Decomposition of Nonmonotone Aggregates in Logic Programming*

Wolfgang Faber

Department of Mathematics, University of Calabria, 87030 Rende (CS), Italy faber@mat.unical.it

Abstract. Aggregates in answer set programming (ASP) have recently been studied quite intensively. The main focus of previous work has been on defining suitable semantics for programs with arbitrary, potentially recursive aggregates. By now, these efforts appear to be converging.
An interesting feature of the respective semantics is that for programs without disjunctive rules, the complexity of reasoning tasks increases with the presence of nonmonotone aggregate atoms. The reason is that with appropriate recursive nonmonotone aggregate atoms it is possible to simulate disjunctive rules, for which reasoning tasks are in general more complex.
However, simple nonmonotone aggregates occur frequently in programs, for example, checking whether the value of an aggregate function is equal to a constant, is nonmonotone in most cases. But many of these aggregate atoms are not powerful enough to simulate disjunctions, and hence reasoning tasks are in fact less complex in these cases.
In this paper we present methods for identifying "easy" nonmonotone aggregate atoms, called *polynomially decomposable nonmonotone aggregate atoms*, giving also a way for translating programs with polynomially decomposable nonmonotone aggregate atoms to programs without nonmonotone aggregate atoms.

1 Introduction

The introduction of aggregate atoms [1–8] is one of the major linguistic extensions to Answer Set Programming of the recent years. While both semantic and computational properties of standard (aggregate-free) logic programs have been deeply investigated, so far relatively few works have focused on logic programs with aggregates; some of their semantic properties and their computational features are still not fully clarified.

The proposal for answer set semantics in [8] seems to be receiving a consensus. Recent works, such as [9–11] give further support for the plausibility of this semantics by relating it to established constructs for aggregate-free programs. In particular, [9] presented a semantics for very general programs, and showed that it coincides with both answer sets of [8] and Smodels answer sets (the latter holds for weight constraints with positive weights only). In [10] the notion of unfounded sets is extended from aggregate-free programs to nondisjunctive programs with a certain class of aggregates in a conservative way, retaining important semantical and computational properties. In [11], this concept was finally generalized to disjunctive programs with arbitrary aggregates.

A particular feature of answer sets for programs with aggregates is that for programs without disjunctive rules, the complexity of reasoning tasks increases with the presence of so-called nonmonotone aggregate atoms. Intuitively, the reason is that with appropriate recursive nonmonotone aggregate atoms it is possible to simulate disjunctive rules, for which reasoning tasks are in general more complex. From this it follows that for programs with nonmonotone aggregates, more involved evaluation mechanisms are necessary.

However, by far not all of the nonmonotone aggregate atoms give rise to an increased complexity. A common example of this is an aggregate counting the number of true atoms in a set and testing whether it is equal to a constant. While these aggregates are nonmonotone, it is not harder

* This work was supported by an APART grant of the Austrian Academy of Sciences and the European Commission under projects IST-2002-33570 INFOMIX, IST-2001-37004 WASP.

to evaluate them than monotone or antimonotone aggregates. Indeed, they can be decomposed into a conjunction of a monotone and an antimonotone aggregate.

In this paper we present methods for identifying such "easy" nonmonotone aggregate atoms, which we will call *polynomially decomposable nonmonotone aggregate atoms*. We will also give a way for translating programs with polynomially decomposable nonmonotone aggregate atoms to programs without nonmonotone aggregate atoms.

Summarizing, our contributions are as follows:

– We define the notion of *polynomially decomposable nonmonotone aggregate atoms*
– We show that reasoning over nondisjunctive programs with decomposable nonmonotone aggregate atoms is less complex than with general nonmonotone aggregate atoms.
– We present a method on how to convert programs with polynomially decomposable nonmonotone aggregate atoms into programs without nonmonotone aggregate atoms.

2 Logic Programs with Aggregates

In this section, we recall syntax, semantics, and some basic properties of logic programs with aggregates. For simplicity, we will only deal with ground programs, for a more general syntax, see, e.g. [8].

2.1 Syntax

We assume that the reader is familiar with standard LP; we refer to the respective constructs as *standard atoms*, *standard literals* (using the default negation symbol `not`), *standard rules*, and *standard programs*. For simplicity, we will assume that strong negation (usually denoted as $\neg$) does not occur in the program[1]. Two literals are said to be complementary if they are of the form p and $\mathtt{not}\ p$ for some atom p. Given a literal L, $\mathtt{not}.L$ denotes its complementary literal. Accordingly, given a set A of literals, $\mathtt{not}.A$ denotes the set $\{\mathtt{not}.L \mid L \in A\}$. For further background, see [12, 13].

Set Terms. A DLP$^{\mathcal{A}}$*set term* is a set of pairs of the form $\langle \bar{t} : Conj \rangle$, where $\bar{t}$ is a list of constants and *Conj* is a 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 of constants to a constant.

Examples for aggregate function symbols are: `#count` (number of terms), `#sum` (sum of numbers), `#times` (product of numbers), `#min` (minimum term), `#max` (maximum term)`#avg` (average of numbers).

We also assume the presence of subset and superset aggregates, denoted $\subseteq_I$, $\subset_I$, $\supseteq_I$, $\supset_I$ which evaluate to 1 if the set of constants is a subset, strict subset, superset, strict superset, respectively, of the set I, and to 0 otherwise.

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

Example 1. Intuitively, the following aggregate atom holds whenever $r(2)$ together with $a(2,k)$ or $a(2,c)$ hold.

$$\#\mathtt{max}\{\langle 2 : r(2), a(2,k)\rangle, \langle 2 : r(2), a(2,c)\rangle\} > 1$$

[1] Note that this is no real restriction, as for the problems we consider in this paper, strong negation can be simulated by introducing new symbols.

An *atom* is either a standard atom or an aggregate atom. A *literal* L is an atom A or a standard literal. Note that for simplicity we do not consider default negated aggregate atoms.

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

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

where $a_1, \cdots, a_n$ are standard atoms, $b_1, \cdots, b_k$ are atoms, $b_{k+1}, \cdots, b_m$ are standard atoms, and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \ \vee \ \cdots \ \vee \ a_n$ is referred to as the *head* of r while the conjunction $b_1, ..., b_k, \ \texttt{not} \ \ b_{k+1}, ..., \texttt{not} \ \ b_m$ is the *body* of r. We denote the set of head atoms by $H(r)$, and we denote the set $\{b_1, ..., b_k, \ \texttt{not} \ \ b_{k+1}, ..., \texttt{not} \ \ b_m\}$ of the body literals by $B(r)$. $B^+(r)$ and $B^-(r)$ denote, respectively, the set of positive and negative literals in $B(r)$.

A DLP$^{\mathcal{A}}$ *program* is a set of DLP$^{\mathcal{A}}$ rules. In the sequel, we will often drop DLP$^{\mathcal{A}}$, when it is clear from the context. A *global* variable of a rule r appears in a standard atom of r (possibly also in other atoms); all other variables are *local* variables.

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

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

An aggregate atom $A = f(S) \prec k$ is *true w.r.t.* I if: (i) $I(f(S)) \neq \bot$, and, (ii) $I(f(S)) \prec k$ holds; otherwise, A is false. A rule r is *satisfied w.r.t.* I if some head atom is true w.r.t. I whenever all body literals are true w.r.t. I.

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

Minimal Models. Given an interpretation I, a rule r is *satisfied w.r.t.* I if some head atom is true w.r.t. I whenever all body literals are true w.r.t. I. A total interpretation M is a *model* of a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ if all $r \in \textit{Ground}(\mathcal{P})$ are satisfied w.r.t. M. A model M for $\mathcal{P}$ is (subset) minimal if no model N for $\mathcal{P}$ exists such that $N \subset M$.

Example 3. Looking at the following program,

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

We can verify that $I = \{q(2), p(2,2), t(2)\}$ is a model of it. Omitting either of $q(2)$ or $p(2,2)$ from I would make one of the disjunctive rules unsatisfied, and since truth of $q(2)$ and $p(2,2)$ makes the body of the second aggregate rule true, also $t(2)$ must be true in any model. We can thus conclude that I is a minimal model of the program.

Answer Sets. We now recall the generalization of the Gelfond-Lifschitz transformation and the associated notion of answer sets for DLP$^{\mathcal{A}}$ programs from [8]: Given a DLP$^{\mathcal{A}}$ program $\mathcal{P}$ and a total interpretation I, let $\mathcal{P}^I$ denote the transformed program obtained from $\mathcal{P}$ by deleting all rules in which a body literal is false w.r.t. I. I is an answer set of a program $\mathcal{P}$ if it is a minimal model of $\mathcal{P}^I$.

Example 4. Consider interpretation $I_1 = \{p(a)\}$, $I_2 = \emptyset$ and two programs

$$P_1 = \{p(a) :- \#\texttt{count}\{\langle a : p(a)\rangle\} > 0.\} \text{ and}$$
$$P_2 = \{p(a) :- \#\texttt{count}\{\langle a : p(a)\rangle\} < 1.\}.$$

We observe that $P_1^{I_1} = P_1$, $P_1^{I_2} = \emptyset$. Furthermore, $P_2^{I_1} = \emptyset$, $P_2^{I_2} = P_2$ hold.

I_2 is the only answer set of P_1 (since I_1 is not a minimal model of $P_1^{I_1}$), while P_2 admits no answer set (I_1 is not a minimal model of $P_2^{I_1}$, and I_2 is not a model of $P_2 = P_2^{I_2}$).

Monotonicity. A literal ℓ is *monotone*, if for all interpretations I, J, such that $I \subseteq J$, we have that: (i) ℓ true w.r.t. I implies ℓ true w.r.t. J, and (ii) ℓ false w.r.t. J implies ℓ false w.r.t. I. A literal ℓ is *antimonotone*, if the opposite happens, that is, for all interpretations I, J, such that $I \subseteq J$, we have that: (i) ℓ true w.r.t. J implies ℓ true w.r.t. I, and (ii) ℓ false w.r.t. I implies ℓ false w.r.t. J. A literal ℓ is *nonmonotone*, if it is neither monotone nor antimonotone.

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

Example 5. $\#\texttt{count}\{S\} > 1$ is monotone for any S, while $\#\texttt{count}\{S\} < 1$ is antimonotone for any S. $\#\texttt{count}\{S\} = 1$ is nonmonotone for any S with $|S| > 1$. $\subseteq_I\{S\} > 0$ is monotone for any I and S. $\subset_I\{S\} > 0$ is monotone for any I and S. $\supseteq_I\{S\} > 0$ is antimonotone for any I and S. $\supset_I\{S\} > 0$ is antimonotone for any I and S.

Computational Complexity We briefly recall the complexities of various tasks associated with logic programs with aggregates. We assume familiarity with standard complexity terminology, in particular classes P (polynomial time) NP/$CONP$, and Σ_2^P/Π_2^P (cf. [14]).

On the left of Table 1, the complexities of model checking (i.e., given a ground program $\mathcal{P}$ and an interpretation I, check whether I is an answer set of $\mathcal{P}$) are reported for various classes of programs. There, the rows indicate the kinds of aggregates (m – monotone, a – antimonotone, n – nonmonotone) allowed in programs, while the columns vary over the presence of negation and disjunction. All co-NP entries are completeness results. The results in the first row are well-known results of the literature (cf. [15]), the P entries for $\{m, a\}$ follow from recent results in [10], and the co-NP entries have been shown in [11]. It becomes clear from this table that a complexity increase occurs with the presence of either disjunction or nonmonotone aggregates, and that these two factors together do not cause a further increase.

On the right hand side of Table 1, we have summarized results from the literature (see [15, 8, 10]) for the problem of cautious reasoning (i.e., given a ground program $\mathcal{P}$ and a ground standard atom a, check whether a is in all answer sets of $\mathcal{P}$). We observe that the complexity increase occurs at the same places, and indeed one can blame the necessity of co-NP checks for the Π_2^P results.

Checking	$\emptyset$	{`not` }	$\{\vee\}$	{`not` , $\vee$}
$\emptyset$	P	P	co-NP	co-NP
$\{m, a\}$	P	P	co-NP	co-NP
$\{m, a, n\}$	co-NP	co-NP	co-NP	co-NP

Cautious	$\emptyset$	{`not` }	$\{\vee\}$	{`not` , $\vee$}
$\emptyset$	P	co-NP	Π_2^P	Π_2^P
$\{m, a\}$	co-NP	co-NP	Π_2^P	Π_2^P
$\{m, a, n\}$	Π_2^P	Π_2^P	Π_2^P	Π_2^P

Table 1. Complexity of Answer Set Checking (left) and Cautious Reasoning (right)

3 Polynomially Decomposable Nonmonotone Aggregate Atoms

It is striking that from Table 1 it appears that a single, apparently "innocent", aggregate like $\#\texttt{count}\{< 1 : a >, < 2 : b >\} = 1$ will sensibly increase the reasoning complexity. Indeed, this aggregate could be equivalently replaced by a conjunction

$$\#\texttt{count}\{< 1 : a >, < 2 : b >\} \leq 1, \#\texttt{count}\{< 1 : a >, < 2 : b >\} \geq 1,$$

of monotone and antimonotone aggregates, thus eliminating the nonmonotone aggregate.

In fact, such a decomposition is possible for each nonmonotone aggregate, if one allows the use of custom aggregates (rather than a limited set of fixed aggregates), especially subset and superset aggregates, and the introduction of new symbols. However, this operation is only polynomial (and hence effective) if the number of the truth value changes of the nonmonotone aggregate in the lattice of total interpretations induced by $<$ is polynomially bounded.

Intuitively, for each change from false to true, one has to use a corresponding custom monotone aggregate, and symmetrically for transitions from true to false a custom antimonotone aggregate. Each of these custom aggregates gives rise to a rule with only the respective aggregate in its body, the heads consist of a new atom, the same for each of these rules. Occurrences of the original nonmonotone aggregate are then replaced by a new symbol.

Definition 1. *Given an aggregate atom ℓ, an* exclusive truth interval (I, J) *consists of two interpretations $I < J$, such that ℓ is false w.r.t. both I and J, and for each interpretation K such that $I < K < J$, ℓ is true w.r.t. K. An exclusive truth interval (I, J) is* non-empty *if some interpretation K exists such that $I < K < J$.*

An inclusive truth interval $[I, J]$ *for ℓ consists of two interpretations $I \subseteq J$ such that ℓ is true for all interpretations K such that $I \subseteq K \subseteq J$. An inclusive truth interval $[I, J]$ is* maximal *if no inclusive truth interval $[K, L]$ exists such that $K < I$ and $J \subseteq L$ or $K \subseteq I$ and $J < L$ holds.*

Any literal admitting a non-empty exclusive truth interval is clearly nonmonotone.

Example 6. Let ℓ_1 be $\#\texttt{count}\{< 1 : a >, < 2 : b >\} = 1$, we can observe that ℓ_1 is false w.r.t. $\emptyset$, true w.r.t. $\{a\}$ and $\{b\}$, and false w.r.t. $\{a, b\}$. Therefore, $(\emptyset, \{a, b\})$ is a non-empty exclusive truth interval, while $[\{a\}, \{a\}]$ and $[\{b\}, \{b\}]$ are the maximal inclusive truth intervals of ℓ_1.

For ℓ_2 being $\#\texttt{count}\{\langle 1 : a\rangle, \langle 2 : b\rangle, \langle 3 : c\rangle, \langle 4 : d\rangle\} = 2$ we obtain

$$\begin{array}{lll}
(\{a\}, \{a, b, c\}) & (\{a\}, \{a, b, d\}) & (\{a\}, \{a, c, d\}) \\
(\{b\}, \{a, b, c\}) & (\{b\}, \{a, b, d\}) & (\{b\}, \{b, c, d\}) \\
(\{c\}, \{a, b, c\}) & (\{c\}, \{a, c, d\}) & (\{c\}, \{b, c, d\}) \\
(\{d\}, \{a, b, d\}) & (\{d\}, \{a, c, d\}) & (\{d\}, \{b, c, d\})
\end{array}$$

as exclusive truth intervals, and

$$\begin{array}{lll}
[\{a, b\}, \{a, b\}] & [\{a, c\}, \{a, c\}] & [\{a, d\}, \{a, d\}] \\
[\{b, c\}, \{b, c\}] & [\{b, d\}, \{b, d\}] & [\{c, d\}, \{c, d\}]
\end{array}$$

as maximal inclusive truth intervals.

Definition 2. *Given an aggregate atom ℓ, it admits an* initial truth interval *up to J, if $[\emptyset, J]$ is a maximal inclusive truth interval for ℓ.*

Symmetrically, ℓ admits a final truth interval *from J on, if $[J, B_{\mathcal{P}}]$ is a maximal inclusive truth interval.*

Monotone literals admit a final truth interval, while antimonotone literals admit an initial truth interval.

Definition 3. *An inclusive truth interval $[I, J]$ and an exclusive truth interval (K, L) are* non-overlapping *if for no interpretation H both $I \subseteq H \subseteq J$ and $K < H < L$ holds.*

Definition 4. *A decomposition of an aggregate atom α w.r.t. a program $\mathcal{P}$ using subset aggregates and fresh atoms is defined as follows: Let $S = \{I_1, \ldots, I_n\}$ be a set of non-overlapping truth intervals (exclusive or inclusive), such that there is no other non-overlapping truth interval for α. For each inclusive truth interval $[I, J] \in S$, we create a rule with the head atom $ti_{[I,J]}$, which is assumed to be a new symbol. ${B_\mathcal{P}}^{B_\mathcal{P}}$ represents a sequence of tuples $\langle a : a \rangle$ for all elements a of $B_\mathcal{P}$.*

$$ti_{[I,J]} :- \supseteq_I \{B_\mathcal{P} : B_\mathcal{P}\} > 0, \subseteq_J \{B_\mathcal{P} : B_\mathcal{P}\} > 0.$$

In a similar way, for each exclusive truth interval $(I, J) \in S$, we create the following rule, assuming again that $ti_{(I,J)}$ is a new symbol.

$$ti_{(I,J)} :- \supset_I \{B_\mathcal{P} : B_\mathcal{P}\} > 0, \subset_J \{B_\mathcal{P} : B_\mathcal{P}\} > 0.$$

Finally, we create the following rule:

$$decomp_\alpha :- ti_{I_1}, \ldots, ti_{I_n}.$$

For any program P, a decomposed program $P^d(S)$ is obtained by replacing each aggregate atom α by $decomp_\alpha$ and adding a decomposition of α.

The intuition is that each truth interval can be covered by the conjunction of two subset aggregates, and that the truth of the whole aggregate atom is the conjunction of all truth intervals. Any decomposed program consists only of monotone and antimonotone literals. We point out that deciding whether a subset or superset aggregate is true is clearly polynomial in the size of the aggregate. A decomposed program has the same answer sets as the original one, when projected on the atoms of the original programs.

Proposition 1. *Any program P can be rewritten into a program P' containing only monotone and antimonotone literals, such that its answer sets coincide on the symbols of P.*

Proof. Any program possesses a decomposed program $P^d(S)$, which consists only of monotone and antimonotone literals, satisfying the conditions. In $P^d(S)$, any atom $decomp_\alpha$ is true in all interpretations in which α is true, since these interpretations are characterised precisely by the set of truth intervals S.

Example 7. Consider the program

$$a \vee b :- \#\mathtt{count}\{< 1 : a >, < 2 : b >\} = 1.$$

As shown in Example 6 above, the maximal inclusive truth intervals of the aggregate atom $\#\mathtt{count}\{< 1 : a >, < 2 : b >\} = 1$ are $[\{a\}, \{a\}]$ and $[\{b\}, \{b\}]$. The decomposed program is then

$$\begin{array}{l} a \vee b :- decomp_\ell. \\ ti_{[\{a\},\{a\}]} :- \subseteq_{\{a\}}\{\langle a : a\rangle, \langle b : b\rangle\} > 0, \supseteq_{\{a\}}\{\langle a : a\rangle, \langle b : b\rangle\} > 0. \\ ti_{[\{b\},\{b\}]} :- \subseteq_{\{b\}}\{\langle a : a\rangle, \langle b : b\rangle\} > 0, \supseteq_{\{b\}}\{\langle a : a\rangle, \langle b : b\rangle\} > 0. \\ decomp_\ell :- ti_{[\{a\},\{a\}]}, ti_{[\{b\},\{b\}]}. \end{array}$$

Alternatively, we may use the non-empty exclusive truth intervals, in this case leading to the more concise program

$$\begin{array}{l} a \vee b :- decomp_\ell. \\ ti_{(\emptyset,\{a,b\})} :- \subset_\emptyset\{\langle a : a\rangle, \langle b : b\rangle\} > 0, \supset_{\{a,b\}}\{\langle a : a\rangle, \langle b : b\rangle\} > 0. \\ decomp_\ell :- ti_{(\emptyset,\{a,b\})}. \end{array}$$

Unfortunately, such a decomposition can be very large, in the worst case there may be exponentially many truth intervals. A practical decomposition must therefore be bounded. In order to guarantee tractability of the translation, a polynomial bound is needed.

Definition 5. *An aggregate atom ℓ is* polynomially decomposable *w.r.t. a program $\mathcal{P}$, if its number of non-empty exclusive truth intervals or maximal inclusive truth intervals is polynomial in the size of $\mathcal{P}$.*

Proposition 2. *Any program $\mathcal{P}$ with polynomially decomposable nonmonotone aggregate atoms (with given truth intervals) can be transformed into a program $\mathcal{P}'$ without nonmonotone aggregates in polynomial time, possibly using new symbols and subset aggregates.*

From the construction, the following proposition can easily be derived:

Proposition 3. *The complexity of reasoning tasks for programs all of whose nonmonotone aggregate atoms are polynomially decomposable is the same as for programs with monotone and antimonotone aggregate atoms, assuming that the truth intervals of all nonmonotone aggregates are known.*

The notion of polynomially decomposable nonmonotone aggregate atoms is not intended to be a notion which is tested during answer set computations, as checking whether an aggregate atom has a polynomially bounded number of truth intervals is in general intractable. Such a test would therefore compromise the advantages brought about by the knowledge that only polynomially decomposable aggregate atoms are present in a program. Instead, one can analyze classes of aggregates in order to test whether they are polynomially decomposable. In the following, we provide two examples of such classes.

Proposition 4. *Aggregate atoms $\#\texttt{count}\{S\} = k$ are polynomially decomposable for any S and k.*

Proof. If $|S| = n$, the number of inclusive truth intervals is $\binom{n}{k}$, which is of the order n^k, where k is a given constant.

Proposition 5. *Aggregate atoms $\#\texttt{sum}\{S\} = k$ are polynomially decomposable for S consisting of positive integers and k.*

Proof. The number of positive summands for k out of a multiset of cardinality n is also bounded by $\binom{n}{k}$, therefore these aggregates have at most n^k inclusive truth intervals.

A similarly general statement cannot be made for $\#\texttt{sum}$, when general integers are involved, as we cannot guarantee that the number of subsets of S for which the sum equals k is bounded by a polynomial. To see this, consider aggregate atoms like $\#\texttt{sum}\{\langle 0 : a_1\rangle, \ldots, \langle 0 : a_n\rangle, \langle 1 : a_{n+1}\rangle\} = 0$ and consider that other $b_1, \ldots, b_n$ standard atoms are present. Then there exists an exponential number of truth intervals for this aggregate atom. Similar considerations hold for average aggregates.

4 Conclusion

In this paper, we have defined a class of programs with nonmonotone aggregates, for which, in the case of nondisjunctive programs, the complexity of cautious reasoning does not jump from co-NP to Σ_2^P. In fact, these programs can be rewritten as programs without nonmonotone aggregates in polynomial time, if the use of "custom" aggregates (as opposed to a fixed set of aggregates), and in particular of subset and superset aggregates, is allowed. Reasoning tasks are therefore always of the same complexity as for programs in which only monotone and antimonotone aggregate atoms are allowed.

References

1. Kemp, D.B., Stuckey, P.J.: Semantics of Logic Programs with Aggregates. In: ISLP'91, MIT Press (1991) 387–401
2. Denecker, M., Pelov, N., Bruynooghe, M.: Ultimate Well-Founded and Stable Model Semantics for Logic Programs with Aggregates. In Codognet, P., ed.: ICLP-2001, (2001) 212–226
3. Gelfond, M.: Representing Knowledge in A-Prolog. In: Computational Logic. Logic Programming and Beyond. LNCS 2408 (2002) 413–451
4. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. Artificial Intelligence **138** (2002) 181–234
5. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate Functions in DLV. In: ASP'03, Messina, Italy (2003) 274–288 Online at `http://CEUR-WS.org/Vol-78/`.
6. Pelov, N., Truszczyński, M.: Semantics of disjunctive programs with monotone aggregates - an operator-based approach. In: NMR 2004. (2004) 327–334
7. Pelov, N., Denecker, M., Bruynooghe, M.: Partial stable models for logic programs with aggregates. In: LPNMR-7. LNCS 2923
8. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: JELIA 2004. LNCS 3229
9. Ferraris, P.: Answer Sets for Propositional Theories. In: Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, 2005, Proceedings. LNCS 3662
10. Calimeri, F., Faber, W., Leone, N., Perri, S.: Declarative and Computational Properties of Logic Programs with Aggregates. In: Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05). (2005) 406–411
11. Faber, W.: Unfounded Sets for Disjunctive Logic Programs with Arbitrary Aggregates. In: Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, 2005, Proceedings. LNCS 3662
12. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. CUP (2002)
13. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC **9** (1991) 365–385
14. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
15. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys **33** (2001) 374–425