On Reductive Semantics of Aggregates in Answer Set Programming

Joohyung Lee and Yunsong Meng

Computer Science and Engineering
Arizona State University, Tempe, AZ, USA
{joolee, Yunsong.Meng}@asu.edu

Abstract. Several proposals of the semantics of aggregates are based on different extensions of the stable model semantics, which makes it difficult to compare them. In this note, building upon a reductive approach to designing aggregates, we provide reformulations of some existing semantics in terms of propositional formulas, which help us compare the semantics and understand their properties in terms of their propositional formula representations. We also present a generalization of semantics of aggregates without involving grounding, and define loop formulas for programs with aggregates guided by the reductive approach.

1 Introduction

Defining a reasonable semantics of aggregates under the stable model semantics has turned out to be a non-trivial task. An obvious "reductive" approach to understand an aggregate as shorthand for a nested expression [1] in the form of disjunctions over conjunctions leads to unintuitive results. For instance, one would expect $\{p(0), p(1)\}$ to be the only answer set of the following program.

$$p(1) \qquad\qquad p(0) \leftarrow \text{SUM}\langle\{x : p(x)\}\rangle = 1.$$

Assuming that the domain is $\{0, 1\}$, one may try to identify $\text{SUM}\langle\{x : p(x)\}\rangle = 1$ with the disjunction over two "solutions,"—one in which only $p(1)$ is true, and the other in which both $p(0)$ and $p(1)$ are true. However, the resulting program

$$p(1) \qquad\qquad p(0) \leftarrow (p(0), p(1))\ ;\ (not\ p(0), p(1))$$

has no answer sets. [1]

The difficulty led to several interesting extensions of the stable model semantics to account for aggregates, such as an extended definition of the reduct [2], an extension of T_P operator with "conditional satisfaction" [3], and an extension of the standard approximating operator Φ_P to Φ_P^{aggr} [4]. On the other hand, a few reasonable "reductive" semantics of aggregates were also developed. In [4] and [5], the authors defined translations of aggregates into nested expressions,

[1] Dropping negative literals in forming each conjunct does not work either. For instance, consider $p(1) \leftarrow \text{SUM}\langle\{x : p(x)\}\rangle \neq 1$, which intuitively has no answer sets, but its translation, assuming that the domain is $\{1\}$, results in $p(1) \leftarrow \top$.

which are somewhat complex than the naive approach above. In [6], instead of translating into nested expressions, Ferraris proposed to identify an aggregate with conjunctions of implications under his extension of the answer set semantics for arbitrary propositional formulas. The extended semantics is essentially a reformulation of the equilibrium logic [7], and was generalized to arbitrary first-order formulas in [8].

While most semantics agree on monotone and anti-monotone aggregates, they have subtle differences in understanding arbitrary aggregates. For example, the following program Π_1

$$\begin{aligned} p(2) &\leftarrow \mathit{not}\ \text{SUM}\langle\{x : p(x)\}\rangle < 2 \\ p(-1) &\leftarrow \text{SUM}\langle\{x : p(x)\}\rangle \geq 0 \\ p(1) &\leftarrow p(-1)\ . \end{aligned}$$

has no answer sets according to [3, 4], one answer set $\{p(-1), p(1)\}$ according to [2], and two answer sets $\{p(-1), p(1)\}$ and $\{p(-1), p(1), p(2)\}$ according to [6].

In this paper we make further developments on the reductive approach. We note that the semantics by Pelov, Denecker and Bruynooghe [4] and the semantics by Ferraris [6] are closely related to each other in terms of propositional formula representations of aggregates, yielding a few interesting alternative characterizations. Furthermore we show that the semantics by Faber, Leone and Pfeifer [2] can also be reformulated in terms of propositional formulas. Such uniform characterization helps us compare the semantics and understand their properties by turning to their propositional formula representations. We define loop formulas for programs with aggregates guided by the reductive approach; such loop formulas contain aggregates, and when these aggregates are turned into corresponding propositional formulas, the resulting formulas are the same as the loop formulas as defined in [9] for the propositional theory corresponding to the program with aggregates.

2 Background

2.1 Answer sets of First-Order Formulas

We review the definition of an answer set from [8]. Let $\mathbf{p}$ be a list of predicate constants $p_1, \dots, p_n$, and let $\mathbf{u}$ be a list of predicate variables $u_1, \dots, u_n$. By $\mathbf{u} \leq \mathbf{p}$ we denote the conjunction of the formulas $\forall\mathbf{x}(u_i(\mathbf{x}) \rightarrow p_i(\mathbf{x}))$ for all $i = 1, \dots n$ where $\mathbf{x}$ is a list of distinct object variables of the same length as the arity of p_i, and by $\mathbf{u} < \mathbf{p}$ we denote $(\mathbf{u} \leq \mathbf{p}) \wedge \neg(\mathbf{p} \leq \mathbf{u})$.

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

$$F \wedge \neg\exists\mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})), \tag{1}$$

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

- $p_i(t_1, \dots, t_m)^* = u_i(t_1, \dots, t_m)$;
- $(t_1 = t_2)^* = (t_1 = t_2)$;
- $\bot^* = \bot$;

- $(F \odot G)^* = (F^* \odot G^*)$, where $\odot \in \{\wedge, \vee\}$;
- $(F \rightarrow G)^* = (F^* \rightarrow G^*) \wedge (F \rightarrow G)$;
- $(QxF)^* = QxF^*$, where $Q \in \{\forall, \exists\}$.

(There is no clause for negation here, $\neg F$ is treated as shorthand for $F \rightarrow \bot$.)

Let $\sigma(F)$ be the signature consisting of the object, function and predicate constants occurring in F. According to [8], an interpretation of $\sigma(F)$ that satisfies SM[F] is called a *stable model* of F. If F contains at least one object constant, an Herbrand stable model of F is called an *answer set* of F. The answer sets of a logic program Π are defined as the answer sets of the FOL-representation of Π (i.e., the conjunction of the universal closure of implications corresponding to the rules).

Ferraris *et al.* [8] shows that this definition, if restricted to the syntax of traditional logic programs, is equivalent to the traditional definition of an answer set based on grounding and the reduct [10], and, if restricted to the syntax of arbitrary propositional formulas, is equivalent to the definition of an answer set given by Ferraris [6].

2.2 Syntax of a Program with Aggregates

An *aggregate function* is any function that maps multisets of objects into numbers, such as *count*, *sum*, *times*, *min* and *max*. For this paper we assume that all numbers are integers. The domain of an aggregate function is defined as usual. For instance, *sum*, *times*, *min* and *max* are defined for multisets of numbers; *min* and *max* do not allow the empty set in their domains.

An *aggregate expression* is of the form

$$\text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b \qquad (2)$$

where

- OP is a symbol for an *aggregate function op*;
- $\mathbf{x}$ is a nonempty list of distinct object variables;
- $F(\mathbf{x})$ is an arbitrary quantifier-free formula;
- $\succeq$ is a symbol for a binary relation over integers, such as $\leq, \geq, <, >, =, \neq$;
- b is an integer constant.

A *rule (with aggregates)* is an expression of the form

$$A_1 \,;\ldots;\, A_l \leftarrow\; E_1,\ldots,E_m, not\ E_{m+1},\ldots, not\ E_n \qquad (3)$$

($l \geq 0$; $n \geq m \geq 0$), where each A_i is an atomic formula (possibly containing equality) and each E_i is an atomic formula or an aggregate expression. A *program (with aggregates)* is a finite set of rules.

Throughout this paper, unless otherwise noted (e.g., Section 4.1), we assume that the program contains no function constants of positive arity. We do not consider symbols OP and $\succeq$ as part of the signature.

We say that an occurrence of a variable v in a rule (3) is *bound* if the occurrence is in an aggregate expression (2) such that v is in $\mathbf{x}$; otherwise it is *free*. We say that v is *free* in the rule if some occurrence of v is free in it. Given a program Π, by $\sigma(\Pi)$ we mean the signature consisting of object and predicate constants that occur in Π. By $Ground(\Pi)$ we denote the program without free variables that is obtained from Π by replacing every free occurrence of variables with every object constant from $\sigma(\Pi)$ in all possible ways.

2.3 Review: FLP Semantics

The FLP semantics [2] is based on an alternative definition of the reduct and the notion of satisfaction extended to aggregate expressions. Let Π be a program such that $\sigma(\Pi)$ contains at least one object constant.[2] We consider Herbrand interpretations of $\sigma(\Pi)$ only. Consider any aggregate expression (2) occurring in $Ground(\Pi)$ and any Herbrand interpretation I of $\sigma(\Pi)$. Let S_I be the multiset consisting of all $\mathbf{c}[1]$ (i.e., the first element of $\mathbf{c}$) in the Herbrand universe where

– $\mathbf{c}$ is a list of object constants of $\sigma(\Pi)$ whose length is the same as the length of $\mathbf{x}$, and
– I satisfies $F(\mathbf{c})$.

A set I of ground atoms of $\sigma(\Pi)$ satisfies the aggregate expression if S_I is in the domain of *op*, and $op(S_I) \succeq b$. [3]

The *FLP reduct* of Π relative to I is obtained from $Ground(\Pi)$ by removing every rule whose body is not satisfied by I. Set I is an *FLP answer set* of Π if it is minimal among the sets of atoms that satisfy the FLP reduct of Π relative to I. For example, in program Π_1 (Section 1), the FLP reduct of Π_1 relative to $\{p(-1), p(1)\}$ contains the last two rules only. Set $\{p(-1), p(1)\}$ is minimal among the sets of atoms that satisfy the reduct, and thus is an FLP answer set of Π_1. One can check that this is the only FLP answer set.

2.4 Review: Ferraris Semantics

The Ferraris semantics [6] can be extended to allow variables as follows.
Notation: Given a multiset of object constants $\{\!\{c_1, \dots, c_n\}\!\}$,

$$\text{OP}\langle\{\!\{c_1, \dots, c_n\}\!\}\rangle \succeq b$$

if

– b is an integer constant,
– multiset $\{\!\{c_1, \dots, c_n\}\!\}$ is in the domain of *op*, and
– $op(\{\!\{c_1, \dots, c_n\}\!\}) \succeq b$.

$\text{OP}\langle\{\!\{c_1, \dots, c_n\}\!\}\rangle \not\succeq b$ if it is not the case that $\text{OP}\langle\{\!\{c_1, \dots, c_n\}\!\}\rangle \succeq b$.

Let $E = \text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b$ be an aggregate expression occurring in $Ground(\Pi)$, let $\mathbf{O}_\Pi(E)$ be the set of all lists of object constants of $\sigma(\Pi)$ whose length is the same as the length of $\mathbf{x}$, and let $\overline{\mathcal{C}}_\Pi(E)$ be the set of all subsets $\mathbf{C}$ of $\mathbf{O}_\Pi(E)$ such that $\text{OP}\langle\{\!\{\mathbf{c}[1] : \mathbf{c} \in \mathbf{C}\}\!\}\rangle \not\succeq b$. For instance, in program Π_1 (Section 1), for $E_1 = \text{SUM}\langle\{x : p(x)\}\rangle < 2$, set $\mathbf{O}_{\Pi_1}(E_1)$ is $\{-1, 1, 2\}$, and $\overline{\mathcal{C}}_{\Pi_1}(E_1)$ is $\{\{2\}, \{1, 2\}, \{-1, 1, 2\}\}$. Similarly, for $E_2 = \text{SUM}\langle\{x : p(x)\}\rangle \geq 0$, set $\overline{\mathcal{C}}_{\Pi_1}(E_2)$ is $\{\{-1\}\}$.

[2] The syntax in [2] requires $F(\mathbf{x})$ to be a conjunction of atoms.
[3] By an *atom* we mean a non-equality atomic formula of the form $p(t_1, \dots, t_n)$.

By $Fer_{\Pi}(E)$ we denote

$$\bigwedge_{\mathbf{C}\in\overline{\mathcal{C}}_{\Pi}(E)} \Big(\bigwedge_{\mathbf{c}\in\mathbf{C}} F(\mathbf{c}) \rightarrow \bigvee_{\mathbf{c}\in\mathbf{O}_{\Pi}(E)\setminus\mathbf{C}} F(\mathbf{c})\Big). \tag{4}$$

For instance, $Fer_{\Pi_1}(E_1)$ is

$$(p(2) \rightarrow p(-1) \vee p(1)) \wedge (p(1) \wedge p(2) \rightarrow p(-1)) \wedge (p(-1) \wedge p(1) \wedge p(2) \rightarrow \bot)\ .$$

By $Fer(\Pi)$ we denote the propositional formula obtained from $Ground(\Pi)$ by replacing every aggregate expression E in it by $Fer_{\Pi}(E)$. The *Ferraris answer sets* of Π are defined as the answer sets of $Fer(\Pi)$ in the sense of Section 2.1. For example, the Ferraris answer sets of Π_1 are the answer sets of the following formula $Fer(\Pi_1)$:[4]

$$\begin{array}{l}(\neg\underline{[(p(2)\rightarrow p(-1)\vee p(1)) \wedge (p(1)\wedge p(2)\rightarrow p(-1)) \wedge (p(-1)\wedge p(1)\wedge p(2)\rightarrow \bot)]} \rightarrow p(2))\\ \wedge\ (\underline{[p(-1)\rightarrow p(1)\vee p(2)]} \rightarrow p(-1))\\ \wedge\ (p(-1) \rightarrow p(1))\ .\end{array} \tag{5}$$

This formula has two answer sets: $\{p(-1), p(1)\}$ and $\{p(-1), p(1), p(2)\}$.

2.5 Review: SPT-PDB Semantics

Son and Pontelli [5] presented two equivalent definitions of aggregates, one in terms of "unfolding" into nested expressions, and the other in terms of "conditional satisfaction." The latter notion was simplified by Son, Pontelli and Tu [3]. Lemma 6 from [5] shows that these definitions are equivalent to the definition by Pelov, Denecker and Bruynooghe [4], which is in terms of translation into nested expressions. Thus we group them together and review only the last one.[5]

Under the SPT-PDB semantics, an aggregate can be identified with a nested expression in the form of disjunctions over conjunctions, but unlike the naive attempt given in the introduction, it involves the notion of a "(maximal) local power set."

Given a set A of some sets, a pair $\langle B, T\rangle$ where $B, T \in A$ and $B \subseteq T$ is called a *local power set (LPS)* of A if every S such that $B \subseteq S \subseteq T$ belongs to A as well. A local power set is called *maximal* if there is no other local power set $\langle B', T'\rangle$ of A such that $B' \subseteq B$ and $T \subseteq T'$.

The SPT-PDB semantics eliminates the negation in front of an aggregate expression using an equivalent transformation. Let $Pos(\Pi)$ be a program obtained from Π by replacing *not* E_i in each rule (3) where $E_i = \text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b$ with $\text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \prec b$ ($\prec$ is the symbol for the relation complementary to $\succeq$). Clearly, $Pos(\Pi)$ contains no negation in front of aggregate expressions. For instance, the first rule of $Pos(\Pi_1)$ is

$$p(2) \leftarrow \text{SUM}\langle\{x : p(x)\}\rangle \geq 2.$$

[4] We underline the parts of a formula that correspond to aggregates.

[5] We ignore some differences in the syntax, and allow disjunctions in the head.

Let $E = \text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b$ be an aggregate expression occurring in $\mathit{Ground}(\mathit{Pos}(\Pi))$, let HU_Π be the set of all ground atoms that can be constructed from $\sigma(\Pi)$. Let $\mathcal{I}_\Pi(E)$ be the set of all Herbrand interpretations I of $\sigma(\Pi)$ such that $I \models E$ (satisfaction as defined in Section 2.3). For instance, in Example Π_1, HU_{Π_1} is $\{p(-1), p(1), p(2)\}$, and, for $\overline{E_1} = \text{SUM}\langle\{x : p(x)\}\rangle \geq 2$, $\mathcal{I}_{\Pi_1}(\overline{E_1})$ is $\{\{p(2)\}, \{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\}$, and $\mathcal{I}_{\Pi_1}(E_2)$ is

$$\{\emptyset, \{p(1)\}, \{p(2)\}, \{p(-1), p(1)\}, \{p(-1), p(2)\}, \{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\}\ .$$

The maximal local power sets of $\mathcal{I}_{\Pi_1}(\overline{E_1})$ are

$$\langle\{p(2)\}, \{p(1), p(2)\}\rangle,\ \ \langle\{p(1), p(2)\}, \{p(-1), p(1), p(2)\}\rangle,$$

and the maximal local power sets of $\mathcal{I}_{\Pi_1}(E_2)$ are

$$\langle\emptyset, \{p(1), p(2)\}\rangle,\ \ \langle\{p(1)\}, \{p(-1), p(1), p(2)\}\rangle,\ \ \langle\{p(2)\}, \{p(-1), p(1), p(2)\}\rangle.$$

For any aggregate expression E occurring in $\mathit{Ground}(\mathit{Pos}(\Pi))$, by $\mathit{SPT\text{-}PDB}_\Pi(E)$ we denote

$$\bigvee_{\langle B,T\rangle \text{ is a maximal LPS of } \mathcal{I}_\Pi(E)} \Big(\bigwedge_{A\in B} A \wedge \bigwedge_{A\in \mathit{HU}_\Pi \setminus T} \neg A\Big). \tag{6}$$

For instance, $\mathit{SPT\text{-}PDB}_{\Pi_1}(\overline{E_1})$ is $(p(2) \wedge \neg p(-1)) \vee (p(1) \wedge p(2))$.

By $\mathit{SPT\text{-}PDB}(\Pi)$ we denote the propositional formula obtained from $\mathit{Ground}(\mathit{Pos}(\Pi))$ by replacing all aggregate expressions E in it by $\mathit{SPT\text{-}PDB}_\Pi(E)$. The $\mathit{SPT\text{-}PDB}$ *answer sets* of Π are defined as the answer sets of $\mathit{SPT\text{-}PDB}(\Pi)$ in the sense of Section 2.1. For example, Π_1 has no SPT-PDB answer sets, and neither does the following formula $\mathit{SPT\text{-}PDB}(\Pi_1)$:

$$\begin{array}{l}(\underline{[(p(2) \wedge \neg p(-1)) \vee (p(1) \wedge p(2))]} \rightarrow p(2)) \\ \wedge\ (\underline{[\neg p(-1) \vee p(1) \vee p(2)]} \rightarrow p(-1)) \\ \wedge\ (p(-1) \rightarrow p(1))\ .\end{array} \tag{7}$$

3 Comparison of the Semantics of Aggregates

3.1 A Reformulation of Ferraris Semantics

The propositional formula representation of an aggregate according to the Ferraris semantics can be written in a more compact way by considering maximal local power sets as in the SPT-PDB semantics.

For an aggregate expression that contains no free variables, by $\mathit{MLPS\text{-}Fer}_\Pi(E)$ we denote

$$\bigwedge_{\langle B,T\rangle \text{ is a maximal LPS of } \overline{\mathcal{C}}_\Pi(E)} \Big(\bigwedge_{\mathbf{c}\in B} F(\mathbf{c}) \rightarrow \bigvee_{\mathbf{c}\in \mathbf{O}_\Pi(E)\setminus T} F(\mathbf{c})\Big). \tag{8}$$

One can prove that formulas (4) and (8) are strongly equivalent [11] to each other, which provides another characterization of Ferraris answer sets. We define MLPS-Ferraris answer sets of Π same as the Ferraris answer sets of Π except that we refer to (8) in place of (4). The following proposition follows from the strong equivalence between (4) and (8).

Proposition 1 *The MLPS-Ferraris answer sets of Π are precisely the Ferraris answer sets of Π.*

For example, in program Π_1, the maximal local power sets of $\overline{\mathcal{C}}_{\Pi_1}(E_1)$ are $\langle\{2\},\{1,2\}\rangle$, $\langle\{1,2\},\{-1,1,2\}\rangle$. The maximal local power set of $\overline{\mathcal{C}}_{\Pi_1}(E_2)$ is $\langle\{-1\},\{-1\}\rangle$. Formula (5) is strongly equivalent to this shorter formula

$$\begin{array}{l}(\neg\underline{[(p(2)\rightarrow p(-1))\wedge(p(1)\wedge p(2)\rightarrow\bot)]}\rightarrow p(2))\\ \wedge\ (\underline{[p(-1)\rightarrow p(1)\vee p(2)]}\rightarrow p(-1))\\ \wedge\ (p(-1)\rightarrow p(1))\ .\end{array}$$

3.2 A Reformulation of FLP Semantics

The FLP semantics can also be defined by reduction to propositional formulas. For an aggregate expression E that contains no free variables, let $\overline{\mathcal{I}}_\Pi(E)$ be the set of all Herbrand interpretations I of $\sigma(\Pi)$ such that $I\not\models E$ (as defined in Section 2.3). Clearly $\overline{\mathcal{I}}_\Pi(E)$ and $\mathcal{I}_\Pi(E)$ partition HU_Π. By $FLP_\Pi(E)$ we denote

$$\bigwedge_{I\in\overline{\mathcal{I}}_\Pi(E)}\Big(\bigwedge_{A\in I}A\rightarrow\bigvee_{A\in HU_\Pi\setminus I}A\Big)\ . \tag{9}$$

As with the SPT-PDB semantics, before turning a program to the propositional formula representation for the FLP semantics, we eliminate the negation in front of an aggregate expression using an equivalent transformation. By $FLP(\Pi)$ we denote the propositional formula obtained from $Ground(Pos(\Pi))$ by replacing all aggregate expressions E in it by $FLP_\Pi(E)$ (Recall the definition of $Pos(\Pi)$ in Section 2.5).

Proposition 2 *For any program Π, the FLP answer sets of Π (Section 2.3) are precisely the answer sets of $FLP(\Pi)$.*

For example, in program Π_1, $\overline{\mathcal{I}}_{\Pi_1}(\overline{E_1})$ is $\{\emptyset,\{p(-1)\},\{p(1)\},\{p(-1),p(1)\},\{p(-1),p(2)\}\}$ and $\overline{\mathcal{I}}_{\Pi_1}(E_2)$ is $\{\{p(-1)\}\}$, so that $FLP(\Pi_1)$ is

$$\begin{array}{l}(\underline{[(p(-1)\vee p(1)\vee p(2))\wedge(p(-1)\rightarrow p(1)\vee p(2))\wedge(p(1)\rightarrow p(-1)\vee p(2))}\\ \qquad\underline{\wedge(p(-1)\wedge p(1)\rightarrow p(2))\wedge(p(-1)\wedge p(2)\rightarrow p(1))]}\rightarrow p(2))\\ \wedge\ (\underline{[p(-1)\rightarrow p(1)\vee p(2)]}\rightarrow p(-1))\\ \wedge\ (p(-1)\rightarrow p(1))\ .\end{array}$$

Similar to (8), formula $FLP(\Pi)$ can also be simplified using the notion of maximal local power sets, which provides yet another characterization of the FLP semantics. We call the resulting propositional formula representation *MLPS-FLP*(Π).

Lemma 1. *Formula (9) is strongly equivalent to*

$$\bigwedge_{\langle B,T\rangle\ is\ a\ maximal\ LPS\ of\ \overline{\mathcal{I}}_\Pi(E)}\Big(\bigwedge_{A\in B}A\rightarrow\bigvee_{A\in HU_\Pi\setminus T}A\Big). \tag{10}$$

For example, $FLP(\Pi_1)$ has the same answer sets as the following *MLPS-FLP*(Π_1):

$$(\underline{[p(2)\wedge(p(-1)\rightarrow p(1))]}\rightarrow p(2))\wedge(\underline{[p(-1)\rightarrow p(1)\vee p(2)]}\rightarrow p(-1))\wedge(p(-1)\rightarrow p(1))\ . \tag{11}$$

3.3 A Reformulation of SPT-PDB Semantics

Consider the following formula modified from (9) by simply eliminating implications in favor of negations and disjunctions as in classical logic:

$$\bigwedge_{I \in \overline{\mathcal{I}}_{\Pi}(E)} \Big(\bigvee_{A \in I} \neg A \;\vee \bigvee_{A \in HU_{\Pi} \setminus I} A \Big) . \tag{12}$$

Formulas (12) and (9) are classically equivalent to each other, but not strongly equivalent. However, interestingly, (12) is strongly equivalent to (6), which in turn provides a simple reformulation of the SPT-PDB semantics, without involving the notion of local power sets. We define *modified FLP answer sets* of Π same as in Section 3.2 except that we refer to (12) in place of (9).

Proposition 3 *For any program Π, the modified FLP answer sets of Π are precisely the SPT-PDB answer sets of Π.*

For instance, the SPT-PDB answer sets of Π_1 are the same as the answer sets of the following formula:

$$\begin{array}{l}
(\underline{[(p(-1)\vee p(1)\vee p(2)) \wedge (\neg p(-1)\vee p(1)\vee p(2)) \wedge (\neg p(1)\vee p(-1)\vee p(2))} \\
\qquad \underline{\wedge(\neg p(-1)\vee\neg p(1)\vee p(2)) \wedge (\neg p(-1)\vee\neg p(2)\vee p(1))]} \rightarrow p(2)) \\
\wedge\ (\underline{[\neg p(-1)\vee p(1)\vee p(2)]} \rightarrow p(-1)) \\
\wedge\ (p(-1) \rightarrow p(1)) \ .
\end{array} \tag{13}$$

Similar to the Ferraris and the FLP semantics, considering maximal local power sets can yield shorter propositional formula representation as the following lemma tells.

Lemma 2. *Formula (12) is strongly equivalent to*

$$\bigwedge_{\langle B,T \rangle \text{ is a maximal LPS of } \overline{\mathcal{I}}_{\Pi}(E)} \Big(\bigvee_{A \in B} \neg A \vee \bigvee_{A \in HU_{\Pi} \setminus T} A \Big) . \tag{14}$$

Again note the similarity between (14) and (10). They are classically equivalent to each other, but not strongly equivalent.

3.4 Relationship between the Semantics

The characterizations of each semantics in terms of the uniform framework of propositional formulas give new insights into their relationships. Note that for any aggregate expression E, formulas $\mathit{SPT\text{-}PDB}_{\Pi}(E)$, $\mathit{FLP}_{\Pi}(E)$, $\mathit{Fer}_{\Pi}(E)$ are classically equivalent to each other, but not strongly equivalent.

It is not difficult to check that for any aggregate expression E occurring in $\mathit{Ground}(\mathit{Pos}(\Pi))$, formula $\mathit{SPT\text{-}PDB}_{\Pi}(E)$ entails $\mathit{FLP}_{\Pi}(E)$ under the logic of Here-and-There, but not the other way around. Using this fact, we can prove the following.

Proposition 4 *[5, Theorem 2] Every SPT-PDB answer set of Π is an FLP answer set of Π.*

For program Π_1, its only FLP answer set is a Ferraris answer set. Indeed, such relationship holds if the program is "semi-positive." We call a program *semi-positive* if, for every aggregate expression (2) occurring in it, $F(\mathbf{x})$ is a quantifier-free formula that contains no implications (this, in particular, means that there are no negations since we treat $\neg G$ as shorthand for $G \rightarrow \bot$). For example, Π_1 is semi-positive.

Proposition 5 *For any semi-positive program Π, every FLP answer set of Π is a Ferraris answer set of Π.*

However, the relationship does not hold for arbitrary programs. For instance, the following non-semi-positive program

$$\begin{array}{l} p(a) \leftarrow \text{COUNT}\langle\{x : \neg\neg p(x) \vee q(x)\}\rangle \neq 1 \\ q(b) \leftarrow p(a) \\ p(a) \leftarrow q(b) \end{array}$$

has no Ferraris answer sets while it has only one FLP answer set $\{p(a), q(b)\}$.

The following proposition is a slight extension of Theorem 3 from [6], which describes a class of programs whose FLP answer sets coincide with Ferraris answer sets.

Proposition 6 *For any semi-positive program Π, the FLP answer sets of Π are precisely the Ferraris answer sets of $Pos(\Pi)$.*

4 Generalized Definition of Aggregates

4.1 Syntax and Semantics of Aggregate Formulas

In this section we provide a general definition of a stable model that applies to arbitrary "aggregate formulas" in the style of the definition in Section 2.1, by extending the notion F^* to aggregate expressions in a way similar to other connectives and using the extended notion of satisfaction as in the FLP semantics (Section 2.3).

We allow the signature to contain any function constants of positive arity, and allow b in aggregate expression (2) to be any term. We define *aggregate formulas* as an extension of first-order formulas by treating aggregate expressions as a base case in addition to (standard) atomic formulas (including equality) and $\bot$ (falsity). In other words, aggregate formulas are constructed from atomic formulas and aggregate expressions using connectives and quantifiers as in first-order logic. For instance,

$$(\text{SUM}\langle\{x : p(x)\}\rangle \geq 1 \;\vee\; \exists y\, q(y)) \rightarrow r(x)$$

is an aggregate formula.

We say that an occurrence of a variable v in an aggregate formula H is *bound* if the occurrence is in a part of H of the form $\{\mathbf{x} : F(\mathbf{x})\}$ where v is in $\mathbf{x}$, or in a part of H of the form QvG. Otherwise it is *free*. We say that v is *free* in H if H

contains a free occurrence of v. An aggregate sentence is an aggregate formula with no free variables.

The definition of an interpretation is the same as in first-order logic. Consider an interpretation I of a first-order signature σ that may contain any function constants of positive arity. By $\sigma^{|I|}$ we mean the signature obtained from σ by adding distinct new object constants d^*, called *names*, for all d in the universe of I. We identify an interpretation I of σ with its extension to $\sigma^{|I|}$ defined by $I(d^*) = d$.

The notion of satisfaction in first-order logic is extended to aggregate sentences, similar to the definition given in Section 2.3. The integer constants and built-in symbols, such as $+$, $-$, $\leq$, $\geq$ are evaluated in the standard way, and we consider only those "standard" interpretations.[6] Let I be an interpretation of signature σ. Consider any aggregate expression (2) that has no free variables. Let S_I be the multiset consisting of all $\mathbf{d}[1]$ in the universe of I where

- $\mathbf{d}^*$ is a list of object names of $\sigma^{|I|}$ whose length is the same as the length of $\mathbf{x}$, and
- I satisfies $F(\mathbf{d}^*)$.

An interpretation I satisfies the aggregate expression if S_I is in the domain of *op*, and $op(S_I) \succeq b^I$.

For any aggregate sentence F, expression $\mathrm{SM}[F]$ stands for (1) where $F^*(\mathbf{u})$ is extended to aggregate expressions as

- $(\textsc{op}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b)^* = (\textsc{op}\langle\{\mathbf{x} : F^*(\mathbf{x})\}\rangle \succeq b) \wedge (\textsc{op}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b)$.

By a *stable model* of F we mean a model of $\mathrm{SM}[F]$ (under the extended notion of satisfaction).

4.2 Programs with Aggregates as a Special Case

The *AF-representation* ("Aggregate Formula representation") of (3) is the universal closure of the aggregate formula

$$E_1 \wedge \cdots \wedge E_m \wedge \neg E_{m+1} \wedge \cdots \wedge \neg E_n \rightarrow A_1 \vee \cdots \vee A_l. \quad (15)$$

The *AF-representation* of Π is the conjunction of the AF-representation of its rules.

The *stable models* of Π are defined as the stable models of the AF-representation of Π. The following proposition shows that this definition is a proper generalization of the Ferraris semantics.

Proposition 7 *Let Π be a program that contains no function constants of positive arity and let F be its AF-representation. The Herbrand stable models of F whose signature is $\sigma(\Pi)$ are precisely the Ferraris answer sets of Π.*

[6] For instance, we assume that, when x or y is not an integer, $x \leq y$ evaluates to false, and $x + y$ has an arbitrary value according to the interpretation.

5 Loop Formulas for Programs with Aggregates

Let us identify rule (3) with

$$A \leftarrow B, C, N \tag{16}$$

where $A = \{A_1, \ldots, A_l\}$, B is the set of all atoms (i.e., non-equality atomic formulas) from $\{E_1, \ldots, E_m\}$, C is the set of all aggregate expressions from $\{E_1, \ldots, E_m\}$ and N is the set of the remaining expressions in the body. We assume that the rules contain no free variables and no function constants of positive arity and that $F(\mathbf{x})$ in every aggregate expression (2) is a conjunction of atoms. Following [12], for any aggregate expression $E = \text{OP}\langle\{\mathbf{x} : F(\mathbf{x})\}\rangle \succeq b$ and any finite set Y of ground atoms, formula $\mathit{NFES}_E(Y)$ is defined as the conjunction of

$$\text{OP}\langle\{\mathbf{x} \ : \ F(\mathbf{x}) \wedge \bigwedge_{\substack{p_i(\mathbf{t}) \text{ occurs in } F(\mathbf{x}) \\ p_i(\mathbf{t}')\in Y}} \mathbf{t} \neq \mathbf{t}'\}\rangle \succeq b$$

and E. For instance, in Example Π_1, formula $\mathit{NFES}_{E_2}(\{p(-1), p(1)\})$ is

$$\text{SUM}\langle\{x : p(x) \wedge x{\neq}{-1} \wedge x{\neq}1\}\rangle \geq 0 \ \wedge \ \text{SUM}\langle\{x : p(x)\}\rangle \geq 0 \ .$$

For any finite set Y of expressions, by $Y^\wedge$ and $Y^\vee$ we denote the conjunction and, respectively, disjunction of the elements of Y. We define the *external support formula* of Y for Π, denoted by $\mathit{ES}_\Pi(Y)$, as the disjunction of

$$B^\wedge \wedge \bigwedge_{E\in C} \mathit{NFES}_E(Y) \wedge N^\wedge \wedge \ \neg(A \setminus Y)^\vee$$

for all rules (3) in Π such that $A \cap Y \neq \emptyset$ and $B \cap Y = \emptyset$. The *(conjunctive) aggregate loop formula* of Y for Π is the aggregate formula

$$Y^\wedge \rightarrow \mathit{ES}_\Pi(Y) \ . \tag{17}$$

This definition extends the definition of a loop formula given in [13], which is limited to programs with monotone aggregates.

For instance, if Y is $\{p(-1), p(1)\}$, the loop formula of Y for Π_1 is

$$p(-1) \wedge p(1) \rightarrow (\text{SUM}\langle\{x : p(x) \wedge x{\neq}{-1} \wedge x{\neq}1\}\rangle \geq 0) \wedge \text{SUM}\langle\{x : p(x)\}\rangle \geq 0 \ .$$

The *PL representation* of (17) is the propositional formula obtained from (17) by replacing all occurrences of aggregate expressions E in it with $\mathit{MLPS\text{-}Fer}_\Pi(E)$.

The *Ferraris dependency graph* of Π is the directed graph such that

- its vertices are the ground atoms of $\sigma(\Pi)$;
- for every rule (16) in $\mathit{Ground}(\Pi)$, it has edges from each element of A to $p(\mathbf{t})$
  - if $p(\mathbf{t})$ is an element of B, or
  - if there are an aggregate expression (2) in C, a maximal local power set $\langle B', T\rangle$ of $\overline{\mathcal{C}}_\Pi((2))$ and an element $\mathbf{c}$ in $\mathbf{O}_\Pi((2)) \setminus T$ such that $p(\mathbf{t})$ belongs to $F(\mathbf{c})$.

It is not difficult to check that the Ferraris dependency graph of Π according to this definition is the same as the dependency graph of the propositional formula $\mathit{MLPS}\text{-}\mathit{Fer}(\Pi)$ according to [9]. A *loop* is a nonempty set L of ground atoms of $\sigma(\Pi)$ such that the subgraph of the dependency graph of Π induced by L is strongly connected. Again, L is a loop of Π according to this definition iff it is a loop of $\mathit{MLPS}\text{-}\mathit{Fer}(\Pi)$ according to [9].[7] For example, Π_1 has four loops: $\{p(-1)\}$, $\{p(1)\}$, $\{p(2)\}$, $\{p(-1), p(1)\}$.

Proposition 8 *For any set X of ground atoms of $\sigma(\Pi)$ that satisfies Π, the following conditions are equivalent to each other.*

(a) X is a Ferraris answer set of Π;
(b) for every loop Y of Π, X satisfies the aggregate loop formula of Y for Π;
(c) for every loop Y of Π, X satisfies the PL representation of the aggregate loop formula of Y for Π;
(d) for every loop Y of $\mathit{MLPS}\text{-}\mathit{Fer}(\Pi)$ according to [9], X satisfies the loop formula of Y for $\mathit{MLPS}\text{-}\mathit{Fer}(\Pi)$ according to [9].

This result can be extended to the general case when $F(\mathbf{x})$ in an aggregate expression (2) is an arbitrary quantifier-free formula, by using the notion of NFES_F that is defined in [12]. The definition of external support formula above is closely related to the definition of unfounded sets under the FLP semantics given in [14]. Indeed, one can define loop formulas and loops under the FLP semantics in a similar way based on the reductive approach.

You and Liu [15] presented the definition of loop formulas under the SPT-PDB semantics. We note that a set of ground atoms is a loop of Π according to their definition iff it is a loop of $\mathit{SPT}\text{-}\mathit{PDB}(\Pi)$ according to [9]. The same can be said about loop formulas.

6 Conclusion

The paper presented several reformulations of the semantics of aggregates in terms of propositional formulas. The resulting formulas are classically equivalent to each other but not strongly equivalent, which results in different semantics. The reformulations give us insights into each of the semantics in terms of the underlying general language. Guided by the reduction, we defined the loop formulas of a program with aggregates, which result in the same as loop formulas of the corresponding propositional formula representation.

The reductive approach led us to the general semantics of aggregates presented in Section 4, which extends the definition of a stable model of a first-order formula to an aggregate formula, using a notion of satisfaction extended from the one used in the FLP semantics. The new semantics is more general than that of RASPL-1 [16] in that it allows arbitrary aggregates and non-Herbrand stable models, along with built-in functions. On the other hand, it is not fully reductive; it requires the notion of satisfaction be extended to aggregate expressions, while

[7] Note that $\mathit{Fer}(\Pi)$ may contain redundant loops not present in $\mathit{MLPS}\text{-}\mathit{Fer}(\Pi)$. For example, consider $\quad p(1) \leftarrow \text{SUM}\langle\{x : p(x)\}\rangle \geq 1 \qquad p(-1) \leftarrow p(1)$.

a counting aggregate expression in RASPL-1 was defined as an abbreviation for a first-order formula.

Acknowledgements We are grateful to Vladimir Lifschitz and Tran Cao Son for helpful discussions and to the anonymous referees for their useful comments on this paper. This work was partially supported by the National Science Foundation under Grant IIS-0839821.

References

1. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Annals of Mathematics and Artificial Intelligence **25** (1999) 369–389
2. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Proc. European Conference on Logics in Artificial Intelligence (JELIA). (2004)
3. Son, T.C., Pontelli, E., Tu, P.H.: Answer sets for logic programs with arbitrary abstract constraint atoms. J. Artif. Intell. Res. (JAIR) **29** (2007) 353–389
4. Pelov, N., Denecker, M., Bruynooghe, M.: Translation of aggregate programs to normal logic programs. In: Proc. Answer Set Programming. (2003)
5. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. TPLP **7**(3) (2007) 355–375
6. Ferraris, P.: Answer sets for propositional theories. In: Proc. International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2005) 119–131
7. Pearce, D.: A new logical characterization of stable models and answer sets. In Dix, J., Pereira, L., Przymusinski, T., eds.: Non-Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216), Springer-Verlag (1997) 57–70
8. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI). (2007) 372–379
9. Ferraris, P., Lee, J., Lifschitz, V.: A generalization of the Lin-Zhao theorem. Annals of Mathematics and Artificial Intelligence **47** (2006) 79–101
10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proceedings of International Logic Programming Conference and Symposium, MIT Press (1988) 1070–1080
11. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2** (2001) 526–541
12. Lee, J., Meng, Y.: On loop formulas with variables. In: Proc. International Conference on Knowledge Representation and Reasoning (KR). (2008) 444–453
13. Liu, L., Truszczynski, M.: Properties and applications of programs with monotone and convex constraints. J. Artif. Intell. Res. (JAIR) **27** (2006) 299–334
14. Faber, W.: Unfounded sets for disjunctive logic programs with arbitrary aggregates. In: Proc. International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR). (2005) 40–52
15. You, J.H., Liu, G.: Loop formulas for logic programs with arbitrary constraint atoms. In: Proc. AAAI Conference on Artificial Intelligence (AAAI). (2008) 584–589
16. Lee, J., Lifschitz, V., Palla, R.: A reductive semantics for counting and choice in answer set programming. In: Proc. AAAI Conference on Artificial Intelligence (AAAI). (2008) 472–479