Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity*

Wolfgang Faber[1], Nicola Leone[2], and Gerald Pfeifer[1]

[1] Institut für Informationssysteme, TU Wien, A-1040 Wien, Austria
faber@kr.tuwien.ac.at, gerald@pfeifer.com
[2] Department of Mathematics, University of Calabria, I-87030 Rende (CS), Italy
leone@unical.it

Abstract. The addition of aggregates has been one of the most relevant enhancements to the language of answer set programming (ASP). They strengthen the modeling power of ASP, in terms of concise problem representations. While many important problems can be encoded using nonrecursive aggregates, some relevant examples lend themselves for the use of recursive aggregates. Previous semantic definitions typically agree in the nonrecursive case, but the picture is less clear for recursion. Some proposals explicitly avoid recursive aggregates, most others differ, and many of them do not satisfy desirable criteria, such as minimality or coincidence with answer sets in the aggregate-free case.
In this paper we define a semantics for disjunctive programs with *arbitrary* aggregates (including monotone, antimonotone, and nonmonotone aggregates). This semantics is a fully declarative, genuine generalization of the answer set semantics for disjunctive logic programming (DLP). It is defined by a natural variant of the Gelfond-Lifschitz transformation, and treats aggregate and non-aggregate literals in a uniform way. We prove that our semantics guarantees the minimality (and therefore the incomparability) of answer sets, and demonstrate that it coincides with the standard answer set semantics on aggregate-free programs. Finally we analyze the computational complexity of this language, paying particular attention to the impact of syntactical restrictions on programs.

1 Introduction

Aggregates significantly enhance the language of answer set programming (ASP), allowing for natural and concise modeling of many problems. Nonrecursive (also called stratified) aggregates have clear semantics and capture a large class of meaningful problem specifications. However, there are relevant problems for which recursive (unstratified) aggregate formulations are natural; the *Company Control* problem, illustrated next, is a typical example, cf. [1–4].

Example 1. We are given a set of facts for predicate $company(X)$, denoting the companies involved, and a set of facts for predicate $ownsStk(C1, C2, Perc)$, denoting the percentage of shares of company $C2$, which is owned by company $C1$. Then, company

* This work was supported by the European Commission under projects IST-2002-33570 INFOMIX, IST-2001-37004 WASP, and IST-2001-33570 COLOGNET.

$C1$ controls company $C2$ if the sum of the shares of $C2$ owned either directly by $C1$ or by companies, which are controlled by $C1$, is more than 50%. This problem has been encoded as the following program $\mathcal{P}_{ctrl}$ by many authors in the literature [1–4].[3]

$$
\begin{array}{l}
controlsStk(C1, C1, C2, P) \text{:}-\ ownsStk(C1, C2, P).\\
controlsStk(C1, C2, C3, P) \text{:}-\ company(C1),\ controls(C1, C2),\ ownsStk(C2, C3, P).\\
controls(C1, C3) \text{:}-\ company(C1),\ company(C3),\\
\qquad\qquad \#\mathtt{sum}\{P, C2 : controlsStk(C1, C2, C3, P)\} > 50.
\end{array}
$$

Intuitively, $controlsStk(C1, C2, C3, P)$ denotes that company $C1$ controls $P\%$ of $C3$ shares "through" company $C2$ (as $C1$ controls $C2$, and $C2$ owns $P\%$ of $C3$ shares). Predicate $controls(C1, C2)$ encodes that company $C1$ controls company $C2$. For two companies, say, $c1$ and $c3$, $controls(c1, c3)$ is derived if the sum of the elements in the *multiset* $\{P \mid \exists C2 : controlsStk(c1, C2, c3, P)\}$ is greater than 50. Note that in the DLV syntax this multiset is expressed by $\{P, C2 : controlsStk(c1, C2, c3, P)\}$ where the variable $C2$ avoids that duplicate occurrences of P are eliminated.

The encoding of *Company Control* contains a recursive aggregate (since predicate $controlsStk$ in the aggregate depends on the head predicate $controls$). Unfortunately, however, recursive aggregates are not easy to handle, and their semantics is not always straightforward.

Example 2. Consider the following two programs:

$$P_1 : \{p(a) \text{:}-\ \#\mathtt{count}\{X : p(X)\} > 0.\} \qquad P_2 : \{p(a) \text{:}-\ \#\mathtt{count}\{X : p(X)\} < 1.\}$$

In both cases $p(a)$ is the only atom for p which might be true, so, intuitively, one may expect that $\#\mathtt{count}\{X : p(X)\} > 0$ is true iff $p(a)$ is true; while $\#\mathtt{count}\{X : p(X)\} < 1$ should be true iff $p(a)$ is false. Thus, the above programs should, respectively, behave like the following standard programs:

$$P_1' : \{p(a) \text{:}-\ p(a).\} \qquad P_2' : \{p(a) \text{:}-\ \mathtt{not}\ p(a).\}$$

This is not always the case in the literature, and there is a debate on the best semantics for recursive aggregates.

There have been several attempts for defining a suitable semantics for aggregates [2, 6, 7, 4, 8]. However, while previous semantic definitions typically agree in the non-recursive case, the picture is not so clear for recursion. Some proposals explicitly avoid recursive aggregates, most others differ, and many of them do not satisfy desirable criteria, such as minimality[4]. Relevant progress towards a suitable semantics for recursive aggregates has been recently made in [4, 8], where the authors provide a semantics which guarantees minimality and extends standard answer sets. However, both definitions are given operationally and do not cover *all* language fragments. The first proposal disregards disjunctive programs, while the latter covers *only* monotone aggregates.

[3] Throughout this paper, we adopt the concrete syntax of the DLV language [5] to express aggregates in the examples.

[4] The subset-minimality of answer sets, which holds in the aggregate-free case and for the main nonmonotonic logics [9], also guarantees that answer sets are incomparable, and allows to define the transitive closure – which becomes impossible if minimality is lost [4].

In this paper, we make a step forward and provide a fully declarative semantics which works also for disjunctive programs and arbitrary aggregates. The main contributions of the paper are the following:

- We provide a definition of the answer sets semantics for disjunctive programs with arbitrary aggregates (including monotone aggregates, antimonotone aggregates, and aggregates which are neither monotone nor antimonotone). This semantics is fully declarative and is given in the standard way for answer sets, by a generalization of the well-known Gelfond-Lifschitz transformation.
- We study the properties of the proposed semantics, and show the following results:
  - Our answer sets are subset-minimal models, and therefore they are incomparable to each other, which is generally seen as an important property of nonmonotonic semantics [10, 4].
  - For aggregate-free programs, our semantics coincides with the standard answer set semantics.
  - From a semantic viewpoint, monotone aggregate literals correspond to positive standard literals, while antimonotone aggregates correspond to negative standard literals. We provide a rewriting from standard logic programs with negation to positive programs with antimonotone aggregate atoms.
- We carry out an in-depth analysis of the computational complexity of disjunctive programs with aggregates and fragments thereof. As long as the values of aggregates are computable in polynomial time, their addition does not increase the complexity of the full DLP language. However, the complexity of some fragments of DLP is affected by aggregates. Interestingly, monotone aggregates never alter the complexity, while antimonotone aggregates cause a complexity gap in many cases (see Section 4); arbitrary aggregates behave precisely like antimonotone aggregates from the complexity viewpoint in the studied cases.

2 The DLP$^{\mathcal{A}}$ Language

In this section, we provide a formal definition of the syntax and semantics of the DLP$^{\mathcal{A}}$ language – an extension of Disjunctive Logic Programming (DLP) by set-oriented functions (also called aggregate functions). We assume that the reader is familiar with standard DLP; we refer to atoms, literals, rules, and programs of DLP, as *standard atoms, standard literals, standard rules*, and *standard programs,* respectively. For further background, see [11, 12].

2.1 Syntax

Set Terms. A (DLP$^{\mathcal{A}}$) *set term* is either a symbolic set or a ground set. A *symbolic set* is a pair $\{\mathit{Vars} : \mathit{Conj}\}$, where Vars is a list of variables and Conj is a conjunction of standard literals.[5] A *ground set* is a set of pairs of the form $\langle \overline{t} : \mathit{Conj} \rangle$, where $\overline{t}$ is a list of constants and Conj is a ground (variable free) conjunction of standard literals.

[5] Intuitively, a symbolic set $\{X : a(X,Y), \texttt{not}\ p(Y)\}$ stands for the set of X-values making $a(X,Y), \texttt{not}\ p(Y)$ true, i.e., $\{X \mid \exists Y\, s.t.\ a(X,Y), \texttt{not}\ p(Y)\ \mathit{is\ true}\}$.

Aggregate Functions. An *aggregate function* is of the form $f(S)$, where S is a set term, and f is an *aggregate function symbol*. Intuitively, an aggregate function can be thought of as a (possibly partial) function mapping multisets[6] of constants to a constant.

Example 3. The aggregate functions currently supported by the DLV system are: $\#\texttt{min}$ (minimal term, undefined for empty set), $\#\texttt{max}$ (maximal term, undefined for empty set), $\#\texttt{count}$ (number of terms), $\#\texttt{sum}$ (sum of non-negative integers), and $\#\texttt{times}$ (product of positive integers).

Aggregate Literals. An *aggregate atom* is $f(S) \prec T$, where $f(S)$ is an aggregate function, $\prec \in \{=, <, \leq, >, \geq\}$ is a predefined comparison operator, and T is a term (variable or constant) referred to as guard.

Example 4. The following aggregate atoms in DLV notation, where the latter contains a ground set and could be a ground instance of the former:

$$\#\texttt{max}\{Z : r(Z), \texttt{not}\ a(Z, V)\} > Y$$
$$\#\texttt{max}\{\langle 2 : r(2), \texttt{not}\ a(2, x)\rangle, \langle 2 : r(2), \texttt{not}\ a(2, y)\rangle\} > 1$$

An *atom* is either a standard (DLP) atom or an aggregate atom. A *literal* L is an atom A or an atom A preceded by the default negation symbol $\texttt{not}$; if A is an aggregate atom, L is an *aggregate literal*.

DLP$^{\mathcal{A}}$ **Programs.** A *(*DLP$^{\mathcal{A}}$*) rule* r is a construct

$$a_1 \vee \cdots \vee a_n :\!- b_1, \cdots, b_k, \texttt{not}\ b_{k+1}, \cdots, \texttt{not}\ b_m.$$

where $a_1, \cdots, a_n$ are standard atoms, $b_1, \cdots, b_m$ are atoms, and $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$. The disjunction $a_1 \vee \cdots \vee a_n$ is referred to as the *head* of r, while the conjunction $b_1, ..., b_k, \texttt{not}\ b_{k+1}, ..., \texttt{not}\ b_m$ is the *body* of r. A *(*DLP$^{\mathcal{A}}$*) program* is a set of DLP$^{\mathcal{A}}$ rules.

Syntactic Properties A *global* variable of a rule r is a variable appearing in a standard atom of r; all other variables are *local* variables.

Safety. A rule r is *safe* if the following conditions hold: (i) each global variable of r appears in a positive standard literal in the body of r; (ii) each local variable of r appearing in a symbolic set $\{ Vars : Conj \}$ appears in a positive literal in $Conj$; (iii) each guard of an aggregate atom of r is a constant or a global variable. A program $\mathcal{P}$ is safe if all $r \in \mathcal{P}$ are safe. In the following we assume that DLP$^{\mathcal{A}}$ programs are safe.

Example 5. Consider the following rules with DLV aggregates:

$$p(X) :\!- q(X, Y, V), \#\texttt{max}\{Z : r(Z), \texttt{not}\ a(Z, V)\} > Y.$$
$$p(X) :\!- q(X, Y, V), \#\texttt{sum}\{Z : \texttt{not}\ a(Z, S)\} > Y.$$
$$p(X) :\!- q(X, Y, V), \#\texttt{min}\{Z : r(Z), \texttt{not}\ a(Z, V)\} > T.$$

The first rule is safe, while the second is not, since both local variables Z and S violate condition (ii). The third rule is not safe either, since the guard T violates condition (iii).

[6] Note that aggregate functions are evaluated on the valuation of a (ground) set w.r.t. an interpretation, which is a multiset, cf. Section 2.2.

Stratification. A DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is *aggregate-stratified* if there exists a function $||\ ||$, called *level mapping*, from the set of (standard) predicates of $\mathcal{P}$ to ordinals, such that for each pair a and b of standard predicates, occurring in the head and body of a rule $r \in \mathcal{P}$, respectively: (i) if b appears in an aggregate atom, then $||b|| < ||a||$, and (ii) if b occurs in a standard atom, then $||b|| \leq ||a||$.

Example 6. Consider the program consisting of a set of facts for predicates a and b, plus the following two rules:

$$q(X) \text{:}-p(X), \#\texttt{count}\{Y : a(Y,X), b(X)\} \leq 2. \qquad p(X) \text{:}-q(X), b(X).$$

The program is aggregate-stratified, as the level mapping $||a|| = ||b|| = 1, \quad ||p|| = ||q|| = 2$ satisfies the required conditions. If we add the rule $b(X) \text{:}-p(X)$, then no such level-mapping exists and the program becomes aggregate-unstratified.

Intuitively, aggregate-stratification forbids recursion through aggregates. While the semantics of aggregate-stratified programs is more or less agreed upon, different and disagreeing semantics for aggregate-unstratified programs have been defined in the past, cf. [4]. In the following we shall provide a novel characterization which directly extends well-known formulations of semantics for aggregate-free programs.

2.2 Semantics

Universe and Base. Given a DLP$^{\mathcal{A}}$ program $\mathcal{P}$, let $U_{\mathcal{P}}$ denote the set of constants appearing in $\mathcal{P}$, and $B_{\mathcal{P}}$ the set of standard atoms constructible from the (standard) predicates of $\mathcal{P}$ with constants in $U_{\mathcal{P}}$. Given a set X, let $\overline{2}^X$ denote the set of all multisets over elements from X. Without loss of generality, we assume that aggregate functions map to $\mathbb{I}$ (the set of integers).

Example 7. Let us now describe the domains of the aggregate functions in DLV (where $\mathbb{N}$ and $\mathbb{N}^+$ denote the set of non-negative integers and positive integers, respectively): $\#\texttt{count}$ is defined over $\overline{2}^{U_{\mathcal{P}}}$, $\#\texttt{sum}$ over $\overline{2}^{\mathbb{N}}$, $\#\texttt{times}$ over $\overline{2}^{\mathbb{N}^+}$, [7] $\#\texttt{min}$ and $\#\texttt{max}$ are defined over $\overline{2}^{\mathbb{N}} - \{\emptyset\}$.

Instantiation. A *substitution* is a mapping from a set of variables to $U_{\mathcal{P}}$. A substitution from the set of global variables of a rule r (to $U_{\mathcal{P}}$) is a *global substitution for r*; a substitution from the set of local variables of a symbolic set S (to $U_{\mathcal{P}}$) is a *local substitution for* S. Given a symbolic set without global variables $S = \{Vars : Conj\}$, the *instantiation of* S is the following ground set of pairs $inst(S)$:
$\{\langle \gamma(Vars) : \gamma(Conj)\rangle \mid \gamma \textit{ is a local substitution for } S\}$.[8]
A *ground instance* of a rule r is obtained in two steps: (1) a global substitution σ for r is first applied over r; (2) every symbolic set S in $\sigma(r)$ is replaced by its instantiation $inst(S)$. The instantiation $Ground(\mathcal{P})$ of a program $\mathcal{P}$ is the set of all possible instances of the rules of $\mathcal{P}$.

[7] $\#\texttt{sum}$ and $\#\texttt{times}$ applied over an empty set return 0 and 1, respectively.

[8] Given a substitution σ and a DLP$^{\mathcal{A}}$ object Obj (rule, set, etc.), we denote by $\sigma(Obj)$ the object obtained by replacing each variable X in Obj by $\sigma(X)$.

Example 8. Consider the following program $\mathcal{P}_1$:

$$q(1) \vee p(2,2). \qquad q(2) \vee p(2,1). \qquad t(X) \text{:-} q(X), \#\mathtt{sum}\{Y : p(X,Y)\} > 1.$$

The instantiation $Ground(\mathcal{P}_1)$ is the following:

$$\begin{array}{ll} q(1) \vee p(2,2). & t(1) \text{:-} q(1), \#\mathtt{sum}\{\langle 1 : p(1,1)\rangle, \langle 2 : p(1,2)\rangle\} > 1. \\ q(2) \vee p(2,1). & t(2) \text{:-} q(2), \#\mathtt{sum}\{\langle 1 : p(2,1)\rangle, \langle 2 : p(2,2)\rangle\} > 1. \end{array}$$

Interpretation. An *interpretation* for a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is a set of standard ground atoms $I \subseteq B_{\mathcal{P}}$. The truth valuation $I(A)$, where A is a standard ground literal or a standard ground conjunction, is defined in the usual way. An interpretation also provides a meaning to (ground) sets, aggregate functions and aggregate literals, namely a multiset, a value, and a truth value, respectively. Let $f(S)$ be a an aggregate function. The valuation $I(S)$ of S w.r.t. I is the multiset of the first constant of the elements in S whose conjunction is true w.r.t. I. More precisely, let $I(S)$ denote the multiset $[t_1 \mid \langle t_1, ..., t_n : Conj\rangle \in S \wedge\ Conj$ *is true w.r.t.* $I\,]$ The valuation $I(f(S))$ of an aggregate function $f(S)$ w.r.t. I is the result of the application of f on $I(S)$. If the multiset $I(S)$ is not in the domain of f, $I(f(S)) = \bot$ (where $\bot$ is a fixed symbol not occurring in $\mathcal{P}$).

An instantiated aggregate atom $A = f(S) \prec k$ is *true w.r.t.* I if: (i) $I(f(S)) \neq \bot$, and, (ii) $I(f(S)) \prec k$ holds; otherwise, A is false. An instantiated aggregate literal $\mathtt{not}\ A = \mathtt{not}\ f(S) \prec k$ is *true w.r.t.* I if (i) $I(f(S)) \neq \bot$, and, (ii) $I(f(S)) \prec k$ does not hold; otherwise, A is false. A rule r is *satisfied w.r.t.* I if some head atom is true w.r.t. I whenever all body literals are true w.r.t. I.

Example 9. Consider the atom $A = \#\mathtt{sum}\{\langle 1 : p(2,1)\rangle, \langle 2 : p(2,2)\rangle\} > 1$ from Example 8. Let S be the ground set in A. For the interpretation $I = \{q(2), p(2,2), t(2)\}$, $I(S) = [2]$, the application of $\#\mathtt{sum}$ over $[2]$ yields 2, and A is therefore true w.r.t. I, since $2 > 1$. I is a model of the program of Example 8.

Definition 1. *A ground literal ℓ is* monotone*, if for all interpretations I, J, such that $I \subseteq J$, ℓ is true w.r.t. I implies that ℓ is true w.r.t. J. A ground literal ℓ is* antimonotone*, if for all interpretations I, J, such that $I \subseteq J$, ℓ is true w.r.t. J implies that ℓ is true w.r.t. I. A ground literal ℓ is* nonmonotone*, if it is neither monotone nor antimonotone.*

Note that positive standard literals are monotone, whereas negative standard literals are antimonotone. Aggregate literals may be monotone, antimonotone or nonmonotone, regardless whether they are positive or negative.

Example 10. All ground instances of the following aggregate literals are monotone

$$\#\mathtt{count}\{Z : r(Z)\} > 1 \qquad \mathtt{not}\ \#\mathtt{count}\{Z : r(Z)\} < 1$$

while the following are antimonotone:

$$\#\mathtt{count}\{Z : r(Z)\} < 1 \qquad \mathtt{not}\ \#\mathtt{count}\{Z : r(Z)\} > 1$$

Nonmonotone literals include the sum over (possibly negative) integers and the average. Also, most monotone or antimonotone functions combined with the equality operator yield nonmonotone literals.

2.3 Answer Sets

We will next define the notion of answer sets for DLP$^{\mathcal{A}}$ programs. While usually this is done by first defining the notion of answer sets for positive programs (coinciding with the minimal model semantics) and then for negative programs by a stability condition on a reduct, once aggregates have to be considered, the notions of positive and negative literals are in general not clear. If only monotone and antimonotone aggregate atoms were considered, one could simply treat monotone literals like positive literals and antimonotone literals like negative ones, and follow the standard approach, as hinted at in [4]. Since we also consider nonmonotone aggregates, such a categorization is not feasible, and we rely on a definition which always employs a stability condition on a reduct.

The subsequent definitions are directly based on models: An interpretation M is a model of a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ if all $r \in Ground(\mathcal{P})$ are satisfied w.r.t. M. An interpretation M is a subset-minimal model of $\mathcal{P}$ if no $I \subset M$ is a model of $Ground(\mathcal{P})$.

Next we provide the transformation by which the reduct of a ground program w.r.t. an interpretation is formed. Note that this definition is a generalization of the Gelfond-Lifschitz transformation for DLP programs (see Theorem 3).

Definition 2. *Given a ground* DLP$^{\mathcal{A}}$ *program $\mathcal{P}$ and an interpretation I, let $\mathcal{P}^I$ denote the transformed program obtained from $\mathcal{P}$ by deleting rules in which a body literal is false w.r.t. I.*

Example 11. Consider Example 2: $Ground(P_1) = \{p(a){:-}\#\mathtt{count}\{\langle a : p(a)\rangle\} > 0.\}$ and $Ground(P_2) = \{p(a){:-}\#\mathtt{count}\{\langle a : p(a)\rangle\} < 1.\}$, and interpretation $I_1 = \{p(a)\}$, $I_2 = \emptyset$. Then, $Ground(P_1)^{I_1} = Ground(P_1)$, $Ground(P_1)^{I_2} = \emptyset$, and $Ground(P_2)^{I_1} = \emptyset$, $Ground(P_2)^{I_2} = Ground(P_2)$ hold.

We are now ready to formulate the stability criterion for answer sets.

Definition 3 (Answer Sets for DLP$^{\mathcal{A}}$ **Programs).** *Given a* DLP$^{\mathcal{A}}$ *program $\mathcal{P}$, an interpretation A of $Ground(\mathcal{P})$ is an answer set if it is a subset-minimal model of $Ground(\mathcal{P})^A$.*

Note that any answer set A of $\mathcal{P}$ is also a model of $\mathcal{P}$ because $Ground(\mathcal{P})^A \subseteq Ground(\mathcal{P})$, and rules in $Ground(\mathcal{P}) - Ground(\mathcal{P})^A$ are satisfied w.r.t. A.

Example 12. For the programs of Example 2, I_2 of Example 11 is the only answer set of P_1 (because I_1 is not a minimal model of $Ground(P_1)^{I_1}$), while P_2 admits no answer set (I_1 is not a minimal model of $Ground(P_2)^{I_1}$, and I_2 is not a model of $Ground(P_2) = Ground(P_2)^{I_2}$.
For Example 1 and the following input facts

$$company(a). \quad company(b). \quad company(c).$$
$$ownsStk(a, b, 40).\ ownsStk(c, b, 20).\ ownsStk(a, c, 40).\ ownsStk(b, c, 20).$$

only the set $A = \{controlsStk(a, a, b, 40),\ controlsStk(a, a, c, 40),\ controlsStk(b, b, c, 20),\ controlsStk(c, c, b, 20)\}$ (omitting facts) is an answer set, which means that no company controls another company. Note that $A_1 = A \cup \{controls(a, b),\ controls(a, c),\ controlsStk(a, b, c, 20),\ controlsStk(a, c, b, 20)\}$ is not an answer set, which is reasonable, since there is no basis for the truth of literals in $A_1 - A$.

This definition is a generalization and simplification of the definitions given in [13, 10]. In particular, different to [10], we define answer sets directly on top of the notion of models of DLP$^{\mathcal{A}}$ programs, rather than transforming them to a positive program.

3 Semantic Properties

A generally desirable and important property of nonmonotonic semantics is minimality [10, 4], in particular a semantics should refine the notion of minimal models. We now show that our semantics has this property.

Theorem 1. *Answer Sets of a* DLP$^{\mathcal{A}}$ *program* $\mathcal{P}$ *are subset-minimal models of* $\mathcal{P}$.

Proof. Our proof is by contradiction: Assume that I_1 is a model of $\mathcal{P}$, I_2 is an answer set of $\mathcal{P}$ and that $I_1 \subset I_2$.[9] Since I_2 is an answer set of $\mathcal{P}$, it is a subset-minimal model of $Ground(\mathcal{P})^{I_2}$ by Definition 3. Therefore, I_1 is not a model of $Ground(\mathcal{P})^{I_2}$ (otherwise, I_2 would not be a subset-minimal model of $Ground(\mathcal{P})^{I_2}$). Thus, some rule $r \in Ground(\mathcal{P})^{I_2}$ is not satisfied w.r.t. I_1. Since $Ground(\mathcal{P})^{I_2} \subseteq Ground(\mathcal{P})$, r is also in $Ground(\mathcal{P})$ and therefore I_1 cannot be a model of $\mathcal{P}$, contradicting the assumption.

Corollary 1. *Answer sets of a* DLP$^{\mathcal{A}}$ *program* $\mathcal{P}$ *are incomparable (w.r.t. set inclusion) among each other.*

Theorem 1 can be refined for DLP$^{\mathcal{A}}$ programs containing only monotone literals.

Theorem 2. *The answer sets of a* DLP$^{\mathcal{A}}$ *program* $\mathcal{P}$*, where* $\mathcal{P}$ *contains only monotone literals, are precisely the minimal models of* $\mathcal{P}$.

Proof. Let $\mathcal{P}$ be a DLP$^{\mathcal{A}}$ program containing only monotone literals, and I be a minimal model of $\mathcal{P}$. Clearly, I is also a model of $\mathcal{P}^I$. We again proceed by contradiction and show that no $J \subset I$ is a model of $\mathcal{P}^I$: Assume that such a model J of $\mathcal{P}$ exists and satisfies all rules in $Ground(\mathcal{P})^I$. All rules in $Ground(\mathcal{P}) - Ground(\mathcal{P})^I$ are satisfied by I because their body is false w.r.t. I. But since $\mathcal{P}$ contains only monotone literals, each false literal in I is also false in $J \subset I$, and hence J also satisfies all rules in $Ground(\mathcal{P}) - Ground(\mathcal{P})^I$ and would therefore be a model of $\mathcal{P}$, contradicting the assumption that I is a minimal model. Together with Theorem 1, the result follows.

Clearly, a very desirable feature of a semantics for an extended language is that it properly extends agreed-upon semantics of the base language, so that the semantics are equal on the base language. Therefore we next show that for DLP programs, our semantics coincides with the standard answer set semantics. Note that not all semantics which have been proposed for programs with aggregates meet this requirement, cf. [4].

Theorem 3. *Given a* DLP *program* $\mathcal{P}$*, an interpretation* I *is an answer set of* $\mathcal{P}$ *according to Definition 3 iff it is an answer set of* $\mathcal{P}$ *according to the standard definition via the classic Gelfond-Lifschitz transformation [11].*

[9] Throughout the paper, $\subset$ denotes *strict* set inclusion.

Proof. ($\Rightarrow$): Assume that I is an answer set w.r.t. Definition 3, i.e. I is a minimal model of $Ground(\mathcal{P})^I$. Let us denote the standard Gelfond-Lifschitz transformed program by $GL(Ground(\mathcal{P}), I)$. For each $r \in Ground(\mathcal{P})^I$ some $r' \in GL(Ground(\mathcal{P}), I)$ exists, which is obtained from r by removing all negative literals. Since $r \in Ground(\mathcal{P})^I$, all negative literals of r are true in I, and also in all $J \subseteq I$. For rules of which an $r'' \in GL(Ground(\mathcal{P}), I)$ exists but no corresponding rule in $Ground(\mathcal{P})^I$, some positive body literal of r'' is false w.r.t. I (hence r'' is not included in $Ground(\mathcal{P})^I$), and also false w.r.t. all $J \subseteq I$. Therefore (i) I is a model of $GL(Ground(\mathcal{P}), I)$ and (ii) no $J \subset I$ is a model of $GL(Ground(\mathcal{P}), I)$, as it would also be a model of $Ground(\mathcal{P})^I$ and I thus would not be a minimal model of $Ground(\mathcal{P})^I$. Hence I is a minimal model of $GL(Ground(\mathcal{P}), I)$ whenever it is a minimal model of $Ground(\mathcal{P})^I$.
($\Leftarrow$): Now assume that I is a standard answer set of $\mathcal{P}$, that is, I is a minimal model of $GL(Ground(\mathcal{P}), I)$. By similar reasoning as in ($\Rightarrow$) a rule $r \in GL(Ground(\mathcal{P}), I)$ with true body w.r.t. I has a corresponding rule $r' \in Ground(\mathcal{P})^I$ which contains the negative body of the original rule $r^o \in Ground(\mathcal{P})$, which is true w.r.t. all $J \subseteq I$. Any rule $r'' \in GL(Ground(\mathcal{P}), I)$ with false body w.r.t. I is not contained in $Ground(\mathcal{P})^I$, but it is satisfied in each $J \subseteq I$. Therefore (i) I is a model of $Ground(\mathcal{P})^I$ and (ii) no $J \subset I$ is a model of $Ground(\mathcal{P})^I$ (otherwise J would also be a model of $GL(Ground(\mathcal{P}), I)$). As a consequence, I is a minimal model of $Ground(\mathcal{P})^I$ whenever it is a minimal model of $GL(Ground(\mathcal{P}), I)$.

4 Computational Complexity

4.1 Complexity Framework

We analyze the complexity of DLP$^{\mathcal{A}}$ on **Cautious Reasoning**, a main reasoning task in nonmonotonic formalisms, amounting to the following decisional problem: Given a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ and a standard ground atom A, is A true in all answer sets of $\mathcal{P}$?

We consider propositional (i.e., variable-free) DLP$^{\mathcal{A}}$ programs, and polynomial-time computable aggregate functions (note that all sample aggregate functions appearing in this paper fall into this class).

4.2 Overview of Complexity Results

Table 1 summarizes the complexity results derived in the next sections. The rows specify the allowance of negation (`not`); the columns specify the allowance of aggregates, namely: M_s = stratified monotone aggregates, M = full (possibly recursive) monotone aggregates, A_s = stratified antimonotone aggregates, A = full antimonotone aggregates, N_s = stratified nonmonotone aggregates, and N = full nonmonotone aggregates.

The good news is that the addition of aggregates does not increase the complexity of disjunctive logic programming. Cautious reasoning on the full DLP$^{\mathcal{A}}$ language, including all considered types of aggregates (monotone, antimonotone, and nonmonotone) even unstratified, remains Π_2^P-complete, as for standard DLP.

The most "benign" aggregates, from the complexity viewpoint, are the monotone ones, whose addition does never cause any complexity increase, even for negation-free programs, and even for unstratified monotone aggregates.

Table 1. The Complexity of Cautious Reasoning on Disjunctive Programs with Aggregates

	$\emptyset$	$\{M_s\}$	$\{M\}$	$\{A_s\}$	$\{N_s\}$	$\{M_s, A_s, N_s\}$	$\{A\}$	$\{N\}$	$\{M, A, N\}$
negation-free	co-NP	co-NP	co-NP	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P
with negation	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P

On negation-free programs, the addition of either antimonotone or nonmonotone aggregates increases the complexity, jumping from co-NP to Π_2^P. In all other cases, the complexity remains the same as for standard programs.

4.3 Proofs of Hardness Results

An important observation is that negation can be rewritten to an antimonotone aggregate. It is therefore possible to turn aggregate-free programs with negation into corresponding positive programs with aggregates.

Definition 4. *Given an (aggregate-free)* DLP *program $\mathcal{P}$, let $\Gamma(\mathcal{P})$ be the* DLP$^{\mathcal{A}}$ *program, which is obtained by replacing each negative literal* `not` a *in $\mathcal{P}$ by* $\#\mathtt{count}\{\langle \epsilon : a\rangle\} < 1$*, where ϵ is an arbitrary constant.*

Theorem 4. *Each aggregate-free* DLP *program $\mathcal{P}$ can be transformed into an equivalent positive* DLP$^{\mathcal{A}}$ *program $\Gamma(\mathcal{P})$ with aggregate literals (all of which are antimonotone). If $\mathcal{P}$ is stratified w.r.t. negation, then $\Gamma(\mathcal{P})$ is aggregate-stratified (i.e., all aggregates in $\Gamma(\mathcal{P})$ are nonrecursive).*

Proof. Note that for any interpretation I, `not` a is true w.r.t. I iff $\#\mathtt{count}\{\langle \epsilon : a\rangle\} < 1$ is true w.r.t. I, and that $\#\mathtt{count}\{\langle \epsilon : a\rangle\} < 1$ is an antimonotone aggregate literal. By virtue of Theorem 3, our answer sets semantics (as in Definition 3) is equivalent to the standard answer set semantics. Thus, since the valuation of literals is equal in $\mathcal{P}$ and $\Gamma(\mathcal{P})$, both programs have the same answer sets.

Since aggregates take the place of negative literals, if the latter are nonrecursive in $\mathcal{P}$ (i.e., $\mathcal{P}$ is stratified w.r.t. negation), the former are nonrecursive as well (i.e., $\Gamma(\mathcal{P})$ is aggregate-stratified).

Theorem 5. *Let $\mathcal{P}$ be a* DLP *program. Then (i) $\Gamma(\mathcal{P})$ has the same size (i.e., number of rules and literals) as $\mathcal{P}$, and (ii) $\Gamma(\mathcal{P})$ is LOGSPACE computable from $\mathcal{P}$.*

Proof. The $\Gamma(\mathcal{P})$ transformation replaces each negative literal by an aggregate atom; and it does not add any further literal to the program. Therefore it does not increase the program size. It is easy to see that $\Gamma(\mathcal{P})$ can be computed by a LOGSPACE Turing Machine. Indeed, $\Gamma(\mathcal{P})$ can be generated by dealing with one rule of $\mathcal{P}$ at a time, without storing any intermediate data apart from a fixed number of indices.

Finally, we state the relation between antimonotone and nonmonotone literals.

Theorem 6. *Each* DLP$^{\mathcal{A}}$ *program, whose aggregates are all antimonotone, can be transformed into an equivalent program, whose aggregates are all nonmonotone.*

Proof. W.l.o.g. we will consider a ground program $\mathcal{P}$. We transform each antimonotone aggregate literal l containing the aggregate atom $f(S) \prec k$ to l' containing $f^l(S') \prec k$. We introduce three fresh constants τ, ϵ, and ν and a new predicate symbol $\mathit{\Pi}$. Let f^l be undefined for the multisets $[\tau]$ and $[\tau, \epsilon, \nu]$ and return a value making l true for $[\tau, \epsilon]$ (such a value does always exist); otherwise f^l is equal to f. Furthermore, S' is obtained by adding $\langle \tau : \mathit{\Pi}(\tau) \rangle$, $\langle \epsilon : \mathit{\Pi}(\epsilon) \rangle$, and $\langle \nu : \mathit{\Pi}(\nu) \rangle$ to the ground set S. The transformed program $\mathcal{P}'$ contains only nonmonotone aggregates and is equivalent to $\mathcal{P}$.

Theorem 7. *Each field of Table 1 states the proper hardness of the corresponding fragment of* DLP$^{\mathcal{A}}$.

Proof. The hardness results for all fields in the second row of Table 1 stem from the $\mathit{\Pi}_2^P$-hardness of disjunctive programs with negation [14].[10] The same result, together with Theorems 4 and 5, entails $\mathit{\Pi}_2^P$-hardness of all the DLP$^{\mathcal{A}}$ fragments admitting antimonotone aggregates. $\mathit{\Pi}_2^P$-hardness of all the DLP$^{\mathcal{A}}$ fragments with nonmonotone aggregates then follows from Theorem 6. Finally, the results in the first three entries in the first row stem from the co-NP-hardness of positive disjunctive programs [14].

4.4 Proofs of Membership Results

In the membership proofs, we will implicitly use the following lemma:

Lemma 1. *Given an interpretation I for a* DLP$^{\mathcal{A}}$ *program $\mathcal{P}$, the truth valuation of an aggregate atom L is computable in polynomial time.*

Proof. Let $L = f(T) \prec k$. To determine the truth valuation of L, we have to: (i) compute the valuation $I(T)$ of the ground set T w.r.t. I, (ii) apply the aggregate function f on $I(T)$, and (iii) compare the result of $f(I(T))$ with k w.r.t. $\prec$.

Computing the valuation of a ground set T only requires scanning each element $\langle t_1, ..., t_n : \mathit{Conj} \rangle$ of T, adding t_1 to the result multiset if Conj is true w.r.t. I. This is evidently polynomial, as is the application of the aggregate function on $I(T)$ in our framework (see Section 4.1). The comparison with k, finally, is straightforward.

Lemma 2. *Let $\mathcal{P}$ be a negation-free* DLP$^{\mathcal{A}}$ *program, whose aggregates are all monotone. A standard ground atom A is not a cautious consequence of $\mathcal{P}$, if and only if there exists a model M of $\mathcal{P}$ which does not contain A.*[11]

Proof. Observe first that, since $\mathcal{P}$ does not contain negation and only monotone aggregate literals, each literal appearing in $\mathcal{P}$ is monotone.

($\Leftarrow$): The existence of a model M of $\mathcal{P}$ not containing A, implies the existence of a minimal model M' of $\mathcal{P}$ (with $M' \subseteq M$) not containing A. By virtue of Theorem 2, M' is an answer set of $\mathcal{P}$. Therefore, A is not a cautious consequence of $\mathcal{P}$.

($\Rightarrow$): Since A is not a cautious consequence of $\mathcal{P}$, by definition of cautious reasoning, there exists an answer set M of $\mathcal{P}$ which does not contain A. By definition of answer sets, M is also a model of $\mathcal{P}$, as remarked after Definition 3.

[10] Recall that even for stratified negation cautious reasoning on disjunctive programs is $\mathit{\Pi}_2^P$-hard.
[11] Note that M can be *any* model, possibly non-minimal, of $\mathcal{P}$.

Theorem 8. *Cautious reasoning over negation-free disjunctive programs, whose aggregates are all monotone, is in co-NP.*

Proof. By Lemma 2 we can check whether a ground atom A *is not* a cautious consequence of a program $\mathcal{P}$ as follows: (i) Guess an interpretation M of $\mathcal{P}$, (ii) check that M is a model and $a \notin M$. The check is clearly polynomial-time computable, and the problem is therefore in co-NP.

Lemma 3. *Checking whether an interpretation M is an answer set of an arbitrary* DLP$^{\mathcal{A}}$ *program $\mathcal{P}$ is in co-NP.*

Proof. To prove that M is not an answer set of $\mathcal{P}$, we guess an interpretation M' of $\mathcal{P}$, and check that (at least) one of the following conditions hold: (i) M' is a model of $\mathcal{P}^M$, and $M' \subset M$, or (ii) M is not a model of $\mathcal{P}^M$. The checking of both conditions above is clearly in polynomial time, and the problem is therefore in co-NP.

Theorem 9. *Cautious reasoning over arbitrary* DLP$^{\mathcal{A}}$ *programs is in Π_2^P.*

Proof. We verify that a ground atom A is not a cautious consequence of a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ as follows: Guess an interpretation $M \subseteq B_{\mathcal{P}}$ and check that (1) M is an answer set for $\mathcal{P}$, and (2) A is not true w.r.t. M. Task (2) is clearly polynomial, while (1) is in co-NP by virtue of Lemma 3. The problem therefore lies in Π_2^P.

5 Related Work and Conclusions

There have been considerable efforts to define semantics for logic programs with recursive aggregates, but most works do not consider disjunctive programs or do not cover all kinds of aggregates. In [4] a partial stable semantics for non-disjunctive programs with aggregates has been defined, for which the "standard" total stable semantics is a special case, while in [8] a stable semantics for disjunctive programs with has been given; but only monotone aggregates are considered. These semantics guarantee the same benign properties as ours, namely minimality and coincidence with answer sets in the aggregate-free case. On the respective language fragment, [4] intuitively coincides with our semantics (but a formal demonstration is still to be done). For [8] there is a slight difference when an aggregate function in a negative literal is undefined. E.g., the program $\{cheap \text{ :- } \texttt{not } \#\texttt{max}\{X : salary(X)\} > 1000\}$ without facts for salary would yield the answer set $\{cheap\}$ w.r.t. [8], while our semantics admits only $\emptyset$.

A thorough discussion of pros and cons for the various approaches for recursive aggregates has been given in [4, 15], so we will only briefly compare our approach with previous ones on typical examples.

The approaches of [2, 6, 7] basically all admit non-minimal answer sets. In particular, program P_1 of Example 2 would have $\emptyset$ and $\{p(a)\}$ as answer sets. As shown in Example 12 (also by Theorem 1), the semantics proposed in this paper only admits $\emptyset$.

The approach of [13] is defined on non-disjunctive programs with particular kinds of aggregates (called cardinality and weight constraints), which basically correspond to programs with *count* and *sum* functions. As shown in [4], the program $\{a\text{:-}\#\texttt{sum}\{\langle 1 : \texttt{not } a\rangle\} \leq 0.\}$ admits two stable models, $\emptyset$ and $\{a\}$, according to [13], whereas our

semantics only allows for $\emptyset$ as an answer set. An extension to this approach has been presented in [10], which allows for arbitrary aggregates in non-disjunctive programs.

Finally, the work in [16] deals with the more abstract concept of generalized quantifiers, and the semantics therein shares several properties with ours.

Concluding, we proposed a declarative semantics for disjunctive programs with arbitrary aggregates. We demonstrated that our semantics is endowed with desirable properties. Importantly, we proved that aggregate literals do not increase the computational complexity of disjunctive programs in our approach. Future work concerns the design of efficient algorithms for the implementation of our proposal in the DLV system. Upon completion of this paper, we have learned that yet another semantics has been independently proposed in [15]; studying the relationship to it is also a subject for future work.

We would like to thank the anonymous reviewers for their useful comments.

References

1. Mumick, I.S., Pirahesh, H., Ramakrishnan, R.: The magic of duplicates and aggregates. In: VLDB'90 (1990) 264–277
2. Kemp, D.B., Stuckey, P.J.: Semantics of Logic Programs with Aggregates. In: ISLP'91, MIT Press (1991) 387–401
3. Ross, K.A., Sagiv, Y.: Monotonic Aggregation in Deductive Databases. JCSS **54** (1997) 79–97
4. Pelov, N., Denecker, M., Bruynooghe, M.: Partial stable models for logic programs with aggregates. In: LPNMR-7. LNCS 2923., Springer (2004) 207–219
5. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In: IJCAI 2003, Acapulco, Mexico, Morgan Kaufmann (2003) 847–852
6. Gelfond, M.: Representing Knowledge in A-Prolog. In: Computational Logic. Logic Programming and Beyond. LNCS 2408. Springer (2002) 413–451
7. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate Functions in DLV. In: ASP'03, Messina, Italy (2003) 274–288 Online at `http://CEUR-WS.org/Vol-78/`.
8. Pelov, N., Truszczyński, M.: Semantics of disjunctive programs with monotone aggregates - an operator-based approach. In: NMR 2004. (2004) 327–334
9. Marek, V.W., Remmel, J.B.: On Logic Programs with Cardinality Constraints. In: NMR'2002 (2002) 219–228
10. Marek, V.W., Remmel, J.B.: Set Constraints in Logic Programming. In: LPNMR-7. LNCS, Springer (2004) 167–179
11. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
12. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Declarative Problem-Solving Using the DLV System. In Minker, J., ed.: Logic-Based Artificial Intelligence. Kluwer (2000) 79–103
13. Niemelä, I., Simons, P., Soininen, T.: Stable Model Semantics of Weight Constraint Rules. In: LPNMR'99. Number 1730 in Lecture Notes in AI (LNAI), Springer (1999) 107–116
14. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys **33** (2001) 374–425
15. Pelov, N.: Semantics of Logic Programs with Aggregates. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium (2004)
16. Eiter, T., Gottlob, G., Veith, H.: Modular Logic Programming and Generalized Quantifiers. In: LPNMR'97. LNCS 1265, Springer (1997) 290–309