ferraris *et al.* 2010, Section 6) to aggregate formulas. These properties show how a process similar to program completion can be sometimes used to simplify the result of applying the operator SM.

We also investigate the relationship between the Lee-Meng semantics of aggregates and the semantics of counting adopted in the answer set programming language RASPL-1 (Lee *et al.* 2008).

Formulas with Aggregates

Aggregate Functions

A *multiset* is usually defined as a set along with a function assigning a positive integer, called the *multiplicity*, to each of its elements. In this note, we understand multisets in a more general way: the multiplicity of each element is assumed to be either a positive integer or $+\infty$. By a *number* we understand an element of some fixed set **Num**. In the examples, **Num** is assumed to be $\mathbf{Z} \cup \{+\infty, -\infty\}$, where $\mathbf{Z}$ is the set of integers.

An *aggregate function* is a partial function from the class of multisets to **Num**.

Example 1 The aggregate function *#count* maps any multiset α to its cardinality if α is finite, and to $+\infty$ otherwise. This function is defined on all multisets.

Example 2 The aggregate function *#sum* maps α

- to the sum of all non-zero integers from α if there are finitely many of them,
- to $+\infty$ if α contains infinitely many positive integers but only finitely many negative integers,

- to $-\infty$ if α contains infinitely many negative integers but only finitely many positive integers.

If α contains both infinitely many positive integers and infinitely many negative integers then $\#sum$ is undefined.

Example 3 The aggregate function $\#max$ maps α to the least upper bound of the integers from α. This function is defined on all multisets.

Specifying a Language with Aggregates

Recall that a first-order language is characterized by its signature—a collection of function and predicate constants with an arity assigned to each of them; function constants of arity 0 are called object constants (see, for instance, (Lifschitz *et al.* 2008), Section 1.2.2). To specify a *first-order language with aggregates*, we should

- choose a signature σ that contains, among its object constants, symbols for all numbers,
- specify which of the binary predicate constants of σ are considered *comparison operators*, and specify a binary relation on **Num** for each of them;
- choose a collection of symbols (not from σ) representing some aggregate functions.

To simplify notation, we will not distinguish here between symbols for numbers, comparison operators, and symbols for aggregate functions on the one hand, and the objects that these symbols represent on the other.

Example 4 Take an arbitrary signature σ containing, among other symbols, the elements of $\mathbf{Z} \cup \{+\infty, -\infty\}$ as object constants and $\leq$, $\geq$ as binary predicate constants. The comparison operators of the language are $\leq$ and $\geq$, and its aggregate functions are $\#count$, $\#sum$, and $\#max$.

Formulas

In a language with aggregates, terms and atomic formulas are defined in the same way as in first-order logic. The usual recursive definition of a formula turns into a joint definition of formulas and *aggregate expressions*, and the following two clauses are added to it:

- if OP is an aggregate function, $\mathbf{x}^1, \dots, \mathbf{x}^n$ are nonempty tuples of pairwise distinct variables ($n \geq 1$), and $F_1, \dots, F_n$ are formulas, then

$$\text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n \rangle$$

  is an aggregate expression;[1]
- if E is an aggregate expression and t a term, then $E = t$ and $E \succeq t$ for any comparison operator $\succeq$ are formulas.

Formulas of the forms $E = t$ and $E \succeq t$ will be called *aggregate formulas*.

[1] This syntax is more general than in (Lee and Meng 2009), where n is assumed to be 1.

In the definition of free and bound variables, the construct $\mathbf{x}.F$ is viewed as a quantifier binding the members of $\mathbf{x}$.

Example 5 In the language from Example 4, if p and q are unary predicate constants then

$$\#count\langle x.p(x) \vee q(x) \rangle \leq 3 \tag{1}$$

and

$$\#count\langle x.p(x), x.q(x) \rangle \leq 3 \tag{2}$$

are sentences (formulas without free variables). Formula (1) expresses, intuitively, that the number of objects that satisfy at least one of the conditions p, q is at most 3. The meaning of (2) is similar, except each object satisfying both p and q is counted twice.

Example 6 In the language from Example 4, if r is a binary predicate constant then

$$\#sum\langle xy.r(x,y) \rangle = 1 \tag{3}$$

is a sentence. It expresses that the sum of the values of x over all pairs x, y satisfying r in which x is an integer equals 1. This example illustrates the special role of the first component of the tuple $\mathbf{x}$ in the semantics of expressions $\mathbf{x}.F$ defined below. In this respect, the Lee-Meng semantics follows the approach adopted in the input language of the answer set solver DLV (see (Faber *et al.* 2010), Section 2.2).

Semantics

An *interpretation* of a first-order language with aggregates is an interpretation of its signature (in the sense of first-order logic) such that each number and each comparison operator is interpreted as itself. It follows that the universe of any interpretation is a superset of **Num**, and that the extent of every comparison operator in any interpretation is a subset of $\mathbf{Num} \times \mathbf{Num}$.

Recall that the semantics of first-order logic is defined by specifying, for any interpretation I of the underlying signature,

- the value t^I of every term t that contains no variables but may contain the names ξ^* of the elements of the universe $|I|$ of I (added to the signature as object constants),
- the truth value F^I of every sentence F (that may contain names ξ^*)

(see, for instance, (Lifschitz *et al.* 2008), Section 1.2.2 for details). In the presence of aggregates, the definition of the value of a term remains the same. The recursive definition of the truth value of a formula is extended by a clause for aggregate formulas, and simultaneously we define

- the subset $(\mathbf{x}.F)^I$ of the universe $|I|$ for every nonempty tuple $\mathbf{x}$ of pairwise distinct variables and for any formula F such that all free variables of F belong to $\mathbf{x}$.

For any set X of n-tuples ($n \geq 1$), let $msp(X)$ ("the multiset projection of X") be the multiset consisting of all ξ_1 such that $(\xi_1, \xi_2, \ldots, \xi_n) \in X$ for at least one $(n-1)$-tuple $(\xi_2, \ldots, \xi_n)$, with the multiplicity equal to the number of such $(n-1)$-tuples (and to $+\infty$ if there are infinitely many of them). Using this notation, we define:

- $(x_1 \cdots x_n.F(x_1, \ldots, x_n))^I$ is

  $msp(\{(\xi_1, \ldots, \xi_n) \in |I|^n \,:\, F(\xi_1^*, \ldots, \xi_n^*)^I = \text{TRUE}\})$;

- $(\text{OP}\langle \mathbf{x^1}.F_1, \ldots, \mathbf{x^n}.F_n\rangle \succeq t)^I$ equals TRUE if the join[2] α of the multisets $(\mathbf{x^1}.F_1)^I, \ldots, (\mathbf{x^n}.F_n)^I$ belongs to the domain of OP and satisfies the condition $\text{OP}(\alpha) \succeq t^I$.

Example 7 Consider the language from Example 4 and assume that the underlying signature consists of the numbers and the comparison operators, the object constants a, b, c, the unary predicate constants p and q, and the binary predicate constant r. The interpretation I is defined as follows:

- its universe is $\mathbf{Z} \cup \{+\infty, -\infty, a, b, c\}$;
- every object constant is interpreted as itself;
- the extent of p is $\{a, b\}$;
- the extent of q is $\{b, c\}$;
- the extent of r is $\{\langle -1, a\rangle, \langle 1, b\rangle, \langle 1, c\rangle\}$.

Then $(x.p(x) \vee q(x))^I = [a, b, c]$, so that I satisfies (1). On the other hand, $(x.p(x))^I = [a, b]$ and $(x.q(x))^I = [b, c]$; the join of these multisets is $[a, b, b, c]$. Since the cardinality of the join is 4, I does not satisfy (2). Furthermore,

$$(xy.r(x, y))^I = msp(\{\langle -1, a\rangle, \langle 1, b\rangle, \langle 1, c\rangle\}) = [-1, 1, 1],$$

so that I satisfies (3).

If **Num** is empty and there are no comparison operators and no symbols for aggregates then the syntax and semantics of languages with aggregates turn into the usual syntax and semantics of first-order logic.

Relation to Generalized Quantifiers

First-order logic with aggregates can be viewed as a special case of first-order logic with generalized quantifiers in the sense of (Westerståhl 2008), Section 5 (that is to say, with Lindström quantifiers (Lindström 1966) without the isomorphism closure condition).

Without going into details, we can say that a Lindström quantifier Q associates with each universe U a relation Q_U between relations on U. For instance, take Q_U to be

$$\{\langle A, B\rangle \,:\, A \subseteq B \subseteq U\}.$$

This is a binary relation between unary relations. For this Q, the formula

$$Q[x][x](F(x), G(x)) \tag{4}$$

has the same meaning as $\forall x(F(x) \rightarrow G(x))$ ("all F's are G's"). In (4), $[x]$ repeated twice indicates that the quantifier binds x in each of the two formulas $F(x)$, $G(x)$.

The aggregate formula $\text{OP}\langle x.F(x)\rangle \succeq t$ can be written as

$$Q[x][y](F(x), y = t),$$

where y is a new variable, and Q is the Lindström quantifier defined by

$$Q_U = \{\langle A, \{n\}\rangle \,:\, A \subseteq U,\ n \in \mathbf{Num},\ \text{OP}(A) \succeq n\}.$$

Entailment and Equivalent Transformations

The definitions of entailment, logical validity, and equivalence for formulas with aggregates are the same as in first-order logic, with interpretations and satisfaction understood as defined above. Since adding aggregates does not affect the semantics of equality, propositional connectives, and quantifiers, the inference rules of first-order logic remain sound in the presence of aggregates, and standard equivalent transformations can be used to verify the equivalence of formulas with aggregates.

There are cases, however, when these inference rules and equivalent transformations are not sufficient. First-order logic does not tell us, for instance, that for any aggregate expression E, the formula

$$E \leq 1 \rightarrow E \leq 2$$

is logically valid, and that the formula

$$\#count\langle x.x = a \vee x = b\rangle = 1$$

is equivalent to $a = b$.

Second-Order Formulas with Aggregates

Predicate variables can be added to a language with aggregates in the usual way (as described, for instance, in (Lifschitz *et al.* 2008, Section 1.2.3)). Syntactically, n-ary predicate variables are used to form atomic formulas in the same way as n-ary predicate constants. Semantically, these variables range over arbitrary truth-valued functions on $|I|^n$.

Operator SM

Background

Recall that according to the definition of SM in (Ferraris *et al.* 2010), for any first-order formula F and any tuple $\mathbf{p}$ of pairwise distinct predicate constants $p_1, \ldots, p_n$ ("intensional predicates"), $\text{SM}_{\mathbf{p}}[F]$ stands for the second-order formula

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})), \tag{5}$$

where $\mathbf{u}$ is a list of distinct predicate variables $u_1, \ldots, u_n$, and $F^*(\mathbf{u})$ is defined recursively:

[2] Recall that by the definition of the join of multisets, the multiplicity of each element of the join is the sum of its multiplicities in the summands.

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any tuple $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic F that does not contain members of $\mathbf{p}$;
- $(F \wedge G)^* = F^* \wedge G^*$;
- $(F \vee G)^* = F^* \vee G^*$;
- $(F \to G)^* = (F^* \to G^*) \wedge (F \to G)$;
- $(\forall x F)^* = \forall x F^*$;
- $(\exists x F)^* = \exists x F^*$.

(There is no clause for negation here because we treat $\neg F$ as shorthand for $F \to \bot$.) A model of a sentence F is *stable* (relative to $\mathbf{p}$) if it satisfies $\mathrm{SM}_{\mathbf{p}}[F]$. The subscript $\mathbf{p}$ will be sometimes omitted.

The clauses for propositional connectives in this definition can be stated in a more uniform way: they can be equivalently replaced by

$$(F \odot G)^* = (F^* \odot G^*) \wedge (F \odot G) \qquad (\odot \in \{\wedge, \vee, \to\}).$$

The possibility of dropping the second conjunctive term in this version when $\odot$ is $\wedge$ or $\vee$ is related to the fact that the truth-valued functions corresponding to these two connectives are monotone. Similarly, the clauses for quantifiers can be equivalently replaced by

$$(QxF)^* = QxF^* \wedge QxF \qquad (Q \in \{\forall, \exists\}).$$

Extending the Definition of SM to Aggregates

To extend the definition of the operator SM to formulas with aggregates, Lee and Meng extend the recursive definition of F^* by adding a clause that treats aggregates in the same way as propositional connectives and quantifiers in the uniform version:

$$\begin{array}{l}(\mathrm{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t)^* = \\ \quad (\mathrm{OP}\langle \mathbf{x}^1.F_1^*, \dots, \mathbf{x}^n.F_n^*\rangle \succeq t) \\ \quad \wedge (\mathrm{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t)\end{array}$$

where $\succeq$ is equality or a comparison operator.

Example 8 If F is the formula

$$p(2) \wedge p(3) \wedge (\#sum\langle x.p(x)\rangle \leq 10 \to q) \qquad (6)$$

and $\mathbf{p}$ is pq then $F^*(u, v)$ is

$$\begin{array}{l}u(2) \wedge u(3) \\ \wedge ((\#sum\langle x.u(x)\rangle \leq 10 \wedge \#sum\langle x.p(x)\rangle \leq 10) \to v) \\ \wedge (\#sum\langle x.p(x)\rangle \leq 10 \to q).\end{array}$$

We will see below that the result of applying SM to (6) is equivalent to

$$\forall x(p(x) \leftrightarrow x = 2 \vee x = 3) \wedge q. \qquad (7)$$

Consequently the stable models of (6) can be characterized as the interpretations in which the extent of p is $\{2, 3\}$, and q is true.

Formula (6) can be viewed as the DLV program

```
p(2).
p(3).
q :- #sum{X : p(X)} <= 10.
```

written as a formula with aggregates. The output returned by DLV for this program

```
{p(2), p(3), q}
```

agrees with the characterization of the stable models of (6) given above. This is not surprising, in view of Propositions 6 and 7 from (Lee and Meng 2009) that relate the semantics of aggregates in DLV to the operator SM.

In view of the relationship between aggregates and generalized quantifiers discussed above, it is interesting to compare the Lee-Meng semantics of aggregates with the definition of a stable model for logic programs with generalized quantifiers proposed by Eiter *et al.* (1997, 1999). The former is more general syntactically in the sense that arbitrary propositional connectives and quantifiers are allowed. Semantically, the difference is that the former generalizes the approach of (Ferraris *et al.* 2010), and the latter is based on the concept of the reduct from (Gelfond and Lifschitz 1988).

Monotonicity

As pointed out above, the concept of a stable model is not affected when in the characterization of $(F \odot G)^*$ as

$$(F^* \odot G^*) \wedge (F \odot G) \qquad (8)$$

the second conjunctive term is dropped for the "monotone" connectives $\wedge$, $\vee$. This follows from the fact that in the presence of the condition $\mathbf{u} < \mathbf{p}$ from (5), or even the weaker condition $\mathbf{u} \leq \mathbf{p}$, the first conjunctive term of (8) implies the second conjunctive term and is consequently equivalent to the whole conjunction. To put it precisely, the implication

$$\mathbf{u} \leq \mathbf{p} \to ((F \odot G)^* \leftrightarrow (F^* \odot G^*))$$

is logically valid for any formulas F, G if $\odot$ is $\wedge$ or $\vee$.

To state a similar fact for aggregates, we need the following definition. An aggregate function OP is *monotone* with respect to a comparison operator $\succeq$ if for any multisets α, β such that $\alpha \subseteq \beta$,

- if $\mathrm{OP}(\alpha)$ is defined then so is $\mathrm{OP}(\beta)$, and
- for any $n \in \mathbf{Num}$, if $\mathrm{OP}(\alpha) \succeq n$ then $\mathrm{OP}(\beta) \succeq n$.

Anti-monotone aggregate functions are defined in a similar way, with $\alpha \subseteq \beta$ replaced by $\beta \subseteq \alpha$.

It is clear, for instance, that *#count* is monotone with respect to $\geq$ and anti-monotone with respect to $\leq$. Function *#sum* is not monotone with respect to $\geq$. Indeed, it is true that $\#sum([1]) \geq 1$, but it is not true that $\#sum([-1, 1]) \geq 1$. On the other hand, the aggregate function $\#sum^+$, which maps any multiset α to the sum of all positive integers from α (and to $+\infty$ if α contains infinitely many positive integers) has the same properties as *#count*: it is monotone with respect to $\geq$ and anti-monotone with respect to $\leq$.

Theorem 1 *(i) If* OP *is monotone with respect to* $\succeq$ *then the formula*

$$\mathbf{u} \leq \mathbf{p} \to ((\text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t)^* \leftrightarrow \text{OP}\langle \mathbf{x}^1.F_1^*, \dots, \mathbf{x}^n.F_n^*\rangle \succeq t)$$

is logically valid. (ii) If OP *is anti-monotone with respect to* $\succeq$ *then the formula*

$$\mathbf{u} \leq \mathbf{p} \to ((\text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t)^* \leftrightarrow \text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t)$$

is logically valid.

Thus in the conjunction

$$(\text{OP}\langle \mathbf{x}^1.F_1^*, \dots, \mathbf{x}^n.F_n^*\rangle \succeq t) \wedge (\text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t),$$

which defines $(\text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t)^*$, the second conjunctive term can be dropped if OP is monotone, and the first term can be dropped if OP is anti-monotone. Part (i) of the theorem shows that monotone aggregates are similar, from the point of view of the stable model semantics, to conjunction and disjunction. In view of Lemma 6 from (Ferraris *et al.* 2010), which asserts the logical validity of

$$\mathbf{u} \leq \mathbf{p} \to (\neg F)^* \leftrightarrow \neg F),$$

we can say that part (ii) demonstrates a similarity between anti-monotone aggregates and negation.

Equivalence with Respect to the Stable Model Semantics

Two sentences that are equivalent to each other do not necessarily have the same stable models. It is known, on the other hand, that if the equivalence between first-order formulas F and G can be proved in intuitionistic logic with equality then $\text{SM}_{\mathbf{p}}[F]$ is equivalent to $\text{SM}_{\mathbf{p}}[G]$ (Lifschitz *et al.* 2007; Ferraris *et al.* 2010).[3] The extension of SM defined above has a similar property: if F and G are first-order formulas with aggregates such that the equivalence between them can be derived using axiom schemas and inference rules of intuitionistic logic with equality then $\text{SM}_{\mathbf{p}}[F]$ is equivalent to $\text{SM}_{\mathbf{p}}[G]$.

For instance, the equivalence between (6) and

$$\forall x((x = 2 \vee x = 3) \to p(x)) \wedge (\#sum\langle x.p(x)\rangle \leq 10 \to q) \tag{9}$$

can be justified using postulates of intuitionistic logic with equality. It follows that these two formulas have the same stable models.

Completion and Tightness

The properties of the operator SM established in (Ferraris *et al.* 2010), Section 6, show how a process similar to Clark's program completion can be sometimes used to simplify the result of applying the operator SM and, in particular, to eliminate second-order quantifiers from it. In this section we show how to extend this process to formulas with aggregates.

The following two definitions are not affected by the presence of aggregates in the language. A formula is in *Clark normal form* (relative to the list $\mathbf{p}$ of intensional predicates) if it is a conjunction of formulas of the form

$$\forall \mathbf{x}(G \to p(\mathbf{x})), \tag{10}$$

one for each intensional predicate p, where $\mathbf{x}$ is a list of distinct object variables. The *completion* of a formula F in Clark normal form, denoted by $\text{COMP}[F]$, is obtained from it by replacing each conjunctive term (10) with

$$\forall \mathbf{x}(p(\mathbf{x}) \leftrightarrow G). \tag{11}$$

For instance, formula (9) is in Clark normal form, and its completion is

$$\forall x(p(x) \leftrightarrow (x = 2 \vee x = 3)) \wedge (q \leftrightarrow \#sum\langle x.p(x)\rangle \leq 10). \tag{12}$$

According to Theorem 10 from (Ferraris *et al.* 2010), the implication

$$\text{SM}[F] \to \text{COMP}[F]$$

is logically valid for any formula F in Clark normal form. Furthermore, according to Theorem 11, the implication in the other direction is logically valid whenever F is "tight." To extend the concept of a tight formula to formulas with aggregates, we need to introduce a few definitions.

In first-order logic, occurrences of predicate constants (or of any other expressions) in a formula are classified into positive and negative as follows: an occurrence is positive if the number of implications containing it in the antecedent is even, and negative otherwise. (Recall that $\neg F$ is shorthand for $F \to \bot$.) For first-order formulas with aggregates, we will distinguish between positive, negative, and "mixed" occurrences. The need for a third category emerges also in other languages—for instance, in propositional logic when $\leftrightarrow$ is treated as a primitive, rather than an abbreviation. In that setting we would say that the occurrences of p and q in $p \leftrightarrow q$ are neither positive nor negative.

Let F be a formula with aggregates. Consider an occurrence of a predicate constant in F, and consider all aggregate subformulas

$$\text{OP}\langle \mathbf{x}^1.F_1, \dots, \mathbf{x}^n.F_n\rangle \succeq t \tag{13}$$

of F containing that occurrence. If in at least one of these subformulas OP is neither monotone nor anti-monotone with respect to $\succeq$ then we say that the occurrence is *mixed*. If the occurrence is not mixed then consider the number k of implications containing that occurrence in the antecedent, and the number l of formulas (13) in which OP is anti-monotone with respect to $\succeq$; the occurrence is *positive* if $k + l$ is even, and *negative* otherwise. For example, the first two occurrences of p

[3] These papers show that this claim holds also for a stronger logical system—for the intermediate logic called **SQHT**$^=$.

in (6) are positive, and the third is mixed; if we replace *#sum* in this formula with *#count* then the third occurrence will become positive as well ($k = l = 1$). Counting anti-monotone aggregate functions along with the antecedents of implications is suggested by the analogy between such functions and negation discussed above.

An occurrence of a predicate constant in F is *negated* if it belongs to a subformula of F that has the form $G \to \bot$ or is an aggregate formula (13) such that OP is anti-monotone with respect to $\succeq$.

For any formula F in Clark normal form, the *predicate dependency graph* of F is the directed graph that

- has all intensional predicates as its vertices, and
- has an edge from p to q if the antecedent G of the conjunctive term (10) of F with p in the consequent has an occurrence of q that is not negative and not negated.[4]

For example, the predicate dependency graph of formula (9) has the vertices p, q, and one edge—from q to p, because the occurrence of p in

$$\#sum\langle x.p(x)\rangle \leq 10$$

is not negative (it is mixed) and not negated. If we replace *#sum* in (9) with *#count* then the edge will disappear, because the occurrence of p in

$$\#count\langle x.p(x)\rangle \leq 10$$

is negative (and also negated). If we replace *#sum* with *#count* and also $\leq$ with $\geq$ then the edge from q to p will come back, because the occurrence of p in

$$\#count\langle x.p(x)\rangle \geq 10$$

is not negative (it is positive) and not negated.

We say that a formula in Clark normal form is *tight* if its predicate dependency graph is acyclic. For example, formula (9) is tight, and so are its two modifications described in the previous paragraph. If we replace the first conjunctive term of (9) with

$$\forall x((x = 2 \vee (x = 3 \wedge q)) \to p(x))$$

then the dependency graph will get a second edge, from p to q, and the formula will become non-tight.

Theorem 2 *If a formula F in Clark normal form is tight then SM[F] is equivalent to COMP[F].*

This theorem shows, for instance, that the result of applying the operator SM to formula (9) is equivalent to the completion (12) of this formula. Since (9) has the same stable models as (6), and (12) can be equivalently rewritten as (7), we have justified the claim regarding the stable models of (6) made in Example 8.

[4]The definition of the predicate dependency graph would not be affected if we replaced $k + l$ with k in the definition of a positive occurrence (Joohyung Lee, personal communication, February 5, 2010).

Relation to RASPL-1

The theorem stated below shows that in application to the aggregate *#count* the Lee-Meng semantics is equivalent to the approach adopted in RASPL-1 (Lee *et al.* 2008). In the statement of the theorem, **Num** is assumed to be $\mathbf{Z} \cup \{+\infty, -\infty\}$, or it can be $\{0, 1, \ldots, +\infty\}$; the set of aggregate functions of the language is assumed to include *#count*, and the set of comparison operators is assumed to include $\geq$.

Theorem 3 *Let F be a formula containing a subformula of the form*

$$\#count\langle \mathbf{x}.G(\mathbf{x})\rangle \geq n,$$

where n is a positive integer. If F' is the result of replacing this subformula with

$$\exists \mathbf{x}^1 \cdots \mathbf{x}^n \left[\bigwedge_{1 \leq i \leq n} G(\mathbf{x}^i) \wedge \bigwedge_{1 \leq i < j \leq n} \neg(\mathbf{x}^i = \mathbf{x}^j) \right],$$

where $\mathbf{x}^1, \ldots, \mathbf{x}^n$ are disjoint lists of distinct variables of the same length as $\mathbf{x}$, then SM[F'] is equivalent to SM[F].

Conclusion

The Lee-Meng semantics of aggregates is perhaps the most attractive of the existing approaches to incorporating aggregates in answer set programming languages. Like the method of (Faber *et al.* 2004), it exhibits good formal properties even in application to aggregates that are neither monotone nor anti-monotone. On the other hand, it inherits the conceptual simplicity of the framework developed by Ferraris *et al.* (2007, 2010), which does not require grounding as an intermediate step.

The main properties of the SM operator established in (Ferraris *et al.* 2010) need to be extended now to formulas with aggregates. We have made here a step in this direction.

Acknowledgements

We are grateful to Joohyung Lee, Yuliya Lierler, Fangkai Yang, and the anonymous referees for comments on a draft of this note. The second author was partially supported by the National Science Foundation under grant IIS-0712113.

References

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

Thomas Eiter, Georg Gottlob, and Helmuth Veith. Modular logic programming and generalized quantifiers. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 290–309. Springer, 1997.

Thomas Eiter, Georg Gottlob, and Helmuth Veith. Generalized quantifiers in logic programs. In *Generalized Quantifiers and Computation, 9th European Summer School in Logic, Language, and Information (Lecture Notes in Computer Science 1754)*, pages 72–98. Springer, 1999.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of European Conference on Logics in Artificial Intelligence (JELIA)*, 2004.

Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 2010. To appear.

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.

Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. Stable models and circumscription.[5] *Artificial Intelligence*, 2010. To appear.

Paolo Ferraris. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 119–131, 2005.

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 Yunsong Meng. On reductive semantics of aggregates in answer set programming. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 182–195, 2009.

Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. A reductive semantics for counting and choice in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 472–479, 2008.

Vladimir Lifschitz, David Pearce, and Agustin Valverde. A characterization of strong equivalence for logic programs with variables. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 2007.

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*, pages 3–88. Elsevier, 2008.

Per Lindström. First-order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics for weight constraint rules. In *Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 317–331, 1999.

Stanley Peters and Dag Westerståhl. *Quantifiers in language and logic*. Oxford University Press, 2006.

Dag Westerståhl. Generalized quantifiers. In *The Stanford Encyclopedia of Philosophy (Winter 2008 Edition)*. 2008. URL = <http://plato.stanford.edu/archives/win2008/entries/generalized-quantifiers/>.

[5] `http://peace.eas.asu.edu/joolee/papers/smcirc.pdf`