Translation of Aggregate Programs to Normal Logic Programs

Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe

Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
E-mail: {pelov,marcd,maurice}@cs.kuleuven.ac.be

Abstract. We define a translation of aggregate programs to normal logic programs which preserves the set of partial stable models. We then define the classes of definite and stratified aggregate programs and show that the translation of such programs are, respectively, definite and stratified logic programs. Consequently these two classes of programs have a single partial stable model which is two-valued and is also the well-founded model. Our definition of stratification is more general than the existing one and covers a strictly larger class of programs.

1 Introduction

In the recent years there is an increasing interest in extending the syntax and semantics of answer set programming systems with aggregate atoms [2, 8, 14]. Current work supports only a limited number of aggregate functions. For example the SMODELS system supports only cardinality and summation aggregates. The **dlv** system supports, in addition, MIN, MAX, and PROD aggregates but it does not support recursion over aggregates [2].

In two papers [5, 11] we defined partial stable semantics for logic programs with arbitrary aggregate relations. The semantics are based on Approximation Theory [3, 4] which provides a solid algebraic framework for defining non-monotonic semantics. The ultimate well-founded and stable semantics of aggregate programs [5] are the most precise semantics which could be defined within the framework of Approximation Theory. They are extensions of the ultimate semantics for standard logic programs [4] to logic programs with aggregates and are, in general, different from the standard well-founded and stable semantics for logic programs. On the other hand, the semantics of [11] are extensions of the partial stable semantics of normal logic programs [12].

In this paper we continue the investigation of aggregates by defining a translation of aggregate programs to normal logic programs. This translation has the property that partial stable models of the aggregate program [11] coincide with partial stable models of its translation [12]. For some aggregates our translation is equivalent to the translation of weight constraints to nested expressions by Ferraris and Liftshitz [6]. However, our translation is defined for arbitrary aggregate relations including non-monotone ones while weight constraints are essentially a combination of a monotone and anti-monotone aggregate relations.

As an application of the translation we present a novel definition of stratification of aggregate programs. The existing definitions [2, 5, 10] treat aggregate atoms as negative literals. So, all atoms defining the set expression of an aggregate atom have to be at a strictly lower stratum than the atom in the head. The difference of our definition is that for monotone aggregate relations, the positive atoms of the set expression can be at the same stratum as the atom in the head. Similarly, for anti-monotone aggregate relations, the negative atoms in the set expression can be at the same stratum as the atom in the head. In this way we obtain a more general definition which covers a strictly larger class of programs. Under the semantics of [11] such stratified aggregate programs have a two-valued well-founded model which coincides with the single two-valued stable model.

The paper is organized as follows. We start by presenting the necessary background on Approximation Theory in Section 2. In Section 3 we recall the syntax of aggregate programs and the partial stable semantics of [11]. The main results of the paper are presented in Section 4 and the proofs are given in Section 5. Finally, we discuss related work (Section 6) and give some concluding remarks (Section 7).

2 Preliminaries on Approximation Theory

We first present the necessary background on Approximation Theory following [4] on which the semantics of aggregate programs of [11] is based. The basic concept is that of an approximation of the elements of a lattice $\langle L, \leq \rangle$ by pairs (x, y). If $x \leq y$ then we call the pair (x, y) *consistent* and if $x = y$ then we call the pair *exact*. We denote the set of all consistent pairs on L with L^c. We call the elements in the set L^c *partial elements*. For example if L is a set of interpretations then the elements of L^c are called partial interpretations. The partial order relation $\leq$ on L can be extended in two ways to partial orders on L^c:

truth order:	$(x, y) \leq_t (x_1, y_1)$ if and only if $x \leq x_1$ and $y \leq y_1$
precision order:	$(x, y) \leq_p (x_1, y_1)$ if and only if $x \leq x_1$ and $y_1 \leq y$

A consistent pair (x, y) can be seen as an approximation of all elements in the interval $[x, y] = \{z \in L \mid x \leq z \leq y\}$ which is always non-empty. In this sense, the precision order $\leq_p$ corresponds to the precision of the approximation, that is $(x, y) \leq_p (x_1, y_1)$ if and only if $[x, y] \supseteq [x_1, y_1]$. Exact pairs (x, x) approximate a single element x and represent the embedding of L in L^c.

Consider the lattice $\mathcal{TWO} = \{\mathbf{f}, \mathbf{t}\}$ of classical truth values ordered as $\mathbf{f} < \mathbf{t}$. We denote the set of consistent elements $\mathcal{TWO}^c$ of $\mathcal{TWO}$ with $\mathcal{THREE}$. The exact pairs $(\mathbf{f}, \mathbf{f})$ and $(\mathbf{t}, \mathbf{t})$ are the embedding of the classical truth values $\mathbf{f}$ and $\mathbf{t}$ respectively and are maximal in the precision order. Only the pair $(\mathbf{f}, \mathbf{t})$ approximates more than one truth value, namely the set $\{\mathbf{f}, \mathbf{t}\}$, and corresponds to the value of *undefined*, denoted with $\mathbf{u}$. The truth order $\leq_t$ is used to define conjunction and disjunction in $\mathcal{THREE}$ which are interpreted as greatest lower bound $\wedge_t$ and least upper bound $\vee_t$ respectively. Negation in $\mathcal{THREE}$ is defined as $\neg(x, y) = (\neg y, \neg x)$. In particular, $\neg\mathbf{f} = \mathbf{t}$, $\neg\mathbf{t} = \mathbf{f}$, and $\neg\mathbf{u} = \mathbf{u}$. This interpretation of the connectives $\neg$, $\wedge$, and $\vee$ is the same as the one given by Kleene's strong three-valued logic.

We briefly recall the definition of partial stable operator, the definition of different classes of stable fixpoints, and some basic properties.

Definition 1 (Partial Approximating Operator). *Let $O : L \to L$ be an operator on a complete lattice L. We say that $A : L^c \to L^c$ is a* partial approximating operator *of O if the following conditions are satisfied:*

– A extends O*, i.e. $A(x, x) = (O(x), O(x))$ for every $x \in L$;*
– *A is $\leq_p$-monotone.*

Because A is a $\leq_p$-monotone operator on a complete semilattice then it has a least fixpoint. It is called the *Kripke-Kleene* fixpoint of A and denoted with $KK(A)$.

We denote the projection of an approximating operator $A : L^c \to L^c$ on the first and second components with A^1 and A^2, i.e. if $A(x, y) = (u, v)$ then $A^1(x, y) = u$ and $A^2(x, y) = v$. From the $\leq_p$-monotonicity of A follows that A^1 is monotone in the first argument and A^2 is monotone in the second argument. Recall that for a fixed element $b \in L$, the operator $A^1(\cdot, b)$ is defined only on the domain $[\bot, b]$. However, it is possible that for some element $x \in [\bot, b]$, $A^1(x, b) \not\in [\bot, b]$. Similarly, for a fixed element $a \in L$, the operator $A^2(a, \cdot)$ is defined on the domain $[a, \top]$ but it is possible that for some element $y \in [a, \top]$, $A^2(a, y) \not\in [a, \top]$. Denecker et al. [4] showed that if (a, b) is a post-fixpoint of A, i.e. $(a, b) \leq_p A(a, b)$ then the operators $A^1(\cdot, b)$ and $A^2(a, \cdot)$ are well-defined on the domains $[\bot, b]$ and $[a, \top]$ respectively. Such pairs (a, b) are called A*-reliable*. Let L^r denote the set of A-reliable pairs. The *partial stable operator* $S : L^r \to L^c$ is defined as follows:

$$\begin{aligned} S(a, b) = (\ & lfp(\lambda x.\ A^1|_{[\bot,b]}(x, b)),\\ & lfp(\lambda y.\ A^2|_{[a,\top]}(a, y))\ ). \end{aligned}$$

The fixpoints of S are called *partial stable fixpoints* of A. The stable operator S is monotone in the precision order $\leq_p$ and has a least fixpoint, called the *well-founded* fixpoint of A and denoted with $WF(A)$. This fixpoint can be computed by transfinite iteration of S starting from the bottom element in the $\leq_p$ order which is $(\bot, \top)$. Of special interest are elements in the set $ST(A) = \{x \in L \mid (x, x)$ is a fixpoint of $S\}$ which are called *exact stable fixpoints* of A. An important property of exact stable fixpoints is that they are minimal fixpoints of O. For such fixpoints it is possible to give a simpler characterization: x is an exact stable fixpoint if and only if $O(x) = x$ and $lfp(A^1(\cdot, x)) = x$.

In logic programming we are interested in approximating the immediate consequence operator T_P. The standard partial approximating operator of T_P is Fitting's three-valued Φ_P operator [7]. The Kripke-Kleene fixpoint of Φ_P is equal to the Kripke-Kleene semantics of P [7]. Although partial stable models [12] are defined in a very different way they do coincide with partial stable fixpoints of Φ_P [3]. Finally, exact stable fixpoints of Φ_P are equal to the set of stable models defined by Gelfond and Lifschitz [9]. To see this, let $GL(P; I)$ denote the Gelfond-Lifschitz transformation of a program P with respect to an interpretation I. It is easy to show that $\Phi^1_P(I_1, I_2) = T_{GL(P;I_2)}(I_1)$. Thus $lfp(\Phi^1_P(\cdot, I))$ is equal to the least model of $GL(P; I)$. Consequently, I is an exact stable fixpoint of Φ_P if and only if I is a stable model of P.

3 Aggregate Programs

3.1 Aggregate Functions and Relations

An aggregate is typically understood as a function from a multiset to a single element. A multiset (also called a *bag*) is similar to a set except that an object can occur multiple times. The most general definition of a multiset M on a domain D is a mapping $M : D \to Card$ where $Card$ is the class of cardinal numbers. For every element $x \in D$, $M(x)$ gives the multiplicity of x. In this work we consider only finite multisets. First, we define a *multiset with finite multiplicity* as a function $M : D \to \mathbb{N}$ where $\mathbb{N}$ is the set of natural numbers. A *finite multiset* is a multiset with finite multiplicity such that $M(x) > 0$ only for a finite number of elements. We denote the set of all finite multisets on D with $\mathcal{M}(D)$. The subset relation between multisets is defined as follows: $M_1 \subseteq M_2$ if and only if $M_1(x) \leq M_2(x)$ for all $x \in D$. The *additive union* $M = M_1 \uplus M_2$ of M_1 and M_2 is defined as $M(x) = M_1(x) + M_2(x)$ for all $x \in D$. The *multiset difference* $M = M_1 - M_2$ of M_1 and M_2 is defined as $M(x) = M_1(x) - M_2(x)$ if $M_1(x) \geq M_2(x)$ and $M(x) = 0$ otherwise. We denote the *empty multiset* with $\emptyset$. For the rest of the paper by multiset we mean a finite multiset.

In this work we treat aggregates as relations — an *aggregate relation* on D is any relation $\textsc{r} \subseteq \mathcal{M}(D_1) \times D_2$. An aggregate relation $\textsc{r}$ represents an *aggregate function* when for every multiset M there is exactly one element $d \in D_2$ such that $(M, d) \in \textsc{r}$. Examples of aggregate relations are $\textsc{card} \subseteq \mathcal{M}(D) \times \mathbb{N}$ for any set D defined as $(M, d) \in \textsc{card}$ if and only if $d = \sum_{x \in D} M(x)$ and $\textsc{sum} \subseteq \mathcal{M}(\mathbb{R}) \times \mathbb{R}$ "returning" the sum of the elements in the input multiset, i.e. $(M, d) \in \textsc{sum}$ if and only if $d = \sum_{x \in D} x.M(x)$. Because we consider only finite multisets then both relations are defined for all input multisets and consequently they are also aggregate functions. An aggregate relation $\textsc{r} \subseteq \mathcal{M}(D_1) \times D_2$ represents a *partial aggregate function* when for every multiset M there is at most one element $d \in D_2$ such that $(M, d) \in \textsc{r}$. For example taking an average of an empty multiset is not defined. So the aggregate relation $\textsc{avg} \subseteq \mathcal{M}(\mathbb{R}) \times \mathbb{R}$ is a partial aggregate function.

Another advantage of representing aggregates as relations is that we can obtain new aggregate relations by composition of an existing aggregate with another relation. Let $\textsc{r} \subseteq \mathcal{M}(D_1) \times D_2$ be an aggregate relation and $P \subseteq D_2 \times D_3$ a binary relation. The *composition* of $\textsc{r}$ and P is an aggregate relation $\textsc{r}_P \subseteq \mathcal{M}(D_1) \times D_3$ defined as follows: $(M, d) \in \textsc{r}_P$ if and only if there exists $a \in D_2$ such that $(M, a) \in \textsc{r}$ and $(a, d) \in P$. Typically, the binary relation P is some partial order relation on the domain D. For example the $\textsc{sum}_{\geq}$ aggregate relation means that the sum of the elements in the multiset is greater than or equal to the second argument.

Monotonicity of aggregate relations is defined as follows.

Definition 2. *Let* $\textsc{r}$ *be an aggregate relation on* D*. We say that* $\textsc{r}$ *is:*

– monotone *if* $(M_1, d) \in \textsc{r}$ *and* $M_1 \subseteq M_2$ *implies* $(M_2, d) \in \textsc{r}$*;*
– anti-monotone *if* $(M_2, d) \in \textsc{r}$ *and* $M_1 \subseteq M_2$ *implies* $(M_1, d) \in \textsc{r}$.

3.2 Set Expressions, Aggregate Atoms, and Logic Programs

Aggregate programs are built over a set of propositional atoms At and a domain D. A *literal* is an atom a (*positive literal*) or the negation of an atom $not\ a$ (*negative literal*). The *complement* $\overline{L}$ of a literal L is defined as $\overline{L} = not\ a$ if $L = a$ and $\overline{L} = a$ if $L = not\ a$.

A *weight literal* is an expression $L = w$ where L is a literal and $w \in D$ is the *weight* associated with L. A *set expression* is a finite set of weight literals. Syntactically, we denote a set expression as $\{L_1 = w_1, \ldots, L_n = w_n\}$. The set of all set expression is denoted with $\mathcal{SE}$. It forms a lattice under the subset order. However, since we consider only finite set expressions, this lattice is not complete. For a set expression s, $w(s)$ denotes the multiset consisting of the weights of all weight literals in s.

An *aggregate atom* has the form $\mathrm{R}(s, d)$ where R is an aggregate relation, s is a set expression, and $d \in D$. An example of an aggregate atom is $\mathrm{SUM}_{\geq}(\{a = 1, b = 2\}, 2)$. It is true in an interpretation I if the aggregate relation is true for the multiset consisting of the weights of the literals true in I, i.e. if the sum of those weights is greater than or equal to 2.

A *rule* has the form $A \leftarrow B_1 \wedge \ldots \wedge B_n$ where A is an atom called the *head* of the rule and every B_i is a a literal or an aggregate atom. The set $B = \{B_1, \ldots, B_n\}$ is called the *body* of the rule. It can be partitioned in three sets namely, positive literals $pos(B)$, negative literals $neg(B)$, and aggregate atoms $aggr(B)$. An *aggregate program* is a (possibly infinite) set of rules. A *normal logic program* is a set of rules which does not contain aggregate atoms.

Definition 3. *A* monotone aggregate atom *is an aggregate atom* $\mathrm{R}(s, d)$ *such that either* R *is a monotone aggregate relation and* s *contains only positive weight literals, or* R *is an anti-monotone aggregate relation and* s *contains only negative weight literals. A* definite aggregate program *is a program which contains only positive literals and monotone aggregate atoms.*

3.3 Semantics of Aggregates

An interpretation for an aggregate program is defined as the set of atoms which are assigned the value true. The set of all interpretations is denoted with $\mathcal{I}$. It forms a complete lattice under the subset inclusion order. Satisfiability of a literal L by an interpretation I is defined in the usual way and denoted by $I \models L$.

For a set expression s and an interpretation I we denote with s^I the subset expression of s which contains only the literals which are true in I that is $s^I = \{L = w \in s \mid I \models L\}$. We now define an evaluation function for set expressions as $[\![s]\!]_I = w(s^I)$ that is the multiset of weights of all literals in s which are true in I. Satisfiability of an aggregate atom $\mathrm{R}(s, d)$ is defined as follows: $I \models \mathrm{R}(s, d)$ if and only if $([\![s]\!]_I, d) \in \mathrm{R}$. For example, $\{b\} \models \mathrm{SUM}_{\geq}(\{a = 1, b = 2\}, 2)$ while $\{a\} \not\models \mathrm{SUM}_{\geq}(\{a = 1, b = 2\}, 2)$.

With every logic program with aggregates P we can associate an immediate consequence operator T_P^{aggr} in an obvious way.

Definition 4. $T_P^{aggr}(I) = \{A \mid A \leftarrow B \in P \text{ and } I \models B\}$

For definite aggregate programs this operator is monotone.

Proposition 1. *If P is a definite aggregate program then the T_P^{aggr} is monotone.*

Example 1 (Party Invitation). A set of people S are invited to a party. A person $p \in S$ accepts the invitation if and only if at least k_p of his friends also attend the party. With every person $p \in S$ we associate an atom p denoting whether p accepts the invitation. Let $F_p = \{q_1, \ldots, q_{n_p}\} \subseteq S$ be the set of friends of p. For every person p we have the following rule:

$$p \leftarrow \text{CARD}_{\geq}(\{q_1 = 1, \ldots, q_{n_p} = 1\}, k_p).$$

We note that $\text{CARD}_{\geq}$ is a monotone aggregate relation and there are no negative literals in the program and in the set expression so the program is a definite aggregate program. Consequently, the T_P^{aggr} operator is monotone.

Consider two people a and b such that a will accept the invitation if and only if b does and vice versa. The precise input is $S = \{a, b\}$, $F_a = \{b\}$ and $F_b = \{a\}$, and $k_a = k_b = 1$. The program is the following:

$$a \leftarrow \text{CARD}_{\geq}(\{b = 1\}, 1).$$
$$b \leftarrow \text{CARD}_{\geq}(\{a = 1\}, 1).$$

The least fixpoint of the T_P^{aggr} operator is the interpretation in which all atoms are false. That is neither a nor b will attend the party. □

3.4 Partial Stable Models

We now recall the definition of the partial stable semantics [11] of aggregate programs. It is defined using Approximation Theory so we have to define a partial approximating operator of T_P^{aggr}. We do this by extending Fitting's Φ_P operator for standard logic programs [7] to aggregate programs. As a consequence the semantics is an extension of the partial stable semantics of logic programs [12]. The Φ_P operator is defined by evaluating the bodies of the rules in three-valued logic. Our goal is to extend this function to aggregate atoms.

First, we extend the evaluation function $[\![s]\!]_I$ of set expressions to partial interpretations as follows:

$$[\![s]\!]_{(I_1,I_2)} = (w(s^{I_1}), w(s^{I_2})).$$

The result of $[\![s]\!]_{(I_1,I_2)}$ is a partial multiset of the form (M_1, M_2), i.e. M_1 and M_2 are multisets such that $M_1 \subseteq M_2$. For example, consider the partial interpretation $(I_1, I_2) = (\emptyset, \{p, q\})$ and the set expression $s = \{p = 1, q = 1\}$. Then $[\![s]\!]_{(I_1,I_2)} = (\emptyset, M)$ where M is a multiset containing two times 1 and no other elements, i.e. $M(1) = 2$ and $M(x) = 0$ for all $x \neq 1$.

In three-valued logic, aggregates are interpreted as partial relations which take partial multisets as input. In [11] we introduced a whole framework for defining semantics of aggregate programs. Any interpretation of an aggregate symbol satisfying certain properties gives rise to a different semantics. We also proposed the semantics obtained by taking the most precise approximating aggregate, called *ultimate approximating aggregate*.

Definition 5. *Let* $\mathrm{R} \subseteq \mathcal{M}(D) \times D$ *be an aggregate relation. The* ultimate approximating aggregate *of* R *is a function* $\bar{\mathrm{R}} : \mathcal{M}(D)^c \times D \to \mathcal{THREE}$*. It is defined as* $\mathrm{R}((M_1, M_2), d) = (\bar{R}^1((M_1, M_2), d), \bar{\mathrm{R}}^2((M_1, M_2), d))$ *where* $\bar{\mathrm{R}}^1, \bar{\mathrm{R}}^2 : \mathcal{M}(D)^c \times D \to \mathcal{TWO}$ *are the projections of* $\bar{\mathrm{R}}$ *on the first and second component, defined as follows:*

$$\bar{\mathrm{R}}^1((M_1, M_2), d) = \mathbf{t} \textit{ if and only if } \forall M \in [M_1, M_2] : (M, d) \in \mathrm{R}$$
$$\bar{\mathrm{R}}^2((M_1, M_2), d) = \mathbf{t} \textit{ if and only if } \exists M \in [M_1, M_2] : (M, d) \in \mathrm{R}$$

The next step is to define a valuation function for aggregate formulas in three-valued logic. For convenience, we view a partial interpretation (I_1, I_2) as a function $\tilde{I} : At \to \mathcal{THREE}$ defined as $\tilde{I}(a) = (I_1(a), I_2(a))$.

Definition 6 (Partial valuation function).

$$\begin{array}{ll}
\mathcal{H}_{\tilde{I}}(A) & = \tilde{I}(A), \textit{ for an atom } A \\
\mathcal{H}_{\tilde{I}}(\mathrm{R}(s,d)) & = \bar{\mathrm{R}}([\![s]\!]_{\tilde{I}}, d), \textit{ for an aggregate atom } \mathrm{R}(s,d) \\
\mathcal{H}_{\tilde{I}}(\neg F) & = \neg \mathcal{H}_{\tilde{I}}(F) \\
\mathcal{H}_{\tilde{I}}(F \wedge G) & = \mathcal{H}_{\tilde{I}}(F) \wedge_t \mathcal{H}_{\tilde{I}}(G) \\
\mathcal{H}_{\tilde{I}}(F \vee G) & = \mathcal{H}_{\tilde{I}}(F) \vee_t \mathcal{H}_{\tilde{I}}(G)
\end{array}$$

Based on $\mathcal{H}_{\tilde{I}}$ we define a three-valued immediate consequence operator Φ_P^{aggr} for aggregate programs.

Definition 7. $\Phi_P(\tilde{I}) = \tilde{J}$ *where* $\tilde{J}(A) = \bigvee_t \{\mathcal{H}_{\tilde{I}}(B) \mid A \leftarrow B \in P\}$.

For logic programs without aggregates the Φ_P^{aggr} operator coincides with Fitting's Φ_P operator [7] and the Ψ_P operator defined by Przymusinski [12]. Partial stable models of programs with aggregates are defined as the partial stable fixpoints of Φ_P^{aggr}.

Example 2. Reconsider the program from Example 1:

$$a \leftarrow \mathrm{CARD}_{\geq}(\{b = 1\}, 1).$$
$$b \leftarrow \mathrm{CARD}_{\geq}(\{a = 1\}, 1).$$

It has a two-valued well-founded model in which both a and b are false. This model is also equal to the least fixpoint of T_P^{aggr} and to the single exact stable model. □

4 Translation to Normal Logic Program

In this section we present a translation of an aggregate program to a normal logic program which preserves the set of partial stable models. Then we define the class of stratified aggregate programs and show that they have two-valued well-founded models.

For two formulas F and G we denote that they are equivalent in two valued logic with $F = G$ and that they are equivalent in three-valued logic with $F =_3 G$, that is $\mathcal{H}_{\tilde{I}}(F) = \mathcal{H}_{\tilde{I}}(G)$ for any partial interpretation $\tilde{I}$. Because two-valued interpretations are also partial interpretations equivalence in three-valued logic implies equivalence in two-valued logic. The opposite is not true. This is because there are no tautologies in three-valued logic. For example, $p \wedge (q \vee \neg q) = p$ but $p \wedge (q \vee \neg q) \neq_3 p$.

The translation $tr(\text{R}(s,d))$ of an aggregate atom $\text{R}(s,d)$ is a propositional formula F in disjunctive normal form — $\bigvee_{i \in I} F_i$ where I is an index set. Every disjunct F_i is a conjunction of literals using only atoms from the set expression s and $F_i \models \text{R}(s,d)$. More precisely, the disjuncts $F^s_{(s_1,s_2)}$ of F are indexed by pairs (s_1,s_2) of subset expressions s_1 and s_2 of s such that $s_1 \subseteq s_2$. The set expression s_1 contains the literals which have to be true and $s - s_2$ contains the literals which have to be false:

$$F^s_{(s_1,s_2)} = \bigwedge\{L \mid L = w \in s_1\} \wedge \bigwedge\{\overline{L} \mid L = w \in (s - s_2)\}.$$

The translation of an aggregate atom $\text{R}(s,d)$ is defined as

$$tr(\text{R}(s,d)) = \bigvee\{F^s_{(s_1,s_2)} \mid s_1 \subseteq s_2 \subseteq s \text{ and } F^s_{(s_1,s_2)} \models \text{R}(s,d)\}$$

Proposition 2. *$A =_3 tr(A)$ for every aggregate atom A.*

Because set expressions are finite we can obtain a simpler translation if we consider only minimal (treating a disjunct as a set of literals) disjuncts $F^s_{(s_1,s_2)}$. Let trm denote this translation:

$$trm(\text{R}(s,d)) = \bigvee\{F^s_{(s_1,s_2)} \mid F^s_{(s_1,s_2)} \text{ is a minimal set of literals such that } s_1 \subseteq s_2 \subseteq s \text{ and } F^s_{(s_1,s_2)} \models \text{R}(s,d)\}$$

Example 3. Consider the program P consisting of the following rule:

$$a \leftarrow \text{SUM}_{\geq}(\{a = 1, b = 2\}, 2).$$

The translation $tr(P)$ of P is the program

$$\begin{aligned} &a \leftarrow b. \\ &a \leftarrow a \wedge b. \\ &a \leftarrow \textit{not}\ a \wedge b. \end{aligned}$$

On the other hand, the translation $trm(P)$ is the program

$$a \leftarrow b.$$

□

Proposition 3. *$tr(A) =_3 trm(A)$ for every aggregate atom A.*

The translation of a program P is obtained by first translating all aggregate atoms in all rules from P. Then, we rewrite the body of every rule to disjunctive normal form and replace the rule with one rule for every disjunct. We denote the translation of a program P with $tr(P)$ or $trm(P)$ depending on which translation we use for aggregate atoms. The main result is an equivalence of the associated operators of the two programs.

Theorem 1. $\Phi_P^{aggr} = \Phi_{tr(P)} = \Phi_{trm(P)}$.

As a consequence the original and the translated programs have the same set of partial stable models.

We now look at the translation of definite aggregate programs.

Proposition 4. *If P is a definite aggregate program then $trm(P)$ is a definite normal program.*

Definite logic programs have a two-valued well-founded model which coincides with the single exact stable model. Consequently, this property also holds for definite aggregate programs.

Proposition 5. *A definite aggregate program has a least model which is equal to the single two-valued partial stable model and the well-founded model.*

More generally, we can define a stratification of a program with aggregates which corresponds to a stratification of its translation.

Definition 8 (Dependency Graph). *The* dependency graph *of a program P is a signed directed graph. Its nodes are the set of atoms of the program P. There is an edge from a to b if and only if there is a rule in P with a in the head and b in the body or in a set expression of an aggregate atom. The edge is* positive *if all occurrences of b in all rules for a are either in a positive literal, in a positive weight literal of a monotone aggregate relation, or in a negative weight literal of an anti-monotone aggregate relation. In all other cases the edge is* negative.

Definition 9 (Stratified Aggregate Program). *An aggregate program is* stratified *if the dependency graph does not have a cycle containing a negative edge.*

For normal logic programs, the notions of dependency graph and stratification coincide with the standard definitions [1]. The previous definition of stratification of aggregate programs [10] (also used in [5]) is more restrictive than ours and covers a smaller class of aggregate programs. The difference is that they do not differentiate between monotone, anti-monotone, and non-monotone aggregate relations.

Proposition 6. *If an aggregate program P is stratified then $trm(P)$ is stratified.*

For stratified logic programs without aggregates the well-founded model is two-valued and coincides with the perfect model and the single exact stable model. Consequently, stratified aggregate programs also have a two-valued well-founded model.

Example 4. Consider the program P consisting of the two rules:

$$a \leftarrow \text{CARD}_{\geq}(\{a = 1\}, 1).$$
$$b \leftarrow \text{CARD}_{\leq}(\{a = 1\}, 0).$$

This program is stratified according to Definition 9. but it is not stratified according to [2, 10]. Also, the associated T_P^{aggr} operator is non-monotone.

The translations $tr(P)$ and $trm(P)$ are both equal to

$$a \leftarrow a.$$
$$b \leftarrow not\ a.$$

which is a stratified logic program with a two-valued well-founded model $\{b\}$. □

We finish the section with a a remark that our definitions of definite and stratified aggregate programs do not cover the entire class of aggregate programs which are translated to definite (resp. stratified) programs.

Example 5. Consider the program P consisting of the single rule:

$$a \leftarrow \text{SUM}_{\geq}(\{not\ a = -2, b = 3\}, 0).$$

Because the set expression of the aggregate in the body contains both positive and negative numbers, the $\text{SUM}_{\geq}$ aggregate is non-monotone. So, the program is neither definite nor stratified. However, the translation of the aggregate is $trm(\text{SUM}_{\geq}(\{not\ a = -2, b = 3\}, 0)) = a \vee b$ and of the program is $trm(P) =$

$$a \leftarrow a.$$
$$a \leftarrow b.$$

which is a definite normal program. Using an algebraic transformation of SUM derived aggregates [14] which is also valid under our semantics, the aggregate atom is equivalent to $\text{SUM}_{\geq}(\{a = 2, b = 3\}, 2)$ and the program P has the same set of partial stable models as the program P':

$$a \leftarrow \text{SUM}_{\geq}(\{a = 2, b = 3\}, 2).$$

which is now a definite aggregate program. □

5 Proofs

To show that for definite aggregate programs the T_P^{aggr} is monotone (Proposition 1) we only need to show that monotone aggregate atoms are monotone with respect to interpretations.

Lemma 1. *If* $\text{R}(s, d)$ *is a monotone aggregate atom and* I *and* J *are interpretations such that* $I \subseteq J$ *then* $I \models \text{R}(s, d)$ *implies* $J \models \text{R}(s, d)$.

Proof. First, consider then case when s contains only positive weight literals and R is monotone. Then $[\![s]\!]_I \subseteq [\![s]\!]_J$ and consequently $([\![s]\!]_I, d) \in \text{R}$ implies $([\![s]\!]_J, d) \in \text{R}$. Similarly, if s contains only negative weight literals then $[\![s]\!]_I \supseteq [\![s]\!]_J$. Consequently $([\![s]\!]_I, d) \in \text{R}$ implies $([\![s]\!]_J, d) \in \text{R}$ because R is an anti-monotone aggregate relation

Before proving Proposition 2 we give an alternative characterization of $\text{F}^s_{(s_1,s_2)} \models \text{R}(s, d)$ in terms of the ultimate approximating aggregate $\bar{\text{R}}$ of R.

Lemma 2. $\mathrm{F}^s_{(s_1,s_2)} \models \mathrm{R}(s,d)$ *if and only if for all* $M \in [w(s_1), w(s_2)] : (M,d) \in \mathrm{R}$ *if and only if* $((w(s_1), w(s_2)), d) \in \bar{\mathrm{R}}^1$.

Proposition 2. $A =_3 tr(A)$ *for every aggregate atom* A.

Proof. We have to prove equivalence for the two components $\mathcal{H}^1$ and $\mathcal{H}^2$ of $\mathcal{H}$ and for each component we have to prove two directions. Let A be an aggregate atom of the form $\mathrm{R}(s,d)$. Fix a partial interpretation $\tilde{I}$ and let $(s_1, s_2) = s^{\tilde{I}}$ and $(M_1, M_2) = (w(s_1), w(s_2))$. In the proof we treat $tr(A)$ as a set of formulas.

($\mathcal{H}^1$, $\Rightarrow$) Suppose that $\mathcal{H}^1_{\tilde{I}}(A) = \mathbf{t}$. This implies that $(w(s_1), w(s_2), d) \in \bar{\mathrm{R}}^1$ and by Lemma 2 $F^s_{(s_1,s_2)} \models \mathrm{R}(s,d)$. So, $F^s_{(s_1,s_2)} \in tr(A)$. It is also the case that $\mathcal{H}^1_{\tilde{I}}(F^s_{(s_1,s_2)}) = t$ and consequently $\mathcal{H}^1_{\tilde{I}}(F) = \mathbf{t}$.

($\mathcal{H}^1$, $\Leftarrow$) Suppose that $\mathcal{H}^1_{\tilde{I}}(tr(A)) = \mathbf{t}$. This means that there is a disjunct $F^s_{(s'_1,s'_2)} \in tr(A)$ such that $\mathcal{H}^1_{\tilde{I}}(F^s_{(s'_1,s'_2)}) = \mathbf{t}$. This means that $(s'_1, s'_2) \leq_p (s_1, s_2)$ and so $[s_1, s_2] \subseteq [s'_1, s'_2]$. By definition of $tr(A)$, $F^s_{(s'_1,s'_2)} \in tr(A)$ only if for all $M \in [w(s'_1), w(s'_2)]$, $(M,d) \in \mathrm{R}$. Because $[s_1, s_2] \subseteq [s'_1, s'_2]$ then also for all $M \in [w(s_1), w(s_2)] = [M_1, M_2]$, $(M,d) \in \mathrm{R}$ and hence $\mathcal{H}^1_{\tilde{I}}(A) = \mathbf{t}$.

($\mathcal{H}^2$, $\Rightarrow$) Assume that $\mathcal{H}^2_{\tilde{I}}(\mathrm{R}(s,d)) = \mathbf{t}$. This means that there exists $M \in [M_1, M_2]$ such that $(M,d) \in \mathrm{R}$. Let $s' \in [s_1, s_2]$ be the set expression such that $w(s') = M$. This means that $F^s_{(s',s')} \in tr(A)$ and it has the form

$$F^s_{(s',s')} = \bigwedge_{L=w\in s_1} L \wedge \bigwedge_{L=w\in s-s_2} \overline{L} \wedge \bigwedge_{L=w\in s_2-s_1} \overset{(-)}{L}$$

where the sign of the literals in the third group depend on whether $L = w \in s'$. The important thing is that all literals in the first and second group are true in $\tilde{I}$ and all literals in the third group are undefined in $\tilde{I}$. So $\mathcal{H}^2_{\tilde{I}}(F^s_{(s',s')}) = \mathbf{t}$. Consequently $\mathcal{H}^2_{\tilde{I}}(tr(A)) = \mathbf{t}$.

($\mathcal{H}^2$, $\Leftarrow$) Suppose that $\mathcal{H}^2_{\tilde{I}}(A) = \mathbf{f}$. This means that there does not exist a multiset $M \in [M_1, M_2]$ such that $(M,d) \in \mathrm{R}$ which is equivalent to

$$\forall M \in [w(s_1), w(s_2)].\ (M,d) \notin \mathrm{R}. \tag{1}$$

Assume also that $\mathcal{H}^2_{\tilde{I}}(tr(A)) = \mathbf{t}$. This means that there exists a disjunct $F^s_{(s'_1,s'_2)} \in tr(A)$ such that $\mathcal{H}^2_{\tilde{I}}(F^s_{(s'_1,s'_2)}) = \mathbf{t}$. By definition of $tr(A)$ for this disjunct we have that $(w(s'_1), w(s'_2), d) \in \bar{\mathrm{R}}^1$ or $\forall M' \in [w(s'_1), w(s'_2)].\ (M', d) \in \mathrm{R}$. Together with (1) this implies

$$[s_1, s_2] \cap [s'_1, s'_2] = \emptyset. \tag{2}$$

The next step is to show the following:

$$\neg(s'_1 \subseteq s_2) \vee \neg(s_1 \subseteq s'_2) \tag{3}$$

Assume the opposite, i.e. $s'_1 \subseteq s_2$ and $s_1 \subseteq s'_2$ and let $s' = s_1 \cup s'_1$. Because $s_1 \subseteq s_2$ and $s'_1 \subseteq s_2$ then also $s' \subseteq s_2$. Similarly, $s' \subseteq s'_2$. Thus, $s' \in [s_1, s_2]$ and $s' \in [s'_1, s'_2]$ which is a contradiction with (2).

Finally, we do a case analysis of (3). Suppose that $s'_1 \not\subseteq s_2$. This means that there exists a weight literal $L = w \in s'_1$ such that $L = w \notin s_2$. For this literal we have $\mathcal{H}^2_{\tilde{I}}(L) = \mathbf{f}$ and consequently $\mathcal{H}^2_{\tilde{I}}(F^s_{(s'_1,s'_2)}) = \mathbf{f}$ which is a contradiction. Now, consider the case when $s_1 \not\subseteq s'_2$. This means that there exists a weight literal $L = w \in s_1$ such that $L = w \notin s'_2$. Note that $L = w \in (s - s'_2)$ and for this literal $\mathcal{H}^1_{\tilde{I}}(L) = \mathbf{t}$ and consequently $\mathcal{H}^2_{\tilde{I}}(\overline{L}) = \mathbf{f}$. Again, we arrive at a contradiction $\mathcal{H}^2_{\tilde{I}}(F^s_{(s'_1,s'_2)}) = \mathbf{t}$. $\square$

Proposition 3. *$tr(A) =_3 trm(A)$ for every aggregate atom A.*

Proof (Sketch). Fix a partial interpretation $\tilde{I}$. If $F^s_{(s_1,s_2)}, F^s_{(s'_1,s'_2)} \in tr(A)$ such that $(s_1, s_2) \leq_p (s'_1, s'_2)$ then $\mathcal{H}_{\tilde{I}}(F^s_{(s_1,s_2)}) \geq_t \mathcal{H}_{\tilde{I}}(F^s_{(s'_1,s'_2)})$. Consequently, removing the disjunct $F^s_{(s'_1,s'_2)}$ from $tr(A)$ will not affect its truth value. $\square$

The proofs of Proposition 4 and Proposition 6 are based on the following two lemmas.

Lemma 3. *Let $\mathrm{R}(s,d)$ be an aggregate atom where R is a monotone aggregate relation. Then*

$$trm(\mathrm{R}(s,d)) = \bigvee\{F^s_{(s_1,s)} \mid F^s_{(s_1,s)} \text{ is a minimal set of literals such that } s_1 \subseteq s \text{ and } F^s_{(s_1,s)} \models \mathrm{R}(s,d)\}.$$

As a consequence, all weight literals in the set expression s which are used in the translation keep their sign.

Lemma 4. *Let $\mathrm{R}(s,d)$ be an aggregate atom where R is an anti-monotone aggregate relation. Then*

$$trm(\mathrm{R}(s,d)) = \bigvee\{F^s_{(\emptyset,s_2)} \mid F^s_{(\emptyset,s_2)} \text{ is a minimal set of literals such that } s_2 \subseteq s \text{ and } F^s_{(\emptyset,s_2)} \models \mathrm{R}(s,d)\}.$$

As a consequence, all weight literals in the set expression s which are used in the translation change their sign.

6 Related Work

A closely related work is the extension of the stable semantics to programs with weight constraint rules [14]. A *weight constraint* is an expression of the form $l \leq s \leq u$ where s is a set expression and l and u are real numbers or one of the symbols $-\infty, +\infty$. In our syntax such expression corresponds to the formula $\mathrm{SUM}_{\geq}(s,l) \wedge \mathrm{SUM}_{\leq}(s,u)$. The precise relationship between the stable semantics of weight constraint rules and exact stable models of aggregate programs as defined in [11] and in the present paper has been studied in [11]. We have shown that for weight constraints without upper bound, i.e. $u = +\infty$, the two semantics coincide. However, for weight constraints with upper bound, the two semantics may be different.

Example 6. Consider the weight constraint program $P_{wc} = \{a \leftarrow \{not\ a\} \leq 0\}$. Intuitively, the rule expresses the fact that a is true if $not\ a$ is false or equivalently, a is true if a is true. The aggregate program corresponding to P_{wc} is $P = \{a \leftarrow \text{CARD}_{\leq}(\{not\ a = 1\}, 0).\}$. According to Definition 3 this is a definite aggregate program with a monotone T_P^{aggr} operator. Its least fixpoint is the empty set. By Proposition 5 this is also the single stable model. This can also be seen by the translation to a normal logic program which is $tr(\text{CARD}_{\leq}(\{not\ a = 1\}, 0)) = a$ and $tr(P) = \{a \leftarrow a.\}$.

However, under the semantics of weight constraints a rule with an upper bound is translated to a rule with a lower bound by introducing an intermediate atom:

$$a \leftarrow not\ b.$$
$$b \leftarrow 1 \leq \{not\ a = 1\}.$$

This program is equivalent to the program $P' = \{a \leftarrow not\ b.\ b \leftarrow not\ a.\}$ which has two stable models $\{a\}$ and $\{b\}$. So the stable models of the original program are $\{a\}$ and $\emptyset$, contrary to intuition. $\square$

A translation of weight constraints to nested expressions which preserves the set of answer sets is given in [6]. The semantics of weight constraints and consequently the translation of [6] is defined only for set expressions with non-negative weights. For multisets of such numbers, the aggregate relation $\text{SUM}_{\geq}$ is monotone. For weight constraints of the form $l \leq s$, the translation of [6] is exactly the same as trm. However, because of the different semantics of weight constraints with upper bounds the translation which we give and the one from [6] are different.

7 Conclusion

If a stratified aggregate program has a two-valued well-founded model according to the semantics of [11] then any other semantics of aggregate programs which is more precise (according to Approximation Theory) also has a two-valued well-founded model. In particular this holds for the ultimate semantics of aggregate programs [5].

Our definition of stratification can also be applied to other languages and semantics in which one can make a distinction between monotone, anti-monotone, and non-monotone aggregates. For example weight constraints with lower bounds [14] are monotone[1] while weight constraints with upper bounds are anti-monotone.

In this paper the proof that stratified aggregate programs have a two-valued well-founded model was done by a translation to a normal logic program. We believe that it is possible to define standard model of stratified aggregate programs in the same way as [1] and perfect model in the same way as [13] thus giving a semantics of stratified aggregate programs which is independent of [11].

References

1. K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann, 1988.

[1] Restricted only to positive weights.

2. T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in dlv. In *Proc. of the 18th Int. Joint Conference on Artificial Intelligence*, 2003.
3. M. Denecker, V. Marek, and M. Truszczyński. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In J. Minker, editor, *Logic-based Artificial Intelligence*, pages 127–144. Kluwer Academic Publishers, 2000.
4. M. Denecker, V. Marek, and M. Truszczyńsky. Ultimate approximations in nonmonotonic knowledge representation systems. In *Principles of Knowledge Representation and Reasoning*, pages 177–188. Morgan Kaufmann, 2002.
5. M. Denecker, N. Pelov, and M. Bruynooghe. Ultimate well-founded and stable model semantics for logic programs with aggregates. In P. Codognet, editor, *Int. Conf. on Logic Programming*, volume 2237 of *LNCS*, pages 212–226. Springer, 2001.
6. P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 2003. To appear.
7. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
8. M. Gelfond. Representing knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *LNCS*, pages 413–451. Springer, 2002.
9. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming, Proc. of the 5th International Conference and Symposium*, pages 1070–1080. MIT Press, 1988.
10. I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The magic of duplicates and aggregates. In *16th Int. Conf. on Very Large Data Bases*, pages 264–277, 1990.
11. N. Pelov, M. Denecker, and M. Bruynooghe. Partial stable semantics for logic programs with aggregates. In *LPNMR*, 2004. In review. Avaliable at http://www.cs.kuleuven.ac.be/∼pelov/papers/lpnmr7.ps.gz.
12. T. Przymusinksi. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–464, 1990.
13. T. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann, 1988.
14. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.