Artificial Intelligence 175 (2011) 278–298

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Semantics and complexity of recursive aggregates in answer set programming ☆

Wolfgang Faber *, Gerald Pfeifer, Nicola Leone

Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy

ARTICLE INFO

Article history:
Available online 3 April 2010

Keywords:
Nonmonotonic reasoning
Answer set programming
Aggregates
Computational complexity

ABSTRACT

The addition of aggregates has been one of the most relevant enhancements to the language of answer set programming (ASP). They strengthen the modelling power of ASP in terms of natural and concise problem representations. Previous semantic definitions typically agree in the case of non-recursive aggregates, but the picture is less clear for aggregates involved in 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 programs with arbitrary aggregates (including monotone, antimonotone, and nonmonotone aggregates) in the full ASP language allowing also for disjunction in the head (disjunctive logic programming — DLP). This semantics is a genuine generalization of the answer set semantics for DLP, it is defined by a natural variant of the Gelfond–Lifschitz transformation, and treats aggregate and non-aggregate literals in a uniform way. This novel transformation is interesting per se also in the aggregate-free case, since it is simpler than the original transformation and does not need to differentiate between positive and negative literals. We prove that our semantics guarantees the minimality (and therefore the incomparability) of answer sets, and we demonstrate that it coincides with the standard answer set semantics on aggregate-free programs.

Moreover, we carry out an in-depth study of the computational complexity of the language. The analysis pays particular attention to the impact of syntactical restrictions on programs in the form of limited use of aggregates, disjunction, and negation. While the addition of aggregates does not affect the complexity of the full DLP language, it turns out that their presence does increase the complexity of normal (i.e., non-disjunctive) ASP programs up to the second level of the polynomial hierarchy. However, we show that there are large classes of aggregates the addition of which does not cause any complexity gap even for normal programs, including the fragment allowing for arbitrary monotone, arbitrary antimonotone, and stratified (i.e., non-recursive) nonmonotone aggregates. The analysis provides some useful indications on the possibility to implement aggregates in existing reasoning engines.

☆ Parts of this work have been published in preliminary form in the proceedings of the conferences JELIA'04 (Faber et al. (2004) [1]) and IJCAI'05 (Calimeri et al. (2005) [2]).

* Corresponding author.
E-mail addresses: faber@mat.unical.it (W. Faber), gerald@pfeifer.com (G. Pfeifer), leone@mat.unical.it (N. Leone).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2010.04.002

1. Introduction

Around 1960, McCarthy proposed the use of *logical formulas* as a basis for a knowledge representation language [3,4]. It was soon realized, however, that classical logic is not always adequate to model commonsense reasoning [5]. As an alternative, it has been suggested to represent commonsense reasoning using logical languages with nonmonotonic consequence relations, which allow new knowledge to invalidate some of the previous conclusions. This observation has led to the development and investigation of new logical formalisms, *nonmonotonic logics*. The most famous of these are circumscription [6,7], default logic [8], and nonmonotonic modal logics [9–11]. More recently, from cross fertilizations between the field of nonmonotonic logics and that of logic programming, another nonmonotonic language, called Answer Set Programming (ASP) [12,13], has emerged.

Answer Set Programs [12,13], also called Disjunctive Logic Programs (DLP) [14], are logic programs where (nonmonotonic) negation may occur in the bodies, and disjunction may occur in the heads of rules. This language is very expressive in a precise mathematical sense: it allows to express every property of finite structures that is decidable in the complexity class Σ_2^P ($\mathrm{NP}^{\mathrm{NP}}$) [15]. The high expressive power of the language, along with its simplicity, and the availability of a number of efficient ASP systems [16–23], has encouraged the usage of ASP and the investigation of new constructs enhancing its capabilities. One of the most relevant improvements to the language of answer set programming has been the addition of aggregates [24–37].

Aggregates significantly enhance the language of answer set programming (ASP), allowing for natural and concise modelling of many problems. Non-recursive (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. [24–26,29].

Example 1.1. 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 $C1$ 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 [24–26,29].[1]

$$controlsStk(C1, C1, C2, P) \text{ :- } ownsStk(C1, C2, P).$$

$$controlsStk(C1, C2, C3, P) \text{ :- } company(C1), controls(C1, C2), ownsStk(C2, C3, P).$$

$$controls(C1, C3) \text{ :- } company(C1), company(C3),$$
$$\#\mathtt{sum}\{P, C2 : controlsStk(C1, C2, C3, P)\} > 50.$$

Intuitively, $controlsStk(C1, C2, C3, P)$ denotes that company $C1$ controls P percent of $C3$ shares "through" company $C2$ (as $C1$ controls $C2$, and $C2$ owns P percent 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 adopted 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 1.2. 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, following the closed-world assumption, 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 recursive aggregates [25,27–30,34–37]. However, while previous semantic definitions typically agree in the non-recursive case, the picture is not so clear for recursion. Some

[1] Throughout this paper, we adopt the concrete syntax of the DLV language [38] to express aggregates in the examples.

proposals explicitly avoid recursive aggregates, many others differ, and several of them do not satisfy desirable criteria, such as minimality.[2] For a more detailed analysis we refer to Section 5.

In this paper, we make a step forward and provide a fully declarative semantics which works for disjunctive programs and arbitrary aggregates. Moreover, we carry out an in-depth analysis of the computational complexity of ASP with aggregates, which pays particular attention to the impact of syntactical restrictions on programs in the form of limited use of aggregates, disjunction, and negation.

The main contributions of the paper are the following:

- We provide a definition of the answer set 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, which treats aggregate and non-aggregate literals in a uniform way. This novel transformation is interesting per se also in the aggregate-free case, since it is simpler than the original transformation and does not differentiate between the types of literals (positive and negative) in the program. Interestingly, the generality of this transformation allows for defining the semantics of arbitrary linguistic extensions of ASP, and has already been applied also in other contexts (see Section 5).
- 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 [32,29].
  - For aggregate-free programs, our semantics coincides with the standard answer set semantics.
  - From a semantic viewpoint, monotone aggregate literals are analogous to positive standard literals, while antimonotone aggregates are analogous 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 polynomial-time computable aggregate functions and fragments thereof, deriving a full picture of the complexity of the ASP languages where negation and/or disjunction are combined with the different kinds of aggregates (monotone, antimonotone, nonmonotone, stratified).[3] The analysis brings many interesting results, including the following:
  - The addition of aggregates does not increase the complexity of the full ASP language. Cautious reasoning on full ASP programs (with disjunction and negation) including all considered types of aggregates (monotone, antimonotone, and nonmonotone) even unstratified, remains Π_2^P-complete, as for standard DLP.
  - The "cheapest" aggregates, from the complexity viewpoint, are the monotone ones, the addition of which does never cause any complexity increase, even for negation-free programs, and even for unstratified monotone aggregates.
  - The "hardest" aggregates, from the complexity viewpoint, are the nonmonotone ones: even on non-disjunctive positive programs (definite horn clauses), their addition causes a big complexity jump from P up to Π_2^P. Instead, antimonotone aggregates behave like negation: on non-disjunctive positive programs their presence increases the complexity from P to co-NP.
  - The largest set of aggregates which can be added to non-disjunctive ASP without inducing a complexity overhead consists of arbitrary monotone, arbitrary antimonotone, and stratified nonmonotone aggregates. When adding these kinds of aggregates to non-disjunctive ASP, the complexity of reasoning remains in co-NP.

Importantly, the above mentioned complexity results give us valuable information about intertranslatability of different languages, having relevant implications also on the possibility to implement aggregates in existing reasoning engines, or using rewriting-based techniques (like those employed in ASSAT [39] or Cmodels [20]) for their implementation (see Section 4.2).

The sequel of the paper is organized as follows. Section 2 defines the syntax and the formal semantics, based on the notion of answer set, of DLP$^{\mathcal{A}}$ — our extension of DLP with aggregates. Section 3 studies the semantic properties of DLP$^{\mathcal{A}}$; while Section 4 carries out the computational complexity analysis, and Section 5 discusses related work. Section 6 draws our conclusion.

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). For further background on DLP, we refer to [13,18].

[2] The subset-minimality of answer sets, which holds in the aggregate-free case and for the main nonmonotonic logics [31], also guarantees that answer sets are incomparable, and allows to define the transitive closure — which becomes impossible if minimality is lost [29].

[3] Note that the results mentioned here refer to the complexity of propositional programs. In Section 4.2, however, we discuss also the complexity of non-ground programs.

2.1. Syntax

We assume sets of *variables*, *constants*, and *predicates* to be given. Similar to Prolog, we assume variables to be strings starting with uppercase letters and constants to be integers or strings starting with lowercase letters. Predicates are strings starting with lowercase letters or symbols such as $=$, $<$, $>$ (so-called built-in predicates that have a fixed meaning). An *arity* (non-negative integer) is associated with each predicate.

Standard atoms and literals. A *term* is either a variable or a constant. A *standard atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity n and $t_1, \ldots, t_n$ are terms. A *standard literal* L is either a standard atom A (in this case, it is *positive*) or a standard atom A preceded by the default negation symbol `not` (in this case, it is *negative*). A conjunction of standard literals is of the form $L_1, \ldots, L_k$ where each L_i $(1 \leqslant i \leqslant k)$ is a standard literal.

An expression (e.g. standard atom, standard literal, conjunction) is ground, if neither the expression itself nor any of its subexpressions contain variables.

Set terms. A (DLP$^{\mathcal{A}}$) *set term* is either a symbolic set or a ground set. A *symbolic set* is a pair $\{Vars : Conj\}$, where *Vars* is a list of variables and *Conj* is a conjunction of standard atoms.[4] A *ground set* is a set of pairs of the form $\langle \tilde{t} : Conj \rangle$, where $\tilde{t}$ is a list of constants and *Conj* is a ground (variable free) conjunction of standard atoms.

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[5] of constants to a constant.

Example 2.1. The following aggregate functions are quite common, and currently supported also by the DLV system: `#min` (minimal term, undefined for empty set), `#max` (maximal term, undefined for empty set), `#count` (number of terms), `#sum` (sum of integers), and `#times` (product of integers).

Aggregate literals. An *aggregate atom* is $f(S) \circ T$, where $f(S)$ is an aggregate function, $\circ \in \{=, <, \leqslant, >, \geqslant, \neq\}$ is a comparison operator, and T is a term (variable or constant).

We note that our choice for the notation of aggregate atoms is primarily motivated by readability. One could define aggregate atoms as an arbitrary relation over a sequence of aggregate functions and terms. In fact, aggregates in DLV and cardinality and weight constraints for Smodels can be of the form $T \circ f(S) \circ U$, but semantically this is a shorthand for the conjunction of $T \circ f(S)$ and $f(S) \circ U$.

Example 2.2. The following are aggregate atoms in DLV notation, where the latter contains a ground set and could be a ground instance of the former:

$$\texttt{\#max}\{Z : r(Z), a(Z, V)\} > Y.$$
$$\texttt{\#max}\{\langle 2 : r(2), a(2, x)\rangle, \langle 2 : r(2), a(2, y)\rangle\} > 1.$$

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 `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

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

where $a_1, \ldots, a_n$ are standard atoms, $b_1, \ldots, b_m$ are atoms, and $n \geqslant 0$, $m \geqslant k \geqslant 0$, $n + m > 0$. The disjunction $a_1 \vee \cdots \vee a_n$ is referred to as the *head* of r, while the conjunction $b_1, \ldots, b_k, \texttt{not}\ b_{k+1}, \ldots, \texttt{not}\ b_m$ is the *body* of r. Let $H(r) = \{a_1, \ldots, a_n\}$, $B^+(r) = \{b_1, \ldots, b_k\}$, $B^-(r) = \{\texttt{not}\ b_{k+1}, \ldots, \texttt{not}\ b_m\}$, and $B(r) = B^+(r) \cup B^-(r)$. Furthermore let $Pred(\sigma)$ denote the set of predicates that occur in σ, where σ may be a program, a rule, a set of atoms or literals, an atom or a literal. Whenever it is clear that this set has one element (for standard atoms and literals), $Pred(\sigma)$ may also denote a single predicate. A (DLP$^{\mathcal{A}}$) *program* is a set of DLP$^{\mathcal{A}}$ rules.

2.1.1. Syntactic properties

A *local* variable of r is a variable appearing solely in an aggregate function in r; a variable of r which is not local is called *global*. A *nested* atom of r is an atom appearing in an aggregate atom of r; an atom of r which is not nested is called *unnested*.

[4] Intuitively, a symbolic set $\{X : a(X, Y), p(Y)\}$ stands for the set of X-values making $a(X, Y), p(Y)$ true, i.e., $\{X \mid \exists Y s.t.\ a(X, Y), p(Y)\ \textit{is true}\}$.

[5] 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.

Definition 2.1 *(Safety).* A rule r is *safe* if the following conditions hold:

(i) each global variable of r appears in a positive standard unnested literal of the body of r;
(ii) each local variable of r that appears in a symbolic set $\{Vars : Conj\}$ also appears in $Conj$. Finally, a program is safe if all of its rules are safe.

Condition (i) is the standard safety condition adopted in datalog, to guarantee that the variables are range restricted [40], while Condition (ii) is specific for aggregates.

Example 2.3. Consider the following rules:

$$p(X) : -q(X,Y,V), \qquad Y < \#\mathtt{max}\{Z : r(Z), a(Z,V)\}.$$
$$p(X) : -q(X,Y,V), \qquad Y < \#\mathtt{sum}\{Z : a(X,S)\}.$$
$$p(X) : -q(X,Y,V), \qquad T < \#\mathtt{min}\{Z : r(Z), a(Z,V)\}.$$

The first rule is safe, while the second is not, since local variables Z violates condition (ii). The third rule is not safe either, since the global variable T violates condition (i).

Definition 2.2 *(Aggregate-stratification).* A DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is *stratified on an aggregate atom* A if there exists a level mapping $\|\ \|$ from $Pred(\mathcal{P})$ to ordinals, such that for each rule $r \in \mathcal{P}$ and for each $a \in Pred(H(r))$ the following holds:

(1) For each $b \in Pred(B(r))$: $\|b\| \leqslant \|a\|$,
(2) if $A \in B(r)$, then for each $b \in Pred(A)$: $\|b\| < \|a\|$, and
(3) for each $b \in Pred(H(r))$: $\|b\| = \|a\|$.

A DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is *aggregate-stratified* if it is stratified on all aggregate atoms in $\mathcal{P}$.

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, see for instance the discussion in [29]. In this paper we shall provide a novel characterization which directly extends well-known formulations of semantics for aggregate-free programs.

Example 2.4. Consider the program consisting of a set of facts for predicates a and b, plus the following two rules:

$$q(X) : -p(X), \quad \#\mathtt{count}\{Y : a(Y,X), b(X)\} \leqslant 2. \qquad p(X) : -q(X), b(X).$$

The program is stratified on $\#\mathtt{count}\{Y : a(Y,X), b(X)\} \leqslant 2$, as the level mapping $\|a\| = \|b\| = 1$, $\|p\| = \|q\| = 2$ satisfies the required conditions. The program is therefore aggregate-stratified.

If we add the rule $b(X) : -p(X)$, then no such level-mapping exists and the program becomes aggregate-unstratified.

Definition 2.3 *(Negation-stratification).* A program $\mathcal{P}$ is called *negation-stratified* [41,42], if there exists a level mapping $\|\ \|_n$ for $Pred(\mathcal{P})$ such that for each rule $r \in \mathcal{P}$ and for each $a \in Pred(H(r))$ the following holds:

(1) For each $b \in Pred(B(r))$: $\|b\| \leqslant \|a\|$,
(2) for each standard literal $L \in B^-(r)$: $\|Pred(L)\| < \|a\|$, and
(3) for each $b \in Pred(H(r))$: $\|b\| = \|a\|$.

We note that when dealing with ground programs, one can consider a program in which each ground standard atom is replaced by a unique predicate with arity 0. This program is clearly equivalent to the original program, modulo the renaming. One can then consider the rewritten program for determining aggregate- and negation-stratification.

Example 2.5. Consider the following ground program:

$$p(a) : -\mathtt{not}\ p(b). \qquad p(a) : -\#\mathtt{count}\{\langle c : p(c)\rangle\} > 0.$$

While it is neither aggregate-stratified nor negation-stratified according to the definition, as it only considers the predicate symbol p, its renamed variant

$$pa : -\mathtt{not}\ pb. \qquad pa : -\#\mathtt{count}\{\langle c : pc\rangle\} > 0.$$

is, however, aggregate-stratified and negation-stratified, and so we may consider also the original program as being aggregate-stratified and negation-stratified.

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 $\mathbb{Z}$ (the set of integers).

Example 2.6. Let us look at common domains for the aggregate functions of Example 2.1: `#count` is defined over $\overline{2}^{U_{\mathcal{P}}}$, `#sum` over $\overline{2}^{\mathbb{Z}}$, `#times` over $\overline{2}^{\mathbb{Z}}$, `#min` and `#max` are defined over $\overline{2}^{\mathbb{Z}} \setminus \{\emptyset\}$.

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 $inst(S)$:

$$\big\{\langle \gamma(Vars) : \gamma(Conj)\rangle \;\big|\; \gamma \text{ is a local substitution for } S\big\}.^{6}$$

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 $\mathcal{P}$.

Example 2.7. Consider the following program $\mathcal{P}_1$:

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

Here $U_{\mathcal{P}_1} = \{1,2\}$ and the instantiation $Ground(\mathcal{P}_1)$ is the following:

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

Interpretation. An *interpretation* for a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ is a set of standard ground atoms $I \subseteq B_{\mathcal{P}}$. A standard ground atom a is true w.r.t. an interpretation I, denoted $I \models a$, if $a \in I$; otherwise it is false w.r.t. I. A standard ground literal `not` a is true w.r.t. an interpretation I, denoted $I \models$ `not` a, if $I \not\models a$, otherwise it is false w.r.t. I.

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 an aggregate function. The valuation $I(S)$ of S w.r.t. I is the multiset $I(S)$ defined as follows: Let $S_I = \{\langle t_1, \ldots, t_n\rangle \mid \langle t_1, \ldots, t_n : Conj\rangle \in S \wedge Conj \text{ is true w.r.t. } I\}$, then $I(S)$ is the multiset obtained as the projection of the tuples of S_I on their first constant, that is $I(S) = \{\{t_1 \mid \langle t_1, \ldots, t_n\rangle \in S_I\}\}$.

The valuation $I(f(S))$ of an aggregate function $f(S)$ w.r.t. I is the result of the application of f[7] 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 $A = f(S) \circ k$ is true w.r.t. an interpretation I, denoted $I \models A$ if: (i) $I(f(S)) \neq \bot$, and, (ii) $I(f(S)) \circ k$ holds[8]; otherwise, A is false. An instantiated aggregate literal `not` $A =$ `not` $f(S) \circ k$ is true w.r.t. an interpretation I, denoted $I \models$ `not` A, if (i) $I(f(S)) \neq \bot$, and, (ii) $I(f(S)) \circ k$ does not hold; otherwise, `not` A is false.

Example 2.8. Let I be the interpretation $\{f(1), g(1,2), g(1,3), g(1,4), g(2,4),\ h(2),\ h(3), h(4)\}$. With respect to the interpretation I, and assuming that all variables are local, we can check that:

- `#count`$\{X : g(X,Y)\} > 2$ is false, because S_I for the corresponding ground set is $\{\langle 1\rangle, \langle 2\rangle\}$, so $I(S) = \{\{1,2\}\}$ and `#count`$(\{\{1,2\}\}) = 2$.
- `#count`$\{X, Y : g(X,Y)\} > 2$ is true, because here $S_I = \{\langle 1,2\rangle,\ \langle 1,3\rangle,\ \langle 1,4\rangle,\ \langle 2,4\rangle\}$, $I(S) = \{\{1,1,1,2\}\}$ and `#count`$(\{\{1,1,1,2\}\}) = 4$.
- $23 <$ `#times`$\{Y : f(X), g(X,Y)\} \leqslant 24$ is true; in this case $S_I = \{\langle 2\rangle,\ \langle 3\rangle,\ \langle 4\rangle\}$, $I(S) = \{\{2,3,4\}\}$ and `#times`$(\{\{2,3,4\}\}) = 24$.
- `#sum`$\{A : g(A,B), h(B)\} \leqslant 3$ is true, as we have that $S_I = \{\langle 1\rangle, \langle 2\rangle\}$, $I(S) = \{\{1,2\}\}$ and `#sum`$(\{\{1,2\}\}) = 3$.
- `#sum`$\{A, B : g(A,B), h(B)\} \leqslant 3$ is false, since $S_I = \{\langle 1,2\rangle,\ \langle 1,3\rangle,\ \langle 1,4\rangle,\ \langle 2,4\rangle\}$, $I(S) = \{\{1,1,1,2\}\}$ and `#sum`$(\{\{1,1,1,2\}\}) = 5$.
- `#min`$\{X : f(X), g(X)\} \geqslant 2$ is false because the evaluation of (the instantiation of) $\{X : f(X), g(X)\}$ w.r.t. I yields the empty set, which does not belong to the domain of `#min` (we have that $I($`#min`$\{\}) = \bot$).

[6] 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 $\sigma(X)$.

[7] We assume that f has a fixed interpretation.

[8] Again, we assume that $\circ$ has a fixed interpretation.

A rule r is *satisfied w.r.t. I*, denoted $I \models r$ if some head atom is true w.r.t. I ($\exists h \in H(r) : I \models h$) whenever all body literals are true w.r.t. I ($\forall b \in B(r) : I \models b$).

Example 2.9. Consider the atom $A = \#\mathtt{sum}\{\langle 1 : p(2,1)\rangle, \langle 2 : p(2,2)\rangle\} > 1$ from Example 2.7. 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 therefore $I \models A$, since $2 > 1$.

Definition 2.4. A ground literal ℓ is

- *monotone*, if for all interpretations I, J, such that $I \subseteq J$, $I \models \ell$ implies $J \models \ell$;
- *antimonotone*, if for all interpretations I, J, such that $I \subseteq J$, $J \models \ell$ implies $I \models \ell$;
- *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 2.10. 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, which however may be decomposed into a conjunction of a monotone and an antimonotone aggregate.

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 [29]. 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 $\mathcal{P}$, denoted $M \models \mathcal{P}$, if $M \models r$ for all rules $r \in \mathit{Ground}(\mathcal{P})$. An interpretation M is a subset-minimal model of $\mathcal{P}$ if no $I \subset M$ is a model of $\mathit{Ground}(\mathcal{P})$.

Example 2.11. It can be verified that $\{q(2), p(2,2), t(2)\}$ is a model of the program of Example 2.7.

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 3.6). The intuition is, however, very similar: Treating an interpretation as an assumption, create the part of the program which is relevant according to the given interpretation. In particular, we consider any rule whose body is not satisfied as irrelevant.

Definition 2.5. 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:

$$\mathcal{P}^I = \{r \mid r \in \mathcal{P},\ \forall b \in B(r) : I \models b\}.$$

Example 2.12. Consider Example 1.2:

$$\mathit{Ground}(P_1) = \{p(a) : -\#\mathtt{count}\{\langle a : p(a)\rangle\} > 0\}.$$
$$\mathit{Ground}(P_2) = \{p(a) : -\#\mathtt{count}\{\langle a : p(a)\rangle\} < 1\}.$$

With interpretations $I_1 = \{p(a)\}$ and $I_2 = \emptyset$ we obtain:

$Ground(P_1)^{I_1} = Ground(P_1)$.

$Ground(P_1)^{I_2} = \emptyset$.

$Ground(P_2)^{I_1} = \emptyset$.

$Ground(P_2)^{I_2} = Ground(P_2)$.

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

Definition 2.6 *(Answer sets for* DLP$^{\mathcal{A}}$ *programs).* Given a DLP$^{\mathcal{A}}$ program $\mathcal{P}$, an interpretation A of $\mathcal{P}$ is an answer set if it is a subset-minimal model of $Ground(\mathcal{P})^A$.

It should be noted that this definition grasps the original motivation for answer sets or stable models, in that an interpretation is a stable model or an answer set if and only if it is a non-redundant explanation of the part of the program which is relevant to it. Looking in particular at aggregates, we observe that aggregates are treated as "black boxes" or "monoliths," that is when checking stability they are either present in their entirety or missing altogether. This is one of the main and distinguishing features of our semantics. Indeed, in Section 5 we will discuss that some other approaches to semantics for programs containing aggregates do not treat aggregates as monoliths.

It is also worth noting that this definition is very general, since it treats all atoms as black boxes. In fact, it is applicable to programs containing arbitrary forms of atoms, as long as their satisfaction by an interpretation can be determined. That means that the syntax adopted for aggregate literals is irrelevant for the definition, and that this definition can and indeed has been used (cf. Section 5) for programs containing arbitrary kinds of atoms.

Example 2.13. For the programs of Example 1.2, I_2 of Example 2.12 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 1.1 and the following input facts

$company(a).$ $company(b).$ $company(c).$

$ownsStk(a, b, 40).$ $ownsStk(c, b, 20).$ $ownsStk(a, c, 40).$ $ownsStk(b, c, 20).$

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 somewhat simpler than the definitions given in [43,32]. In particular, different to [32], 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

We first note two simple consequences of Definition 2.6.

Proposition 3.1. *Any answer set A of a* DLP$^{\mathcal{A}}$ *program $\mathcal{P}$ is a model of $\mathcal{P}$.*

Proof. Since $Ground(\mathcal{P})^A \subseteq Ground(\mathcal{P})$, A satisfies all rules in $Ground(\mathcal{P})^A$, and rules in $Ground(\mathcal{P}) - Ground(\mathcal{P})^A$ are satisfied w.r.t. A by the definition of $Ground(\mathcal{P})^A$. □

Moreover, each answer set is an answer set of its program reduct.

Proposition 3.2. *Any answer set A of a* DLP$^{\mathcal{A}}$ *program $\mathcal{P}$ is an answer set of $Ground(\mathcal{P})^A$.*

Proof. We note that $Ground(Ground(\mathcal{P})^A) = Ground(\mathcal{P})^A$ and that $Ground(\mathcal{P})^{A\,A} = Ground(\mathcal{P})^A$. Since A is an answer set of $\mathcal{P}$, it is a subset-minimal model of $Ground(\mathcal{P})^A = Ground(Ground(\mathcal{P})^A)^A$. □

A generally desirable and important property of nonmonotonic semantics is minimality [32,29], in particular a semantics should refine the notion of minimal models. We now show that our semantics has this property.

Theorem 3.3. *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 2.6. 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. □

As a consequence of this theorem, we get incomparability of answer sets.

Corollary 3.4. *Answer sets of a* DLP$^{\mathcal{A}}$ *program* $\mathcal{P}$ *are incomparable (w.r.t. set inclusion) among each other.*

Theorem 3.3 can be refined for DLP$^{\mathcal{A}}$ programs containing only monotone literals.

Theorem 3.5. *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 3.3, 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. [29].

Theorem 3.6. *Given a* DLP *program* $\mathcal{P}$*, an interpretation* I *is an answer set of* $\mathcal{P}$ *according to Definition* 2.6 *iff it is an answer set of* $\mathcal{P}$ *according to the standard definition via the classic Gelfond–Lifschitz transformation [12].*

Proof. ($\Rightarrow$): Assume that I is an answer set w.r.t. Definition 2.6, 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 decision problem: Given a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ and a standard ground atom A, is A true in all answer sets of $\mathcal{P}$?

For identifying fragments of DLP$^{\mathcal{A}}$, we use the notation LP$^{\mathcal{L}}_{\mathcal{A}}$, where $\mathcal{L} \subseteq \{\texttt{not}, \vee\}$ and $\mathcal{A} \subseteq \{M_s, M, A_s, A, N_s, N\}$.

Let $\mathcal{P} \in \text{LP}^{\mathcal{L}}_{\mathcal{A}}$. If $\texttt{not} \in \mathcal{L}$, then rules in $\mathcal{P}$ may contain negative literals. Likewise, if $\vee \in \mathcal{L}$, then rules in $\mathcal{P}$ may have disjunctive heads. If $M_s \in \mathcal{A}$ (resp., $A_s \in \mathcal{A}$, $N_s \in \mathcal{A}$), then $\mathcal{P}$ may contain monotone (resp. antimonotone, nonmonotone) aggregates, on which $\mathcal{P}$ is stratified. If $M \in \mathcal{A}$ (resp., $A \in \mathcal{A}$, $N \in \mathcal{A}$), then $\mathcal{P}$ may contain monotone (resp. antimonotone, nonmonotone) aggregates (on which $\mathcal{P}$ is not necessarily stratified). If a symbol is absent in a set, then the respective feature

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

Table 1
The complexity of cautious reasoning in ASP with aggregates (completeness results under logspace reductions).

	{}	{not}	{∨}	{not, ∨}	
{}	P	co-NP	co-NP	Π_2^P	1
$\{M\}$	P	co-NP	co-NP	Π_2^P	2
$\{A_s\}$	P	co-NP	Π_2^P	Π_2^P	3
$\{N_s\}$	P	co-NP	Π_2^P	Π_2^P	4
$\{M, A_s\}$	P	co-NP	Π_2^P	Π_2^P	5
$\{M, N_s\}$	P	co-NP	Π_2^P	Π_2^P	6
$\{A_s, N_s\}$	P	co-NP	Π_2^P	Π_2^P	7
$\{M, A_s, N_s\}$	P	co-NP	Π_2^P	Π_2^P	8
$\{A\}$	co-NP	co-NP	Π_2^P	Π_2^P	9
$\{M, A\}$	co-NP	co-NP	Π_2^P	Π_2^P	10
$\{A, N_s\}$	co-NP	co-NP	Π_2^P	Π_2^P	11
$\{M, A, N_s\}$	co-NP	co-NP	Π_2^P	Π_2^P	12
$\{N\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P	13
$\{M, N\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P	14
$\{A_s, N\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P	15
$\{M, A_s, N\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P	16
$\{A, N\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P	17
$\{M, A, N\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P	18
	1	2	3	4	

cannot occur in $\mathcal{P}$, unless another symbol is included which specifies a more general feature. For example, if $\mathcal{P} \in \mathrm{LP}^{\{\}}_{\{A\}}$, then antimonotone aggregates on which $\mathcal{P}$ is stratified may occur in $\mathcal{P}$ even if A_s is not specified.

For the technical results, we consider ground (i.e., variable-free) $\mathrm{DLP}^{\mathcal{A}}$ programs, and polynomial-time computable aggregate functions (note that all sample aggregate functions appearing in this paper fall into this class). However, in the overview we also provide a discussion on how results change when considering non-ground programs or aggregates which are harder to compute.

4.2. Overview of complexity results

Table 1 summarizes the complexity results derived in the next sections for various fragments $\mathrm{LP}^{\mathcal{L}}_{\mathcal{A}}$, where $\mathcal{L}$ is specified in columns and $\mathcal{A}$ in rows. Results for $\mathrm{LP}^{\mathcal{L}}_{\mathcal{A}}$, where $M_s \in \mathcal{L}$ have been omitted from Table 1 for readability, as they are equal to those of the respective fragment containing M instead of M_s.

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

The "cheapest" aggregates, from the viewpoint of complexity, are the monotone ones, the addition of which never causes any complexity increase, even for negation-free programs, and even for unstratified monotone aggregates.

The largest polynomial-time computable fragment is $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ (positive $\vee$-free programs), suggesting that also the stratified aggregates A_s and N_s are rather "cheap". Indeed, they behave similarly to stratified negation from the complexity viewpoint, and increase the complexity only in the case of positive disjunctive programs (from co-NP to Π_2^P).

Antimonotone aggregates (unstratified) behave like unstratified negation: In the positive $\vee$-free case their presence alone increase the complexity from P to co-NP. The complexity remains the same if monotone and stratified nonmonotone aggregates are added. The maximal co-NP-computable fragments are $\mathrm{LP}^{\{\mathtt{not}\}}_{\{M,A,N_s\}}$ and $\mathrm{LP}^{\{\vee\}}_{\{M\}}$.

The most "expensive" aggregates, from the viewpoint of complexity, are the nonmonotone ones: In the positive $\vee$-free case (definite Horn programs) they cause a big complexity jump from P to Π_2^P. For each language fragment containing nonmonotone aggregates we obtain Π_2^P-completeness. Intuitively, the reason is that nonmonotone aggregates can express properties which can be written using negation and disjunction in standard DLP.

Note that implemented ASP systems allow for expressing nonmonotone aggregates such as $1 < \#\mathtt{count}\{X : p(X)\} < 3$, which however, can be treated like a conjunction of a monotone and an antimonotone aggregate atom ($\#\mathtt{count}\{X : p(X)\} > 1$, $\#\mathtt{count}\{X : p(X)\} < 3$). The complexity of non-disjunctive programs with these constructs is therefore the same as for $\mathrm{LP}^{\{\mathtt{not}\}}_{\{M,A\}}$ (lower than $\mathrm{LP}^{\{\mathtt{not}\}}_{\{N\}}$). In [44], a broad class of nonmonotone aggregates, that can be rewritten as monotone and antimonotone aggregates in this style, is identified. Note, however, that sum aggregates (weight constraints) over positive and negative integers are nonmonotone and can in general not be decomposed into monotone and antimonotone aggregates.

The above complexity results give us valuable information about intertranslatability of different languages, having important implication also on the possibility to implement aggregates in existing reasoning engines. For instance, we know now that cautious reasoning on $\mathrm{LP}^{\{\mathtt{not}\}}_{\{M,A,N_s\}}$ can be efficiently translated to UNSAT (the complement of propositional satisfiability) or to cautious reasoning on non-disjunctive ASP; thus, arbitrary monotone, arbitrary antimonotone, and stratified nonmonotone aggregates can be implemented efficiently on top of SAT solvers and non-disjunctive ASP systems. On the other hand, since nonmonotonic aggregates (even without negation and disjunction) bring the complexity to Π^P_2, the existence of a polynomial translation from cautious reasoning with nonmonotonic aggregates to UNSAT cannot exist (unless the polynomial hierarchy collapses). Therefore, a rewriting to UNSAT is not viable to implement nonmonotone aggregates which require more powerful solvers.

As mentioned above, our results rely on the assumption that aggregate functions are computable in polynomial time. If one were to allow computationally more expensive aggregates, complexity would rise according to the complexity of additional oracles, which are needed to compute the truth value of an aggregate.

We end this overview by briefly addressing the complexity of non-ground programs. When considering data-complexity (i.e. a program $\mathcal{P}$ is fixed, while the input consists only of facts), the results are as for propositional programs. If, however, one considers program complexity (i.e. a program $\mathcal{P}$ is given as input), complexity rises in a similar manner as for aggregate-free programs. A non-ground program $\mathcal{P}$ can be reduced, by naive instantiation, to a ground instance of the problem. In the general case, where $\mathcal{P}$ is given in the input, the size of the grounding $\mathit{Ground}(\mathcal{P})$ is single exponential in the size of $\mathcal{P}$. Informally, the complexity of Cautious Reasoning increases accordingly by one exponential, from P to EXPTIME, co-NP to co-NEXPTIME, Π^P_2 to co-NEXPTIME$^{\mathrm{NP}}$. For aggregate-free programs complexity results in the non-ground case are reported in [45]. For the other fragments, the results can be derived using complexity upgrading techniques as presented in [15,46].

4.3. Proofs of hardness results

In this section, we will provide the proofs for all hardness results of Table 1.

4.3.1. Non-disjunctive programs

All P-hardness results in Table 1 (rows 1–8 in column 1) follow directly from the well-known result that (positive) propositional logic programming is P-hard [45].

An important observation is that negation can be simulated by antimonotone aggregates. It is therefore possible to turn aggregate-free programs with negation into corresponding positive programs with aggregates. Let us first define how this simulation can be achieved.

Definition 4.1. Given a program $\mathcal{P} \in \mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{\}}$, let $\Gamma(\mathcal{P})$ be the DLP$^{\mathcal{A}}$ program, which is obtained by replacing each negative literal $\mathtt{not}\ a$ in $\mathcal{P}$ by $\#\mathtt{count}\{\langle \epsilon : a\rangle\} < 1$, where ϵ is an arbitrary constant.

We can show that an aggregate-free program and its transformed version are equivalent.

Lemma 4.1. *Each program $\mathcal{P} \in \mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{\}}$ can be transformed into an equivalent program $\Gamma(\mathcal{P}) \in \mathrm{LP}^{\{\vee\}}_{\{A\}}$ with aggregate literals (all of which are antimonotone). If $\mathcal{P}$ is negation-stratified, then $\Gamma(\mathcal{P}) \in \mathrm{LP}^{\{\vee\}}_{\{A_s\}}$.*

Proof. Note that for any interpretation I, $\mathtt{not}\ a$ 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 3.6, our answer sets semantics (as in Definition 2.6) 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 $\mathcal{P}$ is negation-stratified, then there exists a level mapping, such that predicates in negative literals map to an ordinal which is less than the ordinal any head atom maps to. The same level-mapping can be used for showing that $\Gamma(\mathcal{P})$ is aggregate-stratified on all of its aggregate literals. □

Moreover, we can show that this transformation has a very low computational cost.

Lemma 4.2. *Let $\mathcal{P} \in \mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{\}}$. 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. □

As a consequence of these lemmata, we obtain hardness for positive non-disjunctive programs containing antimonotone aggregates.

Theorem 4.3. *Cautious reasoning over* $\mathrm{LP}^{\{\}}_{\{A\}}$ *programs is co-NP-hard.*

Proof. Follows from co-NP-hardness of cautious reasoning for positive disjunctive aggregate-free programs (programs in $\mathrm{LP}^{\{\vee\}}_{\{\}}$), see Theorem 6.1 in [45], together with Lemmata 4.1 and 4.2. □

Whenever one allows for nonmonotone aggregates in positive, non-disjunctive programs, cautious reasoning becomes harder by one level in the polynomial hierarchy.

Theorem 4.4. *Cautious reasoning over* $\mathrm{LP}^{\{\}}_{\{N\}}$ *programs is* Π^P_2*-hard.*

Proof. We provide a reduction from deciding the validity of a quantified boolean formula (2QBF) $\Psi = \forall x_1, \ldots, x_m \exists y_1, \ldots, y_n E$. Without loss of generality, we assume that E is a propositional formula in 3CNF format, over precisely the variables $x_1, \ldots, x_m, y_1, \ldots, y_n$. Deciding if such a Ψ is valid is still Π^P_2-hard [47]. Observe that Ψ is equivalent to $\neg\Psi'$, where $\Psi' = \exists x_1, \ldots, x_m \forall y_1, \ldots, y_n E'$, and E' is a 3DNF equivalent to $\neg E$, where every literal has reversed polarity w.r.t. E and conjunctions and disjunctions are inverted. Clearly, Ψ' is efficiently constructable from Ψ, and we have that Ψ is valid if and only if Ψ' is invalid. To prove the theorem, we construct an $\mathrm{LP}^{\{\}}_{\{N\}}$ program Π^Ψ which cautiously entails an atom w if and only if Ψ' is invalid (i.e., w is a cautious consequence of Π^Ψ if and only if Ψ is valid).

Let $E' = (l_{1,1} \wedge l_{1,2} \wedge l_{1,3}) \vee \cdots \vee (l_{k,1} \wedge l_{k,2} \wedge l_{k,3})$, we define the $\mathrm{LP}^{\{\}}_{\{N\}}$ program Π^Ψ as follows:

$$r_1 : t(x_i, 1) : \text{-}\#\mathtt{sum}\{\langle 1 : t(x_i, 1)\rangle, \langle -1 : t(x_i, -1)\rangle\} \geqslant 0, \quad i \in \{1, \ldots, m\}.$$
$$r_2 : t(x_i, -1) : \text{-}\#\mathtt{sum}\{\langle 1 : t(x_i, 1)\rangle, \langle -1 : t(x_i, -1)\rangle\} \leqslant 0, \quad i \in \{1, \ldots, m\}.$$
$$r_3 : t(y_i, 1) : \text{-}\#\mathtt{sum}\{\langle 1 : t(y_i, 1)\rangle, \langle -1 : t(y_i, -1)\rangle\} \geqslant 0, \quad i \in \{1, \ldots, n\}.$$
$$r_4 : t(y_i, -1) : \text{-}\#\mathtt{sum}\{\langle 1 : t(y_i, 1)\rangle, \langle -1 : t(y_i, -1)\rangle\} \leqslant 0, \quad i \in \{1, \ldots, n\}.$$
$$r_5 : t(y_i, 1) : \text{-} \mathit{sat}\, E'(1), \quad i \in \{1, \ldots, n\}.$$
$$r_6 : t(y_i, -1) : \text{-} \mathit{sat}\, E'(1), \quad i \in \{1, \ldots, n\}.$$
$$r_7 : \mathit{sat}\, E'(1) : \text{-} \mu(l_{i,1}), \mu(l_{i,2}), \mu(l_{i,3}), \quad i \in \{1, \ldots, k\}.$$
$$r_8 : w : \text{-}\#\mathtt{sum}\{\langle 1 : \mathit{sat}\, E'(1)\rangle, \langle -1 : \mathit{sat}\, E'(-1)\rangle\} \leqslant 0, \quad i \in \{1, \ldots, k\}.$$

where $\mu(l)$ is $t(a, 1)$ if $l = a$ is positive, and $\mu(l)$ is $t(a, -1)$ if $l = \neg a$ is negative. Intuitively, for each propositional variable a appearing in E', there are two atoms in Π^Ψ, namely $t(a, 1)$ and $t(a, -1)$, representing, respectively, the truth and the falsity of a. Atom $\mathit{sat}\, E'(1)$ is derivable from a rule $\mathit{sat}\, E'(1) : \text{-} \mu(l_{i,1}), \mu(l_{i,2}), \mu(l_{i,3})$ in Π^Ψ if the corresponding clause $(l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$ is true in E'.

We claim that w is a cautious consequence of Π^Ψ if and only if Ψ is valid. We can equivalently prove that $\mathit{sat}\, E'(1)$ is a brave consequence of Π^Ψ if and only if Ψ' is valid, since we have that: (1) w a cautious consequence of Π^Ψ if and only if $\mathit{sat}\, E'(1)$ is not a brave consequence of Π^Ψ (note that $\mathit{sat}\, E'(-1)$ is false in every answer set and, under answer set semantics, rule r_8 is then equivalent to $w : \text{-}\,\mathtt{not}\ \mathit{sat}\, E'(1)$), and (2) Ψ is valid if and only if Ψ' is invalid.

Thus, we next show that Π^Ψ has an answer set containing $\mathit{sat}\, E'(1)$ if and only if Ψ' is valid.

Assume first that Π^Ψ has an answer set A containing $\mathit{sat}\, E'(1)$. Observe that A contains exactly one of $t(x_i, 1)$ or $t(x_i, -1)$ for each $1 \leqslant i \leqslant m$ (if none held for some i, a rule would not be satisfied, if both held, A would not be a minimal model of the reduct). Therefore A encodes a truth assignment φ for $x_1, \ldots, x_m$ ($\varphi(x_i) = \mathit{true}$ if $t(x_i, 1) \in A$; $\varphi(x_i) = \mathit{false}$ if $t(x_i, -1) \in A$). Furthermore, A must contain both $t(y_i, 1)$ and $t(y_i, -1)$ for each $1 \leqslant i \leqslant n$, otherwise some rules of r_5 and r_6 would be unsatisfied w.r.t. A (as the body is true w.r.t. A which contains $\mathit{sat}\, E'(1)$). Since A is a minimal model of Π^{Ψ^A}, it follows that no A', which contains an encoding of φ and an arbitrary truth assignment for $y_1, \ldots, y_n$ but not $\mathit{sat}\, E'(1)$, is a model of Π^{Ψ^A}. So there must be at least one of the class of rules r_7 in Π^Ψ such that each body literal is in A' (thus forcing $\mathit{sat}\, E'(1)$). This in turn means that each extension of φ to $y_1, \ldots, y_n$ satisfies E' and thus that Ψ' is valid.

Assume now that Ψ' is valid, so there exists a truth assignment φ for $x_1, \ldots, x_m$ such that for each extension of φ to $y_1, \ldots, y_n$, E' is satisfied. Let I be the interpretation containing the encoding of φ, i.e. $t(x_i, 1)$ iff x_i is assigned true in φ and $t(x_i, -1)$ iff x_i is assigned false in φ, in addition $t(y_i, 1)$, $t(y_i, -1)$ for each $1 \leqslant i \leqslant n$ and $\mathit{sat}\, E'(1)$ (and nothing else). Π^{Ψ^I} contains all rules of Π^Ψ except

$$\mathit{sat}\, E'(1) : \text{-}\#\mathtt{sum}\{\langle 1 : \mathit{sat}\, E'(1)\rangle, \langle -1 : \mathit{sat}\, E'(-1)\rangle\} \leqslant 0.$$

Interpretation I is clearly a model of Π^{Ψ^I}. To prove its minimality, assume that a model $I' \subset I$ exists. It must contain the encoding of φ in order to satisfy the first two groups of rules (r_1 and r_2). Furthermore, I' must contain at least an encoding of a truth assignment for $y_1, \ldots, y_n$ in order to satisfy the third and fourth groups of rules. Then, since E' is satisfied by any such truth assignment, also $sat\,E'(1)$ must be in I' in order to satisfy all of the group of rules r_7. However, that means that all of $t(y_i, 1)$, $t(y_i, -1)$ for $1 \leqslant i \leqslant n$ must be in I' in order to satisfy the groups of rules r_5 and r_6. So $I' = I$, contradicting $I' \subset I$, and I is therefore an answer set of Π^{Ψ} (and clearly contains $sat\,E'(1)$). □

We note that a related result — deciding whether an answer set exists for a positive, non-disjunctive program with weight constraints over possibly negative integers is Σ_2^P-complete — has been shown in [37]. Weight constraints can be monotone, antimonotone, or nonmonotone aggregate atoms.

Leveraging results in the literature, we get hardness proofs for all fields for non-disjunctive programs in Table 1.

Theorem 4.5. *All fields in column* 1 *and all fields in column* 2 *of Table* 1 *states the respective hardness of cautious reasoning for the corresponding fragment of* $\mathrm{DLP}^{\mathcal{A}}$.

Proof. P-hardness results for the fields in rows 1 to 8 in column 1 follow from the fact that cautious reasoning over $\mathrm{LP}^{\{\}}_{\{\}}$ programs is P-hard [45] and that all corresponding languages are supersets of $\mathrm{LP}^{\{\}}_{\{\}}$. co-NP-hardness for the fields in rows 9 to 12 in column 1 stem from Theorem 4.3, as all corresponding languages are supersets of $\mathrm{LP}^{\{\}}_{\{A\}}$. The co-NP-hardness for the fields in rows 1 to 12 in column 2 are based on Theorem 6.7 in [48], which states that cautious reasoning over $\mathrm{LP}^{\{\mathtt{not}\}}_{\{\}}$ is co-NP-hard. All languages corresponding to the fields are supersets of $\mathrm{LP}^{\{\mathtt{not}\}}_{\{\}}$. All Π_2^P-hardness results for the fields in rows 13 to 18 in columns 1 and 2 are backed by Theorem 4.4, and the fact that all corresponding languages are supersets of $\mathrm{LP}^{\{\}}_{\{N\}}$. □

4.3.2. Disjunctive programs

Exploiting Lemma 4.1, which says that any aggregate-free program with negation can be transformed to an equivalent program with antimonotone aggregates, converting negation-stratification to aggregate-stratification, we can show Π_2^P-hardness for cautious reasoning over $\mathrm{LP}^{\{\vee\}}_{\{A_s\}}$ programs.

Theorem 4.6. *Cautious reasoning over* $\mathrm{LP}^{\{\vee\}}_{\{A_s\}}$ *programs is* Π_2^P*-hard.*

Proof. Follows from Π_2^P-hardness of cautious reasoning on standard literal queries for positive disjunctive aggregate-free ($\mathrm{LP}^{\{\vee\}}_{\{\}}$) programs, see Theorem 36 of [49]. Given such a program $\mathcal{P}$ and a literal l (of the form a or $\mathtt{not}\ a$, where a is a standard ground atom), let $\mathcal{P}' = \mathcal{P} \cup \{q : \text{-}l.\}$, where q is a ground atom that does not occur in $\mathcal{P}$. Obviously, $\mathcal{P}' \in \mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{\}}$ is negation-stratified, and cautious reasoning on q over $\mathcal{P}'$ is equivalent to cautious reasoning on l over $\mathcal{P}$. Together with Lemmata 4.1 and 4.2, the result follows. □

Next, we note that any program containing only stratified antimonotone aggregates can be transformed into an equivalent program containing only stratified nonmonotone aggregates.

Lemma 4.7. *Each* $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{A_s\}}$ *program can be transformed into an equivalent* $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{N_s\}}$ *program.*

Proof. W.l.o.g. we will consider a ground program $\mathcal{P}$. We transform each antimonotone aggregate literal l containing the aggregate atom $f(S) \circ k$ to l' containing $f^l(S') \circ k$. We introduce three fresh constants τ, ϵ, and ν and a new predicate symbol Π. 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 $\langle \tau : \Pi(\tau)\rangle$, $\langle \epsilon : \Pi(\epsilon)\rangle$, and $\langle \nu : \Pi(\nu)\rangle$ to the ground set S. The transformed program $\mathcal{P}'$ contains only nonmonotone aggregates, all of which are stratified on $\mathcal{P}$, and is clearly equivalent to $\mathcal{P}$. □

As a consequence, Π_2^P-hardness holds also for $\mathrm{LP}^{\{\vee\}}_{\{N_s\}}$ programs.

Corollary 4.8. *Cautious reasoning over* $\mathrm{LP}^{\{\vee\}}_{\{N_s\}}$ *programs is* Π_2^P*-hard.*

Proof. Follows directly from Theorem 4.6 and Lemma 4.7. □

These results, together with results from the literature, are sufficient to show all hardness results in columns 3 and 4 in Table 1.

Theorem 4.9. *Each field in columns* 3 *and* 4 *of Table* 1 *states the respective hardness of cautious reasoning for the corresponding fragment of* DLP$^{\mathcal{A}}$.

Proof. co-NP-hardness for the fields in rows 1 and 2 in column 3 rely on Theorem 6.1 of [45], which states that cautious reasoning over $\mathrm{LP}^{\{\vee\}}_{\{\}}$ programs is co-NP-hard, and the fact that $\mathrm{LP}^{\{\vee\}}_{\{\}} \subseteq \mathrm{LP}^{\{\vee\}}_{\{M\}}$. Π^P_2-hardness for the fields in rows 3 to 18 in column 3 follow from Theorem 4.6 and Corollary 4.8 and the fact that all corresponding languages are supersets of $\mathrm{LP}^{\{\vee\}}_{\{A_s\}}$ or $\mathrm{LP}^{\{\vee\}}_{\{N_s\}}$. Π^P_2-hardness for all fields in column 4 follows from Theorem 6.2 in [45], which states that cautious reasoning over $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{\}}$ is Π^P_2-hard, and the fact that all corresponding languages are supersets of $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{\}}$. □

In total, we have proved all hardness results for Table 1.

4.4. Proofs of membership results

For the membership proofs, we will go in the reverse order, and first prove results for richer languages, which cover also several results for sublanguages.

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

Lemma 4.10. *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 $L = f(T) \circ k$. 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. $\circ$.

Computing the valuation of a ground set T only requires scanning each element $\langle t_1, \ldots, t_n : \mathit{Conj}\rangle$ 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. □

4.4.1. Disjunctive programs

Let us first focus on the full language. Let us first show that the problem of answer set checking is in co-NP.

Lemma 4.11. *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. □

Using this result, we are able to give a "guess and check" algorithm for proving membership in Π^P_2.

Theorem 4.12. *Cautious reasoning over* $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{M,A,N\}}$ *programs is in* Π^P_2.

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 4.11. The problem therefore lies in Π^P_2. □

Concerning disjunctive programs, for most fragments cautious reasoning is in Π^P_2, with two exceptions which are in co-NP. The reason is that for the respective classes it is sufficient to look at an arbitrary model, rather than an answer set or a minimal model.

Lemma 4.13. *Let* $\mathcal{P}$ *be an* $\mathrm{LP}^{\{\vee\}}_{\{M\}}$ *program, 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.*[10]

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 3.5, M' is an answer set of $\mathcal{P}$. Therefore, A is not a cautious consequence of $\mathcal{P}$.

[10] Note that M can be *any* model, possibly non-minimal, 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 Proposition 3.1, M is also a model of $\mathcal{P}$. □

This lemma allows us to prove co-NP-membership for cautious reasoning over these programs.

Theorem 4.14. *Cautious reasoning over* $\mathrm{LP}^{\{\vee\}}_{\{M\}}$ *programs is in co-NP.*

Proof. By Lemma 4.13 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. □

These results are sufficient to show all hardness results in columns 3 and 4 in Table 1.

Theorem 4.15. *Each field in columns* 3 *and* 4 *of Table* 1 *states the respective membership of cautious reasoning for the corresponding fragment of* $\mathrm{DLP}^{\mathcal{A}}$.

Proof. Membership in Π^P_2 for all the fields in column 4 and fields in rows 3 to 18 of column 3 follow from Theorem 4.12, because all corresponding languages are subsets of $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{M,A,N\}}$. Membership in co-NP for the fields in rows 1 and 2 of column 3 follow from Theorem 4.14 and the fact that $\mathrm{LP}^{\{\vee\}}_{\{\}} \subseteq \mathrm{LP}^{\{\vee\}}_{\{M\}}$. □

4.4.2. Non-disjunctive programs

Π^P_2-memberships for non-disjunctive programs already follow from the respective result for disjunctive programs, and it remains to show co-NP- and P-memberships.

Let us first consider the less complex language $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$. We can show that programs in this fragment have either one or no answer sets, which can be computed efficiently.

Lemma 4.16. *An* $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ *program has at most one answer set and the answer sets of an* $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ *program can be computed in polynomial time.*

Proof. For an $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ program $\mathcal{P}$, let us define an operator $\mathbb{T}_{\mathcal{P}}$ on interpretations of $\mathcal{P}$ as follows: $\mathbb{T}_{\mathcal{P}}(I) = \{h \mid r \in \mathcal{P}, I \models B(r), h \in H(r)\}$. Furthermore, given an interpretation I, let the sequence $\{\mathbb{T}^n_{\mathcal{P}}(I)\}_{n\in\mathcal{N}}$ be defined as $\mathbb{T}^0_{\mathcal{P}}(I) = I$ and $\mathbb{T}^i_{\mathcal{P}} = \mathbb{T}_{\mathcal{P}}(\mathbb{T}^{i-1}_{\mathcal{P}}(I))$ for $i > 0$. Since $\mathbb{T}_{\mathcal{P}}$ is monotone and the number of interpretations for $\mathcal{P}$ is finite, the sequence reaches a fixpoint $\mathbb{T}^\infty_{\mathcal{P}}(I)$.

Consider a level mapping $\|\ \|$ such that for each rule $r \in \mathcal{P}$, for which $H(r) = \{h\}$ and an antimonotone or nonmonotone aggregate literal $A \in B(r)$, it holds for each predicate p nested in A that $\|p\| < \|p'\|$, where p' is the predicate of h. Moreover, $\|p\| \leqslant \|p'\|$ holds for any pair of predicates p and p' such that p' occurs in the head and p in the body of a rule. Without loss of generality, we assume the co-domain of $\|\|$ to be $0, \ldots, n$.

Based on $\|\ \|$, we define a partition $\mathcal{P}_0, \ldots, \mathcal{P}_n, \mathcal{P}_{constr}$ of $\mathcal{P}$ (where n is the maximum of the co-domain of $\|\ \|$ – since $\mathcal{P}$ is a finite, this is an integer) as follows: $\mathcal{P}_i = \{r \mid r \in \mathcal{P}, H(r) = \{h\}, \|Pred(h)\| = i\}$, $\mathcal{P}_{constr} = \{r \mid r \in \mathcal{P}, H(r) = \emptyset\}$. Furthermore, we define $FP^0_{\mathcal{P}} = \mathbb{T}^\infty_{\mathcal{P}_0}(\emptyset)$ and $FP^i_{\mathcal{P}} = \mathbb{T}^\infty_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}})$ for $0 < i \leqslant n$, and let $FP_{\mathcal{P}} = FP^n_{\mathcal{P}}$. If $FP_{\mathcal{P}}$ is a model of $\mathcal{P}_{constr}$, let $FM_{\mathcal{P}} = \{FP_{\mathcal{P}}\}$, otherwise $FM_{\mathcal{P}} = \emptyset$.

In the sequel we will use the shorthand $H(\mathcal{P}) = \{h \mid \exists r \in \mathcal{P}\colon\ h \in H(r)\}$ to denote the set of head atoms of a program.

We next show by induction that $FP_{\mathcal{P}} = A$ for each answer set A of $\mathcal{P}$. The base is $FP^0_{\mathcal{P}} \cap H(\mathcal{P}_0) = A \cap H(\mathcal{P}_0)$ for each answer set A of $\mathcal{P}$.

To prove $FP^0_{\mathcal{P}} \cap H(\mathcal{P}_0) \subseteq A \cap H(\mathcal{P}_0)$, we use another induction over $\mathbb{T}^i_{\mathcal{P}_0}(\emptyset)$. The base here is $\mathbb{T}^0_{\mathcal{P}_i}(\emptyset) = \emptyset \subseteq A$ for each answer set A of $\mathcal{P}$. Then, assuming that $S \subseteq A$ for each answer set A of $\mathcal{P}$, we can show that $\mathbb{T}_{\mathcal{P}_0}(S) \subseteq A$ for each answer set A of $\mathcal{P}$: Each rule $r \in \mathcal{P}_0$ is also in $\mathcal{P}$ and since A is a model by Proposition 3.1, whenever $S \models b$ for all $b \in B(r)$, then also for any answer set A, $A \models b$, as $B(r)$ may not contain antimonotone or nonmonotone aggregate literals, otherwise $\|p\| < 0$ for some predicate in such an aggregate would hold. Since $H(r) = \{h\}$, h must be contained in each answer set. It follows that $FP^0_{\mathcal{P}} = \mathbb{T}^\infty_{\mathcal{P}_0} \subseteq A$. It is easy to see that $FP^0_{\mathcal{P}} \subseteq H(\mathcal{P}_0)$, so $FP^0_{\mathcal{P}} \cap H(\mathcal{P}_0) \subseteq A \cap H(\mathcal{P}_0)$.

Now assume that $X = (A \cap H(\mathcal{P}_0)) \setminus (FP^0_{\mathcal{P}} \cap H(\mathcal{P}_0)) \neq \emptyset$. We show that then $A \setminus X$ is a model of $\mathcal{P}^A$, contradicting the assumption that A is an answer set. Each rule in $\mathcal{P}^A \cap \mathcal{P}_0$ is clearly satisfied by $A \setminus X$, because it is satisfied by $FP^0_{\mathcal{P}}$. Now recall that each rule r in $\mathcal{P}^A \setminus \mathcal{P}_0$ has a true body w.r.t. A, which is either true or false w.r.t. $A \setminus X$. Since $H(r) \cap X = \emptyset$ (because $X \subseteq H(\mathcal{P}_0)$ and by the definition of the partition $H(\mathcal{P}_0) \cap H(\mathcal{P} \setminus \mathcal{P}_0) = \emptyset$), r is also satisfied by $A \setminus X$. Therefore A is not an answer set of $\mathcal{P}$ if $X \neq \emptyset$, and so $FP^0_{\mathcal{P}} \cap H(\mathcal{P}_0) \supseteq A \cap H(\mathcal{P}_0)$. We have shown the base of the induction, $FP^0_{\mathcal{P}} \cap H(\mathcal{P}_0) = A \cap H(\mathcal{P}_0)$.

For the inductive step, we assume $FP^k_{\mathcal{P}} \cap H(\mathcal{P}_k) = A \cap H(\mathcal{P}_k)$ holds for all $k < i, i > 0$ and each answer set A. In order to show $FP^i_{\mathcal{P}} \cap H(\mathcal{P}_i) = A \cap H(\mathcal{P}_i)$, we use yet another induction over $\mathbb{T}^j_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}})$. The base is $\mathbb{T}^0_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}}) = FP^{i-1}_{\mathcal{P}} \subseteq A$ for each answer set A, which holds by the inductive hypothesis of the "larger" induction. Now, we assume that $\mathbb{T}^j_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}}) \subseteq A$ holds for each answer set, and show that $\mathbb{T}_{p_i}(\mathbb{T}^j_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}})) \subseteq A$ holds for each answer set. We observe that each rule $r \in \mathcal{P}_i$ is also in $\mathcal{P}$ and since A is a model by Proposition 3.1, whenever $\mathbb{T}^j_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}}) \models b$ for all $b \in B(r)$, then also for any answer set A, $A \models b$, because the only antimonotone or nonmonotone literals are aggregates which, however, contain only atoms formed by predicates p, for which $\|p\| < i$. Any of these atoms are however in $H(\mathcal{P}_k)$ for $k < i$ and so by the inductive hypothesis (of the "larger" induction), $\mathbb{T}^j_{\mathcal{P}_i}(FP^{i-1}_{\mathcal{P}}) \cap H(\mathcal{P}_k) = A \cap H(\mathcal{P}_k)$. In total, we get $FP^i_{\mathcal{P}} = \mathbb{T}^\infty_{\mathcal{P}_i} \subseteq A$.

It remains to show that $FP^i_{\mathcal{P}} \cap H(\mathcal{P}_i) \supseteq A \cap H(\mathcal{P}_i)$. Similar to the base case of the "larger" induction, we assume $X = (A \cap H(\mathcal{P}_i)) \setminus (FP^i_{\mathcal{P}} \cap H(\mathcal{P}_i)) \neq \emptyset$. We show that then $A \setminus X$ is a model of $\mathcal{P}^A$, contradicting the assumption that A is an answer set. Each rule in $\mathcal{P}^A \cap \mathcal{P}_i$ is clearly satisfied by $A \setminus X$, because it is satisfied by $FP^i_{\mathcal{P}}$. Now recall that each rule r in $\mathcal{P}^A \setminus \mathcal{P}_i$ has a true body w.r.t. A, which is either true or false w.r.t. $A \setminus X$. Since $H(r) \cap X = \emptyset$ (because $X \subseteq H(\mathcal{P}_i)$ and by the definition of the partition $H(\mathcal{P}_i) \cap H(\mathcal{P} \setminus \mathcal{P}_i) = \emptyset$), r is also satisfied by $A \setminus X$. Therefore A is not an answer set of $\mathcal{P}$ if $X \neq \emptyset$, and so $FP^i_{\mathcal{P}} \cap H(\mathcal{P}_i) \supseteq A \cap H(\mathcal{P}_i)$. We have shown the step of the induction, $FP^i_{\mathcal{P}} \cap H(\mathcal{P}_i) = A \cap H(\mathcal{P}_i)$ for each answer set A.

In total, for $FP_{\mathcal{P}}$ we have $FP_{\mathcal{P}} \cap (\bigcup_{i=1}^n H(\mathcal{P}_i)) = A \cap (\bigcup_{i=1}^n H(\mathcal{P}_i))$ for each answer set A of $\mathcal{P}$. It is easy to see that each answer set of $\mathcal{P}$ is also an answer set of $(\bigcup_{i=1}^n H(\mathcal{P}_i)) = \mathcal{P} \setminus \mathcal{P}_{constr}$. Therefore, for each answer set A of $\mathcal{P}$, we know that $A = FP_{\mathcal{P}}$. It follows that $\mathcal{P}$ has at most one answer set.

Moreover, note that any rules in $\mathcal{P}_{constr}$ can only be satisfied if one of its body literals is false (as the heads are empty). Now since $FP_{\mathcal{P}}$ is an answer set of $\mathcal{P} \setminus \mathcal{P}_{constr}$, it is a minimal model of $(\mathcal{P} \setminus \mathcal{P}_{constr})^{FP_{\mathcal{P}}}$. If $FP_{\mathcal{P}}$ satisfies all rules in $\mathcal{P}_{constr}$, then $(\mathcal{P} \setminus \mathcal{P}_{constr})^{FP_{\mathcal{P}}} = \mathcal{P}^{FP_{\mathcal{P}}}$, and $FP_{\mathcal{P}}$ is an answer set of $\mathcal{P}$. If any rule of $\mathcal{P}_{constr}$ exists which is not satisfied by $FP_{\mathcal{P}}$, this rule also occurs in $\mathcal{P}^{FP_{\mathcal{P}}}$, and therefore $FP_{\mathcal{P}}$ cannot be a model of $\mathcal{P}^{FP_{\mathcal{P}}}$, and hence it cannot be an answer set of $\mathcal{P}$ in this case. In total, we get that $FM_{\mathcal{P}}$ is the set of answer sets for $\mathcal{P}$.

Computing $FP_{\mathcal{P}}$ and $FM_{\mathcal{P}}$ using $\mathbb{T}_{\mathcal{P}}$ is clearly feasible in polynomial time in the size of the program. □

Given that we can compute the set of answer sets in polynomial time and that the cardinality of this set is at most 1, cautious reasoning can be done easily over the computed answer sets.

Theorem 4.17. *Cautious reasoning over* $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ *is in* P.

Proof. This is a simple consequence of Lemma 4.16. We compute the set of answer sets in polynomial time. If it is empty, all atoms are a cautious consequence. If there is one answer set, check in polynomial time whether it contains the query atom. □

Let us now focus on the co-NP-memberships. For doing so, we will re-use the fact that answer sets $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ programs are computable in polynomial time. The point is that for checking whether an interpretation I is an answer set of an $\mathrm{LP}^{\{\mathrm{not}\}}_{\{M,A,N_s\}}$ program $\mathcal{P}$, we can form the reduct $\mathcal{P}^I$, which is also an $\mathrm{LP}^{\{\mathrm{not}\}}_{\{M,A,N_s\}}$ program. The crucial point is that for checking whether I is a minimal model of $\mathcal{P}^I$ (in which case it is an answer set), one can eliminate antimonotone literals from $\mathcal{P}^I$.

Lemma 4.18. *Given an* $\mathrm{LP}^{\{\mathrm{not}\}}_{\{M,A,N_s\}}$ *program* $\mathcal{P}$ *and an interpretation* $I \subseteq B_{\mathcal{P}}$, *I is a subset-minimal model of* $\mathcal{P}^I$ *iff it is a subset-minimal model of* $\Psi(\mathcal{P}^I)$, *which is derived from* $\mathcal{P}^I$ *by deleting all antimonotone literals.*

Proof. ($\Rightarrow$) If I is a minimal model of $\mathcal{P}^I$, it is obviously also a model of $\Psi(\mathcal{P}^I)$. Moreover, each interpretation $N \subset I$ is not a model of $\mathcal{P}^I$, so there is at least one rule $r \in \mathcal{P}^I$, for which $N \not\models r$, that is all body atoms are true w.r.t. N but all head atoms are false w.r.t. N. Now there is a rule $r' \in \Psi(\mathcal{P}^I)$ with $H(r) = H(r')$ and $B(r) \supseteq B(r')$. So also the body of r' is true w.r.t. N, and hence r' is not satisfied by N. As a consequence, N is not a model of $\Psi(\mathcal{P}^I)$, and therefore I is a minimal model of $\Psi(\mathcal{P}^I)$.

($\Leftarrow$) Let I be a minimal model of $\Psi(\mathcal{P}^I)$. We first note that no rule in $\mathcal{P}^I$ has a body literal which is false w.r.t. I by construction of $\mathcal{P}^I$, and therefore also no rule in $\Psi(\mathcal{P}^I)$ has a body literal which is false w.r.t. I. So for any rule in $\Psi(\mathcal{P}^I)$, all body literals are true w.r.t. I, and hence one of its head atoms is true w.r.t. I, since I is a model. Since each rule in $\Psi(\mathcal{P}^I)$ has a corresponding rule in $\mathcal{P}^I$ with equal head, and since no rule in $\mathcal{P}^I$ has a body literal which is false w.r.t. I, I is also a model of $\mathcal{P}^I$.

Now, consider an arbitrary interpretation $N \subset I$. N is not a model of $\Psi(\mathcal{P}^I)$, that is, there is a rule $r \in \Psi(\mathcal{P}^I)$ for which all body literals in r are true w.r.t. N, and all head atoms in r are false w.r.t. N. Now consider the corresponding rule $r' \in p^I$, for which $B(r) \subseteq B(r')$. By construction of p^I, all literals of r' are true w.r.t. I, and since each deleted body literal $\ell \in B(r') \setminus B(r)$ is an antimonotone literal (either a negative standard literal or an antimonotone aggregate literal), ℓ is also

true w.r.t. N. Hence, the body of r' is true w.r.t. N, and since $H(r') = H(r)$, each head atom of r' is false w.r.t. N. Hence r' is not satisfied and N is not a model of $\mathcal{P}^I$, and we obtain that I is a minimal model of $\mathcal{P}^I$. □

So answer set checking for an $\mathrm{LP}^{\{\mathtt{not}\}}_{\{M,A,N_s\}}$ program can be done by checking whether an interpretation is a minimal model for an $\mathrm{LP}^{\{\}}_{\{M,N_s\}}$ program, which in this case is equivalent to checking whether it is an answer set. We have already shown earlier that this task is polynomial.

Theorem 4.19. *Cautious reasoning over* $\mathrm{LP}^{\{\mathtt{not}\}}_{\{M,A,N_s\}}$ *is in co-NP.*

Proof. We guess an interpretation I, and check whether it is an answer set and does not contain the queried atom. The latter check is clearly polynomial. Answer set checking amounts to checking whether I is a subset-minimal model of $\mathcal{P}^I$. Because of Lemma 4.18, I is a subset-minimal model of $\mathcal{P}^I$ iff I is a subset-minimal model of $\Psi(\mathcal{P}^I)$, in which all negative standard and antimonotone aggregate literals have been deleted (this transformation is obviously polynomial). Because of Proposition 3.2, I is a subset-minimal model of $\mathcal{P}^I$ if I is an answer set of $\mathcal{P}^I$, hence if I is an answer set of $\Psi(\mathcal{P}^I)$. Now since $\Psi(\mathcal{P}^I) \in \mathrm{LP}^{\{\}}_{\{M,N_s\}} \subseteq \mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$ we know by Lemma 4.16 that its answer sets (at most one) are computable in polynomial time. So we can compute the set of minimal models of $\Psi(\mathcal{P}^I)$ in polynomial time. If it is empty, I is not an answer set; otherwise there is exactly one minimal model, and we check whether it is equal to I. If it is, I is an answer set, otherwise it is not. Checking whether I is an answer set is therefore feasible in polynomial time. □

We have therefore proved all membership results of Table 1 for non-disjunctive programs.

Theorem 4.20. *Each field in columns* 3 *and* 4 *of Table* 1 *states the respective membership of cautious reasoning for the corresponding fragment of* $\mathrm{DLP}^{\mathcal{A}}$.

Proof. Membership in Π^P_2 for all the fields in rows 13 to 18 in columns 1 and 2 follow from Theorem 4.12, because all corresponding languages are subsets of $\mathrm{LP}^{\{\mathtt{not},\vee\}}_{\{M,A,N\}}$. Membership in co-NP for the fields in rows 9 to 12 of column 1 and in rows 1 to 12 of column 2 are a consequence of Theorem 4.19, since all corresponding languages are subsets of $\mathrm{LP}^{\{\mathtt{not}\}}_{\{M,A,N_s\}}$. Finally, membership in P for the fields in rows 1 to 8 of column 1 are due to Theorem 4.17, since all corresponding languages are subsets of $\mathrm{LP}^{\{\}}_{\{M,A_s,N_s\}}$. □

5. Related work

There have been considerable efforts to define semantics for logic programs with aggregates. For a historical background, we refer to [50]. Here we will focus on work which has been proposed in the field of Answer Set Programming for defining semantics for recursive aggregates. Several of these works consider only monotone aggregates, such as [31,33,30]. We will not go into further details with respect to these approaches, as their focus is either on having aggregate atoms in rule heads (a feature which is absent in our framework) or on working out algebraic methods for disjunctive programs. Moreover, semantically, monotone aggregates in rule bodies are straightforward to handle, as they perfectly correspond to standard positive atoms in their behavior. We also note that most of the related works do not consider disjunctive programs. A thorough discussion of pros and cons for the various approaches for recursive aggregates has been given in [50,34,36].

The approaches of [25,27,28] basically all admit non-minimal answer sets. In particular, program P_1 of Example 1.2 would have $\emptyset$ and $\{p(a)\}$ as answer sets. As shown in Example 2.13 (also by Theorem 3.3), the semantics proposed in this paper only admits $\emptyset$, and always guarantees the minimality of answer sets. The work in [51] deals with the more abstract concept of generalized quantifiers, and the semantics therein also allows for non-minimal answer sets.

The approach of [43] 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 [29] and [52], in presence of negative weights or negative literals inside aggregates,[11] the semantics in [43] can lead to unintuitive results. For example, the program $\{a\,\mathtt{:-\#sum}\{\langle -1 : a\rangle\} \leqslant -1.\}$ should intuitively have only $\emptyset$ as an answer set, as $\{a\}$ would not be minimal and the truth of a is not founded. However, according to [43], both $\emptyset$ and $\{a\}$ are answer sets.[12] Our semantics only allows for $\emptyset$ as an answer set, according to the intuition. However, in [37] it has been shown that the semantics of [43] is equal to the answer set semantics as in Definition 2.6 on programs with `#sum` (respectively weight constraints) over positive integers. An extension to the approach of [43] has been presented in [32], which allows for arbitrary aggregates in non-disjunctive programs. A difference with respect to [43,32] is also that these languages allow for aggregate atoms in rule heads, which we do not consider in this paper.

[11] Note that while negative literals inside aggregates are not allowed in our framework, negative integers are allowed and correctly dealt with.

[12] Interestingly, `lparse` (version 1.0.17) and `smodels` (version 2.32), the software implementing the semantics of [43], computes only $\emptyset$.

A major contribution to the understanding of aggregates in ASP has been presented in [37]. The author provides a way to represent (ground) aggregates by means of propositional formulas, building on earlier work reported in [52]. Together with the reduct-based semantics for propositional formulas presented in [37] (which are called answer sets as well), this yields a semantics for programs with aggregates as well. In Theorem 3 of [37], Ferraris proves that this semantics coincides with the one presented in this paper in Definition 2.6 on what Ferraris refers to as *FLP-programs* (ground $\text{DLP}^{\mathcal{A}}$ programs in which aggregate atoms are not preceded by `not`).

It should be noted that the representation in [37] is done in a careful way in order to guarantee monolithic stability justification capabilities of aggregates. In particular, when forming the reduct with respect to an interpretation I as defined in [37], any formula representing an aggregate not satisfied by I will be completely replaced by $\bot$ (falsity), rendering the corresponding rule irrelevant in the reduct. On the other hand, a formula representing an aggregate satisfied by I will stay in the reduct as is. This behavior precisely coincides with the main motivation for the reduct of Definition 2.5, and distinguishes this approach from others, as discussed below.

However, there is a difference with respect to the semantics in [37] when negated aggregate atoms occur in the program. This is because in our work we treat the negation operator simply as a complement operator for aggregates, while in [37] it is treated as a negation-as-failure operator. The difference is best shown using an example.

Example 5.1. Given the program

$$r\colon\quad a\texttt{ :-not }\#\texttt{count}\big\{\langle 1:a\rangle\big\} < 1.$$

there is one answer set ($\emptyset$) with respect to Definition 2.6, while [37] would allow for two answer sets $\emptyset$ and $\{a\}$.

So in the presence of negated literals, the semantics of [37] allows non-minimal answer sets. Both ways of dealing with `not` in front of aggregates can be motivated: For our language it is seen as a shorthand for the complement of the aggregate, and the above rule is equivalent to:

$$r'\colon\quad a\texttt{ :-}\#\texttt{count}\big\{\langle 1:a\rangle\big\} \geqslant 1.$$

In [37], rule r is viewed as equivalent to

$$r''\colon\quad a\texttt{ :-not not }a.$$

which also has two answer sets $\emptyset$ and $\{a\}$ according to [53].

It is however notable that even though the language considered in [37] is very general and its semantics has been defined independently, without having the $\text{DLP}^{\mathcal{A}}$ language in mind, the two semantics coincide for the most part. We view this as a confirmation of the robustness of our semantics.

In [37], the author has also given some complexity results. In particular, he has shown that deciding whether a (non-disjunctive) program with weight constraints (a `#sum`-aggregate in our notation) has an answer set, is Σ_2^P-complete. This is strictly related to our result that cautious reasoning over a program in $\text{LP}^{\{\}}_{\{N\}}$ is Π_2^P-complete.

Recently, in [54] a language called RASPL-1 has been defined, which essentially allows for (possibly non-ground) counting aggregates. The semantics of this language is defined analogously to [37], but in this case by means of a representation as a first-order formula which is then interpreted using a semantics for arbitrary first-order formulas which has been presented in [55]. Also the semantics of RASPL-1 has been shown to coincide with Definition 2.6 on a large common language fragment; we refer to [54] for details.

We would furthermore like to point out that the reduct and the semantics defined in this paper has already spread in the scientific community and has been used in the work of others. Indeed one main advantage of our semantic definition in this respect is its generality. Being based on a definition of reduct, which does not refer to aggregates or special structures at all, it allows for defining the semantics of arbitrary linguistic extensions. Indeed, in [56,57] the authors use Definition 2.5 for defining a semantics for programs with higher order and externally defined atoms. This work is set in the context of reasoning in the Semantic Web (where "aggregates" involve querying ontologies, for example), and can be seen as a variant of our semantics for that framework.

However, there are also other suggestions for the semantics of programs with aggregates. Most representative of those, in [29,34], several semantics for non-disjunctive programs with aggregates have been defined, the closest one to the semantics in this paper being the $\tilde{\mathcal{D}}$*-stable semantics*. In [36,35] the notions of *fixpoint answer set* and *unfolding answer set* have been defined for non-disjunctive programs with aggregates, which, in [35], have been shown to be equivalent. Moreover, the $\tilde{\mathcal{D}}$-stable semantics and fixpoint answer sets are also equivalent, as shown in [36,35]. Also for the $\tilde{\mathcal{D}}$-stable semantics, minimality and coincidence with answer sets in the aggregate-free case is guaranteed. Another equivalent definition for programs with c-atoms (which are essentially extensional representations of aggregate atoms) has been given in [58].

In Theorem 4 in [36] and Proposition 8.1 in [34] it has been shown that any $\tilde{\mathcal{D}}$-stable model is also an answer set as defined in Definition 2.6. However, an answer set as defined in Definition 2.6 is not necessarily a $\tilde{\mathcal{D}}$-stable model, as noted in [36,34]. In his doctoral thesis [50], Pelov also defines various semantics for disjunctive programs with aggregates, among them one which is close to ours. However, the same differences as for the $\tilde{\mathcal{D}}$-stable model semantics surface.

To see these differences, let us consider Example 9 of [36].

Example 5.2. Given the program

$$p(1) \text{ :- } \#\mathtt{sum}\{X : p(X)\} \geqslant 0. \qquad p(1) \text{ :- } p(-1). \qquad p(-1) \text{ :- } p(1).$$

we obtain one answer set $\{p(1), p(-1)\}$ with respect to Definition 2.6, but no $\tilde{\mathcal{D}}$-stable model.

The authors of [36] argue that the program should be equivalent to the aggregate-free program

$$p(1) \text{ :- } \mathtt{not}\ p(-1). \qquad p(1) \text{ :- } \mathtt{not}\ p(1), \mathtt{not}\ p(-1). \qquad p(1) \text{ :- } p(1), \mathtt{not}\ p(-1).$$

$$p(1) \text{ :- } p(1). \qquad p(1) \text{ :- } p(1), p(-1). \qquad p(1) \text{ :- } p(-1). \qquad p(-1) \text{ :- } p(1).$$

Here, when forming the reduct w.r.t. $\{p(1), p(-1)\}$, the first three rules are deleted. This is against our intuition that any literal, and in particular aggregate literals are to be considered as a monolithic structure when verifying stability. Indeed, in this example only some part of the representation of the aggregate is retained in the reduct. This is a situation which cannot occur in our setting, any aggregate is either relevant in its entirety or has no effect at all. Interestingly, also the semantics of [37] shares precisely our view and yields the (unique) answer set $\{p(1), p(-1)\}$ on this program.

As this example shows, our approach is in line with the semantics of [37], and differs from [36,34] in the assumption how an aggregate literal may justify an answer set. We believe that both approaches can be motivated and the choice of the "right" semantics depends on how one interprets the justification capabilities of an aggregate. However, if one accepts our assumption that aggregates must serve as justifiers in a monolithic way, these other semantics do not behave in an intuitive way. Indeed, as shown in Example 5.2 it is unclear why one would allow only for some part of an aggregate to give stability to an answer set candidate. Moreover, our "monolithic" approach has the advantage to be generally applicable, since it is not specific to aggregates, but it depends only on basic satisfaction of the expressions in the language.

In [50], Pelov provides also a complexity analysis for reasoning tasks in the setting of the semantics proposed in that work. In particular, the problem of model existence is studied, which is related to the query answering problems studied in this work. Pelov does not differentiate among the types of literals as we do, but differentiates among the semantics defined and the evaluation complexity of the aggregate literals. Also [35] contains a similar analysis. The results are compatible to the ones derived in this paper, model existence being located on the first and second level of the polynomial hierarchy.

6. Conclusions

Concluding, we have proposed a declarative semantics for full ASP programs with arbitrary aggregates (DLP$^{\mathcal{A}}$ programs). This semantics generalizes the answer set semantics for standard ASP in a simple and elegant way, through a new definition of reduct which is simpler than the original one and treats negative literals, positive literals, and aggregates literals in a fully uniform manner. We have demonstrated that our semantics is endowed with desirable properties: it guarantees subset-minimality (and therefore the incomparability) of answer sets, and it coincides with the standard answer set semantics on aggregate-free programs. We have analyzed the computational complexity of the language in depth, drawing a full picture of the complexity of the ASP fragments where negation and/or disjunction are combined with different kinds of aggregates (monotone, antimonotone, nonmonotone, stratified). Importantly, we proved that aggregate literals do not increase the computational complexity of full (disjunctive) ASP programs in our approach; while they do increase the complexity of normal (non-disjunctive) programs up to Π_2^P. We have singled out, however, relevant classes of aggregates which do not cause any complexity overhead even for normal programs, and can be efficiently implemented in normal ASP systems.

Acknowledgements

This work has greatly benefited from interesting discussions with and comments of Paolo Ferraris, Michael Gelfond, Vladimir Lifschitz, Nikolay Pelov. We are also grateful to the competent comments and suggestions in the reviews. The work was partially supported by M.I.U.R. under projects "Potenziamento e Applicazioni della Programmazione Logica Disgiuntiva," "Sistemi basati sulla logica per la rappresentazione di conoscenza: estensioni e tecniche di ottimizzazione," and "tocai.it: Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet".

References

[1] W. Faber, N. Leone, G. Pfeifer, Recursive aggregates in disjunctive logic programs: Semantics and complexity, in: J.J. Alferes, J. Leite (Eds.), Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004), in: Lecture Notes in AI (LNAI), vol. 3229, Springer-Verlag, 2004, pp. 200–212.
[2] F. Calimeri, W. Faber, N. Leone, S. Perri, Declarative and computational properties of logic programs with aggregates, in: Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), 2005, pp. 406–411.
[3] J. McCarthy, Programs with common sense, in: Proceedings of the Teddington Conference on the Mechanization of Thought Processes, Her Majesty's Stationery Office, 1959, pp. 75–91.
[4] J. McCarthy, P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, vol. 4, Edinburgh University Press, 1969, pp. 463–502, reprinted in [59].
[5] M. Minsky, A framework for representing knowledge, in: P.H. Winston (Ed.), The Psychology of Computer Vision, McGraw–Hill, 1975, pp. 211–277.
[6] J. McCarthy, Circumscription — A form of non-monotonic reasoning, Artificial Intelligence 13 (1–2) (1980) 27–39.

[7] J. McCarthy, Applications of circumscription to formalizing common-sense knowledge, Artificial Intelligence 28 (1) (1986) 89–116.
[8] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1–2) (1980) 81–132.
[9] D.V. McDermott, J. Doyle, Non-monotonic logic I, Artificial Intelligence 13 (1–2) (1980) 41–72.
[10] D.V. McDermott, Non-monotonic logic II: Nonmonotonic modal theories, Journal of the ACM 29 (1) (1982) 33–57.
[11] R.C. Moore, Semantical considerations on nonmonotonic logic, Artificial Intelligence 25 (1) (1985) 75–94.
[12] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (1991) 365–385.
[13] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2003.
[14] J. Minker, On indefinite data bases and the closed world assumption, in: D.W. Loveland (Ed.), Proceedings of the 6th Conference on Automated Deduction (CADE '82), in: Lecture Notes in Computer Science, vol. 138, Springer, New York, 1982, pp. 292–308.
[15] T. Eiter, G. Gottlob, H. Mannila, Disjunctive DATALOG, ACM Transactions on Database Systems 22 (3) (1997) 364–418.
[16] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (2002) 181–234.
[17] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, J.-H. You, Unfolding partiality and disjunctions in stable model semantics, ACM Transactions on Computational Logic 7 (1) (2006) 1–37.
[18] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Transactions on Computational Logic 7 (3) (2006) 499–562.
[19] F. Lin, Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers, Artificial Intelligence 157 (12) (2004) 115–137.
[20] Y. Lierler, M. Maratea, Cmodels-2: SAT-based answer set solver enhanced to non-tight programs, in: V. Lifschitz, I. Niemelä (Eds.), Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7), in: Lecture Notes in AI (LNAI), vol. 2923, Springer, 2004, pp. 346–350.
[21] C. Anger, K. Konczak, T. Linke, NoMoRe: A system for non-monotonic reasoning, in: T. Eiter, W. Faber, M. Truszczyński (Eds.), Logic Programming and Nonmonotonic Reasoning — Proceedings of the 6th International Conference (LPNMR '01), Vienna, Austria, September 2001, in: Lecture Notes in AI (LNAI), vol. 2173, Springer-Verlag, 2001, pp. 406–410.
[22] C. Anger, M. Gebser, T. Linke, A. Neumann, T. Schaub, The nomore++ approach to answer set solving, in: G. Sutcliffe, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference (LPAR 2005), in: Lecture Notes in Computer Science, vol. 3835, Springer-Verlag, 2005, pp. 95–109.
[23] M. Gebser, B. Kaufmann, A. Neumann, T. Schaub, Conflict-driven answer set solving, in: Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), Morgan Kaufmann Publishers, 2007, pp. 386–392.
[24] I.S. Mumick, H. Pirahesh, R. Ramakrishnan, The magic of duplicates and aggregates, in: Proceedings of the 16th International Conference on Very Large Data Bases (VLDB '90), Morgan Kaufmann, 1990, pp. 264–277.
[25] D.B. Kemp, P.J. Stuckey, Semantics of logic programs with aggregates, in: V.A. Saraswat, K. Ueda (Eds.), Proceedings of the International Symposium on Logic Programming (ISLP '91), MIT Press, 1991, pp. 387–401.
[26] K.A. Ross, Y. Sagiv, Monotonic aggregation in deductive databases, Journal of Computer and System Sciences 54 (1) (1997) 79–97.
[27] M. Gelfond, Representing knowledge in A-Prolog, in: A.C. Kakas, F. Sadri (Eds.), Computational Logic. Logic Programming and Beyond, in: Lecture Notes in Computer Science, vol. 2408, Springer, 2002, pp. 413–451.
[28] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, G. Pfeifer, Aggregate functions in DLV, in: M. de Vos, A. Provetti (Eds.), Proceedings ASP03 — Answer Set Programming: Advances in Theory and Implementation, Messina, Italy, 2003, pp. 274–288, online at http://CEUR-WS.org/Vol-78/.
[29] N. Pelov, M. Denecker, M. Bruynooghe, Partial stable models for logic programs with aggregates, in: Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7), in: Lecture Notes in AI (LNAI), vol. 2923, Springer, 2004, pp. 207–219.
[30] N. Pelov, M. Truszczyński, Semantics of disjunctive programs with monotone aggregates — An operator-based approach, in: Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, BC, Canada, 2004, pp. 327–334.
[31] V.W. Marek, J.B. Remmel, On logic programs with cardinality constraints, in: S. Benferhat, E. Giunchiglia (Eds.), Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR 2002), Toulouse, France, 2002, pp. 219–228.
[32] V.W. Marek, J.B. Remmel, Set constraints in logic programming, in: V. Lifschitz, I. Niemelä (Eds.), Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7), in: Lecture Notes in AI (LNAI), vol. 2923, Springer, 2004, pp. 167–179.
[33] V.W. Marek, I. Niemelä, M. Truszczyński, Logic programming with monotone cardinality atom, in: V. Lifschitz, I. Niemelä (Eds.), Proceedings of the 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7), in: Lecture Notes in AI (LNAI), vol. 2923, Springer, 2004, pp. 154–166.
[34] N. Pelov, M. Denecker, M. Bruynooghe, Well-founded and stable semantics of logic programs with aggregates, Theory and Practice of Logic Programming 7 (3) (2007) 301–353.
[35] T.C. Son, E. Pontelli, A constructive semantic characterization of aggregates in ASP, Theory and Practice of Logic Programming 7 (2007) 355–375.
[36] T.C. Son, E. Pontelli, I. Elkabani, On logic programming with aggregates, Tech. Rep. NMSU-CS-2005-006, New Mexico State University, 2005.
[37] P. Ferraris, Answer sets for propositional theories, in: C. Baral, G. Greco, N. Leone, G. Terracina (Eds.), Logic Programming and Nonmonotonic Reasoning — 8th International Conference (LPNMR '05), Diamante, Italy, September 2005, in: Lecture Notes in Computer Science, vol. 3662, Springer-Verlag, 2005, pp. 119–131.
[38] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, G. Pfeifer, Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in DLV, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI) 2003, Morgan Kaufmann Publishers, Acapulco, Mexico, 2003, pp. 847–852.
[39] F. Lin, Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers, in: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI 2002), AAAI Press/MIT Press, Edmonton, Alberta, Canada, 2002.
[40] J.D. Ullman, Principles of Database and Knowledge Base Systems, Computer Science Press, 1989.
[41] K.R. Apt, H.A. Blair, A. Walker, Towards a theory of declarative knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann Publishers, Inc., Washington, DC, 1988, pp. 89–148.
[42] T.C. Przymusinski, On the declarative semantics of deductive databases and logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann Publishers, Inc., 1988, pp. 193–216.
[43] I. Niemelä, P. Simons, T. Soininen, Stable model semantics of weight constraint rules, in: M. Gelfond, N. Leone, G. Pfeifer (Eds.), Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '99), in: Lecture Notes in AI (LNAI), vol. 1730, Springer-Verlag, El Paso, Texas, USA, 1999, pp. 107–116.
[44] W. Faber, Decomposition of nonmonotone aggregates in logic programming, in: M. Fink, H. Tompits, S. Woltran (Eds.), Proceedings of the 20th Workshop on Logic Programming (WLP 2006), Vienna, Austria, 2006, pp. 164–171.
[45] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Computing Surveys 33 (3) (2001) 374–425.
[46] G. Gottlob, N. Leone, H. Veith, Succinctness as a source of expression complexity, Annals of Pure and Applied Logic 97 (13) (1999) 231–260.
[47] L.J. Stockmeyer, A.R. Meyer, Word problems requiring exponential time: Preliminary report, in: Conference Record of 5th Annual ACM Symposium on Theory of Computing (STOC '73), ACM Press, 1973, pp. 1–9.
[48] V.W. Marek, M. Truszczyński, Autoepistemic logic, Journal of the ACM 38 (3) (1991) 588–619.

[49] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Annals of Mathematics and Artificial Intelligence 15 (3/4) (1995) 289–323.
[50] N. Pelov, Semantics of logic programs with aggregates, Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, Apr. 2004.
[51] T. Eiter, G. Gottlob, H. Veith, Modular logic programming and generalized quantifiers, in: J. Dix, U. Furbach, A. Nerode (Eds.), Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97), in: Lecture Notes in Computer Science, vol. 1265, Springer, 1997, pp. 290–309.
[52] P. Ferraris, V. Lifschitz, Weight constraints as nested expressions, Theory and Practice of Logic Programming 5 (1–2) (2005) 45–74.
[53] V. Lifschitz, L.R. Tang, H. Turner, Nested expressions in logic programs, Annals of Mathematics and Artificial Intelligence 25 (34) (1999) 369–389.
[54] J. Lee, V. Lifschitz, R. Palla, A reductive semantics for counting and choice in answer set programming, in: D. Fox, C.P. Gomes (Eds.), Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI '08), AAAI Press, 2008, pp. 472–479.
[55] P. Ferraris, J. Lee, V. Lifschitz, A new perspective on stable models, in: Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), 2007, pp. 372–379.
[56] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, A uniform integration of higher-order reasoning and external evaluations in answer set programming, in: International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, UK, 2005, pp. 90–96.
[57] T. Eiter, G. Ianni, H. Tompits, R. Schindlauer, Effective integration of declarative rules with external evaluations for semantic web reasoning, in: Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), 2006, pp. 273–287.
[58] T.C. Son, E. Pontelli, P.H. Tu, Answer sets for logic programs with arbitrary abstract constraint atoms, Journal of Artificial Intelligence Research 29 (2007) 353–389.
[59] J. McCarthy, Formalization of Common Sense, papers by John McCarthy edited by V. Lifschitz, Ablex, 1990.