Rewriting recursive aggregates in answer set programming: back to monotonicity

MARIO ALVIANO
University of Calabria, Italy

WOLFGANG FABER
University of Huddersfield, UK

MARTIN GEBSER
Aalto University, HIIT, Finland

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Aggregation functions are widely used in answer set programming for representing and reasoning on knowledge involving sets of objects collectively. Current implementations simplify the structure of programs in order to optimize the overall performance. In particular, aggregates are rewritten into simpler forms known as monotone aggregates. Since the evaluation of normal programs with monotone aggregates is in general on a lower complexity level than the evaluation of normal programs with arbitrary aggregates, any faithful translation function must introduce disjunction in rule heads in some cases. However, no function of this kind is known. The paper closes this gap by introducing a polynomial, faithful, and modular translation for rewriting common aggregation functions into the simpler form accepted by current solvers. A prototype system allows for experimenting with arbitrary recursive aggregates, which are also supported in the recent version 4.5 of the grounder GRINGO, using the methods presented in this paper.

KEYWORDS: answer set programming; polynomial, faithful, and modular translation; aggregation functions.

1 Introduction

Answer set programming (ASP) is a declarative language for knowledge representation and reasoning (Brewka et al. 2011). In ASP knowledge is encoded by means of logic rules, possibly using disjunction and default negation, interpreted according to the stable model semantics (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991). Since its first proposal, the basic language was extended by several constructs in order to ease the representation of practical knowledge, and particular interest was given to aggregate functions (Simons et al. 2002; Liu et al. 2010; Bartholomew et al. 2011; Faber et al. 2011; Ferraris 2011; Gelfond and Zhang 2014). In fact, aggregates allow for expressing properties on sets of atoms declaratively, and are widely used for example to enforce *functional dependencies*, where a rule of the form

$$\bot \leftarrow R'(\overline{X}),\ \textsc{count}[\overline{Y} : R(\overline{X}, \overline{Y}, \overline{Z})] \leq 1$$

constrains relation R to satisfy the functional dependency $\overline{X} \rightarrow \overline{Y}$, where $\overline{X} \cup \overline{Y} \cup \overline{Z}$ is the set of attributes of R, and R' is the projection of R on $\overline{X}$.

Among the several semantics proposed for interpreting ASP programs with aggregates, two of them (Faber et al. 2011; Ferraris 2011) are implemented in widely-used ASP solvers (Faber et al. 2008; Gebser et al. 2012). The two semantics agree for programs without negated aggregates, and are thus referred indistinctly in this paper as F-stable model semantics. It is important to observe that the implementation of F-stable model semantics is *incomplete* in current ASP solvers. In fact, the grounding phase rewrites aggregates into simpler forms known as *monotone* aggregates, and many common reasoning tasks on normal programs with monotone aggregates belong to the first level of the polynomial hierarchy, while in general they belong to the second level for normal programs with aggregates (Faber et al. 2011; Ferraris 2011). Since disjunction is not introduced during the rewriting of aggregates, this is already evidence that currently available rewritings can be correct only if recursion is limited to *convex* aggregates (Liu and Truszczyński 2006), the largest class of aggregates for which the common reasoning tasks still belong to the first level of the polynomial hierarchy in the normal case (Alviano and Faber 2013).

However, non-convex aggregations may arise in several contexts while modeling complex knowledge (Eiter et al. 2008; Eiter et al. 2012; Abseher et al. 2014). A minimalistic example is provided by the Σ_2^P-complete problem called *Generalized Subset Sum* (Berman et al. 2002), where two vectors u and v of integers as well as an integer b are given, and the task is to decide whether the formula $\exists x \forall y(ux + vy \neq b)$ is true, where x and y are vectors of binary variables of the same length as u or v, respectively. For example, for $u = [1, 2]$, $v = [2, 3]$, and $b = 5$, the task is to decide whether the following formula is true: $\exists x_1 x_2 \forall y_1 y_2 (1 \cdot x_1 + 2 \cdot x_2 + 2 \cdot y_1 + 3 \cdot y_2 \neq 5)$. Any natural encoding of such an instance would include an aggregate of the form $\text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$, and it is not immediate how to obtain an equivalent program that comprises monotone aggregates only.

The aim of this paper is to overcome the limitations of current rewritings in order to provide a *polynomial, faithful, and modular* translation (Janhunen 2006) that allows to compile logic programs with aggregates into equivalent logic programs that only comprise monotone aggregates. The paper focuses on common aggregation functions such as SUM, AVG, MIN, MAX, COUNT, EVEN, and ODD. Actually, all of them are mapped to possibly non-monotone sums in Section 3.1, and non-monotonicity is then eliminated in Section 3.2. The rewriting is further optimized in Section 3.3 by taking strongly connected components of a refined version of the positive dependency graph into account. Crucial properties like correctness and modularity are established in Section 3.4, followed by the discussion of related work and conclusions. The proposed rewriting is implemented in a prototype system (`http://alviano.net/software/f-stable-models/`), and is also adopted in the recent version 4.5 of the grounder GRINGO. With the prototype, aggregates are represented by reserved predicates, so that the grounding phase can be delegated to DLV (Alviano et al. 2010) or GRINGO (Gebser et al. 2011). The output of a grounder is then processed to properly encode aggregates for the subsequent stable model search, as performed by CLASP (Gebser et al. 2012), CMODELS (Giunchiglia et al. 2006), or WASP (Alviano et al. 2014).

2 Background

Let $\mathcal{V}$ be a set of propositional atoms including $\bot$. A propositional literal is an atom possibly preceded by one or more occurrences of the *negation as failure* symbol $\sim$. An aggregate literal, or simply aggregate, is of one of the following three forms:

$$\text{AGG}_1[w_1 : l_1, \ldots, w_n : l_n] \odot b \qquad \text{COUNT}[l_1, \ldots, l_n] \odot b \qquad \text{AGG}_2[l_1, \ldots, l_n] \quad (1)$$

where $\text{AGG}_1 \in \{\text{SUM}, \text{AVG}, \text{MIN}, \text{MAX}\}$, $\text{AGG}_2 \in \{\text{EVEN}, \text{ODD}\}$, $n \geq 0$, $b, w_1, \ldots, w_n$ are integers, $l_1, \ldots, l_n$ are propositional literals, and $\odot \in \{<, \leq, \geq, >, =, \neq\}$. (Note that $[w_1 : l_1, \ldots, w_n : l_n]$ and $[l_1, \ldots, l_n]$ are multisets. This notation of propositional aggregates differs from ASP-Core-2 (`https://www.mat.unical.it/aspcomp2013/ASPStandardization/`) for ease of presentation.) A literal is either a propositional literal, or an aggregate. A rule r is of the following form:

$$p_1 \vee \cdots \vee p_m \leftarrow l_1 \wedge \cdots \wedge l_n \quad (2)$$

where $m \geq 1$, $n \geq 0$, $p_1, \ldots, p_m$ are propositional atoms, and $l_1, \ldots, l_n$ are literals. The set $\{p_1, \ldots, p_m\} \setminus \{\bot\}$ is referred to as head, denoted by $H(r)$, and the set $\{l_1, \ldots, l_n\}$ is called body, denoted by $B(r)$. A program Π is a finite set of rules. The set of propositional atoms (different from $\bot$) occurring in a program Π is denoted by $At(\Pi)$, and the set of aggregates occurring in Π is denoted by $Ag(\Pi)$.

Example 1
Consider the following program Π_1:

$$x_1 \leftarrow {\sim\sim} x_1 \qquad x_2 \leftarrow {\sim\sim} x_2 \qquad y_1 \leftarrow \mathit{unequal} \qquad y_2 \leftarrow \mathit{unequal} \qquad \bot \leftarrow {\sim}\mathit{unequal}$$
$$\mathit{unequal} \leftarrow \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$$

As will be clarified after defining the notion of a stable model, Π_1 encodes the instance of Generalized Subset Sum introduced in Section 1. ■

An *interpretation* I is a set of propositional atoms such that $\bot \notin I$. Relation $\models$ is inductively defined as follows:

- for $p \in \mathcal{V}$, $I \models p$ if $p \in I$;
- $I \models {\sim}l$ if $I \not\models l$;
- $I \models \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \odot b$ if $\sum_{i \in [1..n], I \models l_i} w_i \odot b$;
- $I \models \text{AVG}[w_1 : l_1, \ldots, w_n : l_n] \odot b$ if $m := |\{i \in [1..n] \mid I \models l_i\}|$, $m \geq 1$, and $\sum_{i \in [1..n], I \models l_i} \frac{w_i}{m} \odot b$;
- $I \models \text{MIN}[w_1 : l_1, \ldots, w_n : l_n] \odot b$ if $\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) \odot b$;
- $I \models \text{MAX}[w_1 : l_1, \ldots, w_n : l_n] \odot b$ if $\max(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{-\infty\}) \odot b$;
- $I \models \text{COUNT}[l_1, \ldots, l_n] \odot b$ if $|\{i \in [1..n] \mid I \models l_i\}| \odot b$;
- $I \models \text{EVEN}[l_1, \ldots, l_n]$ if $|\{i \in [1..n] \mid I \models l_i\}|$ is an even number;
- $I \models \text{ODD}[l_1, \ldots, l_n]$ if $|\{i \in [1..n] \mid I \models l_i\}|$ is an odd number;
- for a rule r of the form (2), $I \models B(r)$ if $I \models l_i$ for all $i \in [1..n]$, and $I \models r$ if $H(r) \cap I \neq \emptyset$ when $I \models B(r)$;
- for a program Π, $I \models \Pi$ if $I \models r$ for all $r \in \Pi$.

For any expression π, if $I \models \pi$, we say that I is a *model* of π, I satisfies π, or π is true in I. In the following, $\top$ will be a shorthand for ${\sim}\bot$, i.e., $\top$ is a literal true in all interpretations.

Example 2
Continuing with Example 1, the models of Π_1, restricted to the atoms in $At(\Pi_1)$, are X, $X \cup \{x_1\}$, $X \cup \{x_2\}$, and $X \cup \{x_1, x_2\}$, where $X = \{\mathit{unequal}, y_1, y_2\}$. ■

The *reduct* of a program Π with respect to an interpretation I is obtained by removing rules with false bodies and by fixing the interpretation of all negative literals. More formally, the following function is inductively defined:

- for $p \in \mathcal{V}$, $F(I, p) := p$;
- $F(I, {\sim}l) := \top$ if $I \not\models l$, and $F(I, {\sim}l) := \bot$ otherwise;
- $F(I, \text{AGG}_1[w_1 : l_1, \ldots, w_n : l_n] \odot b) := \text{AGG}_1[w_1 : F(I, l_1), \ldots, w_n : F(I, l_n)] \odot b$;
- $F(I, \text{COUNT}[l_1, \ldots, l_n] \odot b) := \text{COUNT}[F(I, l_1), \ldots, F(I, l_n)] \odot b$;
- $F(I, \text{AGG}_2[l_1, \ldots, l_n]) := \text{AGG}_2[F(I, l_1), \ldots, F(I, l_n)]$;
- for a rule r of the form (2), $F(I, r) := p_1 \vee \cdots \vee p_m \leftarrow F(I, l_1) \wedge \cdots \wedge F(I, l_n)$;
- for a program Π, $F(I, \Pi) := \{F(I, r) \mid r \in \Pi, I \models B(r)\}$.

Program $F(I, \Pi)$ is the reduct of Π with respect to I. An interpretation I is a *stable model* of a program Π if $I \models \Pi$ and there is no $J \subset I$ such that $J \models F(I, \Pi)$. Let $SM(\Pi)$ denote the set of stable models of Π. Two programs Π, Π' are equivalent with respect to a context $V \subseteq \mathcal{V}$, denoted $\Pi \equiv_V \Pi'$, if both $|SM(\Pi)| = |SM(\Pi')|$ and $\{I \cap V \mid I \in SM(\Pi)\} = \{I \cap V \mid I \in SM(\Pi')\}$. An aggregate A is *monotone* (in program reducts) if $J \models F(I, A)$ implies $K \models F(I, A)$, for all $J \subseteq K \subseteq I \subseteq \mathcal{V}$, and it is *convex* (in program reducts) if $J \models F(I, A)$ and $L \models F(I, A)$ implies $K \models F(I, A)$, for all $J \subseteq K \subseteq L \subseteq I \subseteq \mathcal{V}$; when either property applies, $I \models A$ and $J \models F(I, A)$ yield $K \models F(I, A)$, for all $J \subseteq K \subseteq I$.

Example 3
Continuing with Example 2, the only stable model of Π_1 is $\{x_1, \mathit{unequal}, y_1, y_2\}$. Indeed, the reduct $F(\{x_1, \mathit{unequal}, y_1, y_2\}, \Pi_1)$ is

$$\begin{array}{lll} x_1 \leftarrow \top & y_1 \leftarrow \mathit{unequal} & y_2 \leftarrow \mathit{unequal} \\ \multicolumn{3}{l}{\mathit{unequal} \leftarrow \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5} \end{array}$$

and no strict subset of $\{x_1, \mathit{unequal}, y_1, y_2\}$ is a model of the above program. On the other hand, the reduct $F(\{x_2, \mathit{unequal}, y_1, y_2\}, \Pi_1)$ is

$$\begin{array}{lll} x_2 \leftarrow \top & y_1 \leftarrow \mathit{unequal} & y_2 \leftarrow \mathit{unequal} \\ \multicolumn{3}{l}{\mathit{unequal} \leftarrow \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5} \end{array}$$

and $\{x_2, y_2\}$ is a model of the above program. Similarly, it can be checked that $\{\mathit{unequal}, y_1, y_2\}$ and $\{x_1, x_2, \mathit{unequal}, y_1, y_2\}$ are not stable models of Π_1. Further note that the aggregate $\text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$ is non-convex. The aggregate is also recursive, or not stratified, a notion that will be formalized later in Section 3.3. ■

3 Compilation

Current ASP solvers (as opposed to grounders) only accept a limited set of aggregates, essentially aggregates of the form (1) such that AGG_1 is SUM, $b, w_1, \ldots, w_n$ are non-negative integers, and $\odot$ is $\geq$. The corresponding class of programs will be referred to as LPARSE-like programs. Hence, compilations from the general language are required. More formally, what is needed is a polynomial-time computable function associating every program Π with an LPARSE-like program Π' such that $\Pi \equiv_{At(\Pi)} \Pi'$. To define such a translation is nontrivial, and indeed most commonly used rewritings that are correct in the stratified case are unsound for recursive aggregates.

Example 4

Consider program Π_1 from Example 1 and the following program Π_2, often used as an intermediate step to obtain an LPARSE-like program:

$$x_1 \leftarrow {\sim}{\sim}x_1 \qquad x_2 \leftarrow {\sim}{\sim}x_2 \qquad y_1 \leftarrow \mathit{unequal} \qquad y_2 \leftarrow \mathit{unequal} \qquad \bot \leftarrow {\sim}\mathit{unequal}$$
$$\mathit{unequal} \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5$$
$$\mathit{unequal} \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] < 5$$

The two programs only minimally differ: the last rule of Π_1 is replaced by two rules in Π_2, following the intuition that the original aggregate is true in an interpretation I if and only if either $I \models \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5$ or $I \models \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] < 5$. However, the two programs are not equivalent. Indeed, it can be checked that Π_2 has no stable model, and in particular $\{x_1, \mathit{unequal}, y_1, y_2\}$ is not stable because $F(\{x_1, \mathit{unequal}, y_1, y_2\}, \Pi_2)$ is

$$x_1 \leftarrow \top \qquad y_1 \leftarrow \mathit{unequal} \qquad y_2 \leftarrow \mathit{unequal}$$
$$\mathit{unequal} \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5$$

and $\{x_1\}$ is one of its models. ■

Also replacing negative integers may change the semantics of programs.

Example 5

Let $\Pi_3 := \{p \leftarrow \textsc{sum}[1 : p, -1 : q] \geq 0,\ p \leftarrow q,\ q \leftarrow p\}$. Its only stable model is $\{p, q\}$. The negative integer is usually removed by means of a rewriting adapted from pseudo-Boolean constraint solvers, which replaces each element $w : l$ in (1) such that $w < 0$ by $-w : {\sim}l$, and also adds $-w$ to b. The resulting program in the example is $\{p \leftarrow \textsc{sum}[1 : p, 1 : {\sim}q] \geq 1, p \leftarrow q,\ q \leftarrow p\}$, which has no stable models. ■

Actually, stable models cannot be preserved in general by rewritings such as those hinted in the above examples unless the polynomial hierarchy collapses to its first level. In fact, while checking the existence of a stable model is Σ_2^P-complete for programs with atomic heads, this problem is in NP for LPARSE-like programs with atomic heads, and disjunction is necessary for modeling Σ_2^P-hard instances. It follows that, in order to be correct, a polynomial-time compilation must possibly introduce disjunction when rewriting recursive programs. This intuition is formalized in Section 3.2. Before, in Section 3.1, the structure of input programs is simplified by mapping all aggregates to conjunctions of sums, where comparison operators are either $>$ or $\neq$. While $>$ can be viewed as $\geq$ relative to an incremented bound $b + 1$, negative integers as well as $\neq$ constitute the remaining gap to LPARSE-like programs.

3.1 Mapping to sums

The notion of strong equivalence (Lifschitz et al. 2001; Turner 2003; Ferraris 2011) will be used in this section. Let $\pi := l_1 \wedge \cdots \wedge l_n$ be a conjunction of literals, for some

$n \geq 0$. A pair (J, I) of interpretations such that $J \subseteq I$ is an *SE-model* of π if $I \models \pi$ and $J \models F(I, l_1) \wedge \cdots \wedge F(I, l_n)$. Two conjunctions π, π' are *strongly equivalent*, denoted by $\pi \equiv_{SE} \pi'$, if they have the same SE-models. Strong equivalence means that replacing π by π' preserves the stable models of any logic program.

Proposition 1 (*Lifschitz et al. 2001; Turner 2003; Ferraris 2011*)
Let π, π' be two conjunctions of literals such that $\pi \equiv_{SE} \pi'$. Let Π be a program, and Π' be the program obtained from Π by replacing any occurrence of π by π'. Then, it holds that $\Pi \equiv_{\mathcal{V}} \Pi'$ (where $\mathcal{V}$ is the set of all propositional atoms).

The following strong equivalences can be proven by showing equivalence with respect to models, and by noting that $\sim$ is neither introduced nor eliminated:

(A) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] < b \equiv_{SE} \textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] > -b$
(B) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \leq b \equiv_{SE} \textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] > -b - 1$
(C) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \geq b \equiv_{SE} \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] > b - 1$
(D) $\textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] = b \equiv_{SE} \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] > b - 1 \wedge$
$\textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] > -b - 1$

For instance, given an interpretation I, (A) is based on the fact that $\sum_{i \in [1..n], I \models l_i} w_i < b$ if and only if $\sum_{i \in [1..n], I \models l_i} -w_i > -b$, so that $I \models \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] < b$ if and only if $I \models \textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] > -b$. Similar observations apply to (B)–(D), and strong equivalences as follows hold for further aggregates:

(E) $\textsc{avg}[w_1 : l_1, \ldots, w_n : l_n] \odot b \equiv_{SE} \textsc{sum}[w_1 - b : l_1, \ldots, w_n - b : l_n] \odot 0 \wedge$
$\textsc{sum}[1 : l_1, \ldots, 1 : l_n] > 0$
(F) $\textsc{min}[w_1 : l_1, \ldots, w_n : l_n] < b \equiv_{SE} \textsc{sum}[1 : l_i \mid i \in [1..n], w_i < b] > 0$
(G) $\textsc{min}[w_1 : l_1, \ldots, w_n : l_n] \leq b \equiv_{SE} \textsc{sum}[1 : l_i \mid i \in [1..n], w_i \leq b] > 0$
(H) $\textsc{min}[w_1 : l_1, \ldots, w_n : l_n] \geq b \equiv_{SE} \textsc{sum}[-1 : l_i \mid i \in [1..n], w_i < b] > -1$
(I) $\textsc{min}[w_1 : l_1, \ldots, w_n : l_n] > b \equiv_{SE} \textsc{sum}[-1 : l_i \mid i \in [1..n], w_i \leq b] > -1$
(J) $\textsc{min}[w_1 : l_1, \ldots, w_n : l_n] = b \equiv_{SE} \textsc{sum}[1 - n \cdot (b - w_i) : l_i \mid i \in [1..n], w_i \leq b] > 0$
(K) $\textsc{min}[w_1 : l_1, \ldots, w_n : l_n] \neq b \equiv_{SE} \textsc{sum}[n \cdot (b - w_i) - 1 : l_i \mid i \in [1..n], w_i \leq b] > -1$
(L) $\textsc{max}[w_1 : l_1, \ldots, w_n : l_n] \odot b \equiv_{SE} \textsc{min}[-w_1 : l_1, \ldots, -w_n : l_n]\ f(\odot)\ -b$
where $< \overset{f}{\mapsto} >, \leq \overset{f}{\mapsto} \geq, \geq \overset{f}{\mapsto} \leq, > \overset{f}{\mapsto} <, = \overset{f}{\mapsto} =$, and $\neq \overset{f}{\mapsto} \neq$
(M) $\textsc{count}[l_1, \ldots, l_n] \odot b \equiv_{SE} \textsc{sum}[1 : l_1, \ldots, 1 : l_n] \odot b$
(N) $\textsc{even}[l_1, \ldots, l_n] \equiv_{SE} \bigwedge_{i \in [1..\lceil n/2 \rceil]} \textsc{sum}[1 : l_1, \ldots, 1 : l_n] \neq 2 \cdot i - 1$
(O) $\textsc{odd}[l_1, \ldots, l_n] \equiv_{SE} \bigwedge_{i \in [0..\lfloor n/2 \rfloor]} \textsc{sum}[1 : l_1, \ldots, 1 : l_n] \neq 2 \cdot i$

Given a program Π, the successive application of (A)–(O), from the last to the first, gives an equivalent program Π' whose aggregates are sums with comparison operators $>$ and $\neq$.

Example 6
Let $\Pi_4 := \{p \vee q \leftarrow,\ p \leftarrow \textsc{avg}[5 : p, 3 : p, 2 : q, 7 : q] \geq 4\}$. By applying (E), the aggregate becomes $\textsc{sum}[1 : p, -1 : p, -2 : q, 3 : q] \geq 0 \wedge \textsc{sum}[1 : p, 1 : p, 1 : q, 1 : q] > 0$, and an application of (C) yields $\textsc{sum}[1 : p, -1 : p, -2 : q, 3 : q] > -1 \wedge \textsc{sum}[1 : p, 1 : p, 1 : q, 1 : q] > 0$. Simplifying the latter expression leads to the program $\Pi_4' := \{p \vee q \leftarrow, p \leftarrow \textsc{sum}[1 : q] > -1 \wedge \textsc{sum}[2 : p, 2 : q] > 0\}$. Note that $\{p\}$ is the unique stable model of both Π_4 and Π_4', so that $\Pi_4 \equiv_{\{p,q\}} \Pi_4'$. ■

3.2 Eliminating non-monotone aggregates

The structure of input programs can be further simplified by eliminating non-monotone aggregates. Without loss of generality, we hereinafter assume aggregates to be of the form

$$\textsc{sum}[w_1 : l_1, \dots, w_n : l_n] \odot b \tag{3}$$

such that $\odot \in \{>, \neq\}$. For A of the form (3), by $Lit(A) := \{l_1, \dots, l_n\} \setminus \{\bot\}$, we refer to the set of propositional literals (different from $\bot$) occurring in A. Moreover, let $\Sigma(l, A) := \sum_{i\in[1..n], l_i=l} w_i$ denote the weight of any $l \in Lit(A)$. We write $wLit^*(A) := [\Sigma(l, A) : l \mid l \in Lit(A), \Sigma(l, A) \neq 0]$, $wLit^+(A) := [\Sigma(l, A) : l \mid l \in Lit(A), \Sigma(l, A) > 0]$, and $wLit^-(A) := [\Sigma(l, A) : l \mid l \in Lit(A), \Sigma(l, A) < 0]$ to distinguish the (multi)sets of literals associated with non-zero, positive, or negative weights, respectively, in A. For instance, letting $A := \textsc{sum}[1 : p, -1 : p, -2 : q, 3 : q] > -1$, we have that $wLit^*(A) = wLit^+(A) = [1 : q]$ and $wLit^-(A) = []$. In the following, we call an aggregate A of the form (3) non-monotone if $\{p \in \mathcal{V} \mid (w : p) \in wLit^-(A)\} \neq \emptyset$, or if $\odot$ is $\neq$, thus disregarding special cases in which A would still be monotone or convex. (The rewritings presented below are correct also in such cases, but they do not exploit the particular structure of an aggregate for avoiding the use of disjunction in rule heads.)

For an aggregate A of the form (3) such that $\odot$ is $>$ and a set $V \subseteq \mathcal{V}$ of atoms, we define a rule with a fresh propositional atom aux as head and a monotone aggregate as body by:

$$aux \leftarrow \textsc{sum}\begin{pmatrix} wLit^+(A) \cup \\ [-w : p^F \mid (w : p) \in wLit^-(A), p \in V] \cup \\ [-w : {\sim}l \mid (w : l) \in wLit^-(A), l \notin V] \end{pmatrix} > b - \sum_{(w:l)\in wLit^-(A)} w \tag{4}$$

Note that (4) introduces a fresh, hidden propositional atom p^F (Eiter et al. 2005; Janhunen and Niemelä 2012) for any $p \in V$ associated with a negative weight in A. However, when $V = \emptyset$, every $(w : l) \in wLit^-(A)$ is replaced by $-w : {\sim}l$, thus rewarding the falsity of l rather than penalizing l, which is in turn compensated by adding $-w$ to the bound b; such a replacement preserves models (Simons et al. 2002), but in general not stable models (Ferraris and Lifschitz 2005). By $pos(A, V)$, we denote the program including rule (4) along with the following rules for every $p \in V$ such that $(w : p) \in wLit^-(A)$:

$$p^F \leftarrow {\sim}p \tag{5}$$

$$p^F \leftarrow aux \tag{6}$$

$$p \vee p^F \leftarrow {\sim}{\sim}aux \tag{7}$$

Intuitively, any atom p^F introduced in $pos(A, V)$ must be true whenever p is false, but also when aux is true, so to implement the concept of *saturation* (Eiter and Gottlob 1995). Rules (5) and (6) encode such an intuition. Moreover, rule (7) guarantees that at least one of p and p^F belongs to any model of reducts obtained from interpretations I containing aux. In fact, p^F represents the falsity of p in the reduct of rule (4) with respect to I in order to test the satisfaction of the monotone aggregate in (4) relative to subsets of I. For a program Π, the rewriting $rew(\Pi, A, V)$ is the union of $pos(A, V)$ and the program obtained from Π by replacing any occurrence of A by aux. That is, $rew(\Pi, A, V)$ eliminates a (possibly) non-monotone aggregate A with comparison operator $>$ in favor of a monotone aggregate and disjunction within the subprogram $pos(A, V)$. In this section, we further rely on

$V = \mathcal{V}$, i.e., saturation is applied to all atoms associated with negative weights in A, while a refinement based on positive dependencies will be provided in the next section.

Example 7
Consider Π_3 from Example 5 whose first rule is strongly equivalent to $p \leftarrow \textsc{sum}[1 : p, -1 : q] > -1$. For $A := \textsc{sum}[1 : p, -1 : q] > -1$, the program $pos(A, \mathcal{V})$ is as follows:

$$aux \leftarrow \textsc{sum}[1 : p, 1 : q^F] > 0 \qquad q^F \leftarrow {\sim}q \qquad q^F \leftarrow aux \qquad q \vee q^F \leftarrow {\sim}{\sim}aux$$

Moreover, we obtain $rew(\Pi_3, A, \mathcal{V}) = pos(A, \mathcal{V}) \cup \{p \leftarrow aux,\ p \leftarrow q,\ q \leftarrow p\}$ as the full rewriting of Π_3 for A and $\mathcal{V}$. One can check that no strict subset of $\{p, q, aux, q^F\}$ is a model of $rew(\Pi_3, A, \mathcal{V})$ or the reduct $F(\{p, q, aux, q^F\}, rew(\Pi_3, A, \mathcal{V}))$, respectively, where the latter includes $q \vee q^F \leftarrow \top$. In fact, $SM(rew(\Pi_3, A, \mathcal{V})) = \{\{p, q, aux, q^F\}\}$ and $SM(\Pi_3) = \{\{p, q\}\}$ yield that $\Pi_3 \equiv_{\{p,q\}} rew(\Pi_3, A, \mathcal{V})$. ■

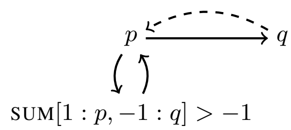
We further extend the rewriting to an aggregate $A := \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] \neq b$ by considering two cases based on splitting A into $A_> := \textsc{sum}[w_1 : l_1, \ldots, w_n : l_n] > b$ and $A_< := \textsc{sum}[-w_1 : l_1, \ldots, -w_n : l_n] > -b$. While $A_>$ is true in any interpretation I such that $\sum_{i\in[1..n], I\models l_i} w_i > b$, in view of the strong equivalence given in (A), I satisfies $A_<$ if and only if $\sum_{i\in[1..n], I\models l_i} w_i < b$. For a program Π and $V \subseteq \mathcal{V}$, we let $pos(A, V) := pos(A_>, V) \cup pos(A_<, V)$, and the rewriting $rew(\Pi, A, V)$ is the union of $pos(A, V)$ and the program obtained from Π by replacing any occurrence of A by aux, where the fresh propositional atom aux serves as the head of rules of the form (4) in both $pos(A_>, V)$ and $pos(A_<, V)$. Note that $pos(A, V)$ also introduces fresh propositional atoms p^F for any $p \in V$ such that $(w : p) \in wLit^*(A)$. Again, an atom p^F represents the falsity of p in the reduct of rule (4) from either $pos(A_>, V)$ or $pos(A_<, V)$ with respect to interpretations I containing aux, which allows for testing the satisfaction of monotone counterparts of $A_>$ and $A_<$ relative to subsets of I.

Example 8
Consider program Π_1 from Example 1, and let $A := \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$. Then, we obtain the following rewriting $rew(\Pi_1, A, \mathcal{V})$:

$$\begin{array}{llll}
x_1 \leftarrow {\sim}{\sim}x_1 & x_2 \leftarrow {\sim}{\sim}x_2 & y_1 \leftarrow unequal & y_2 \leftarrow unequal \\
\bot \leftarrow {\sim}unequal & \multicolumn{3}{l}{aux \leftarrow \textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5} \\
unequal \leftarrow aux & \multicolumn{3}{l}{aux \leftarrow \textsc{sum}[1 : x_1^F, 2 : x_2^F, 2 : y_1^F, 3 : y_2^F] > 3} \\
x_1^F \leftarrow {\sim}x_1 & x_2^F \leftarrow {\sim}x_2 & y_1^F \leftarrow {\sim}y_1 & y_2^F \leftarrow {\sim}y_2 \\
x_1^F \leftarrow aux & x_2^F \leftarrow aux & y_1^F \leftarrow aux & y_2^F \leftarrow aux \\
x_1 \vee x_1^F \leftarrow {\sim}{\sim}aux & x_2 \vee x_2^F \leftarrow {\sim}{\sim}aux & y_1 \vee y_1^F \leftarrow {\sim}{\sim}aux & y_2 \vee y_2^F \leftarrow {\sim}{\sim}aux
\end{array}$$

The only stable model of $rew(\Pi_1, A, \mathcal{V})$ is $\{x_1, unequal, y_1, y_2, aux, x_1^F, x_2^F, y_1^F, y_2^F\}$. In particular, note that $x_1 \leftarrow \top$, $x_2^F \leftarrow \top$, $y_1 \vee y_1^F \leftarrow \top$, and $y_2 \vee y_2^F \leftarrow \top$ belong to the reduct, and any choice between y_1 and y_1^F as well as y_2 and y_2^F leads to the satisfaction of $\textsc{sum}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5$ or $\textsc{sum}[1 : x_1^F, 2 : x_2^F, 2 : y_1^F, 3 : y_2^F] > 3$ along with saturation. As a consequence, $\Pi_1 \equiv_{\{x_1, x_2, unequal, y_1, y_2\}} rew(\Pi_1, A, \mathcal{V})$. ■

The subprogram $pos(A, V)$ for A of the form (3) such that $\odot \in \{>, \neq\}$ and $V \subseteq \mathcal{V}$ is LPARSE-like. Moreover, the rewriting $rew(\Pi, A, V)$ can be iterated to eliminate all non-monotone aggregates A from Π. Thereby, it is important to note that fresh propositional



$$\Pi_3 = \left\{ \begin{array}{l} p \leftarrow \text{SUM}[1 : p, -1 : q] > -1 \\ p \leftarrow q \\ q \leftarrow p \end{array} \right. \left.\vphantom{\begin{array}{l} p \\ p \end{array}}\right\} = \Pi_3'$$

Fig. 1. Dependency graphs considered in Example 9: the dashed arc belongs to $\mathcal{G}_{\Pi_3}$, but not to $\mathcal{G}_{\Pi_3'}$.

atoms introduced in $pos(A_1, V_1)$ and $pos(A_2, V_2)$ for $A_1 \neq A_2$ are distinct. As hinted in the above examples, $rew(\Pi, A, \mathcal{V})$ preserves stable models of Π, which extends to an iterated elimination of aggregates. Before formalizing respective properties in Section 3.4, however, we refine $rew(\Pi, A, V)$ to subsets V of $\mathcal{V}$ based on positive dependencies in Π.

3.3 Refined rewriting

Given a program Π such that all aggregates in $Ag(\Pi)$ are of the form (3) for $\odot \in \{>, \neq\}$, the (positive) *dependency graph* $\mathcal{G}_\Pi$ of Π consists of the vertices $At(\Pi) \cup Ag(\Pi)$ and (directed) arcs (α, β) if either of the following conditions holds for $\alpha, \beta \in At(\Pi) \cup Ag(\Pi)$:

- there is a rule $r \in \Pi$ such that $\alpha \in H(r)$ and $\beta \in B(r)$;
- $\alpha \in Ag(\Pi)$ is of the form (3) such that $\odot$ is $>$ and $(w : \beta) \in wLit^+(\alpha)$;
- $\alpha \in Ag(\Pi)$ is of the form (3) such that $\odot$ is $\neq$ and $(w : \beta) \in wLit^*(\alpha)$.

That is, $\mathcal{G}_\Pi$ includes arcs from atoms in $H(r)$ to positive literals in $B(r)$ for rules $r \in \Pi$, and from aggregates $A \in Ag(\Pi)$ to atoms associated with a positive or non-zero weight in A if the comparison operator of A is $>$ or $\neq$, respectively. A strongly connected component of $\mathcal{G}_\Pi$, also referred to as *component* of Π, is a maximal subset C of $At(\Pi) \cup Ag(\Pi)$ such that any $\alpha \in C$ reaches each $\beta \in C$ via a path in $\mathcal{G}_\Pi$. The set of propositional atoms in the component of Π containing an aggregate $A \in Ag(\Pi)$ is denoted by $rec(\Pi, A)$ (or $rec(\Pi, A) := \emptyset$ when $A \notin Ag(\Pi)$). Then, the rewriting $rew(\Pi, A, rec(\Pi, A))$ restricts saturation for fresh propositional atoms p^F introduced in $pos(A, rec(\Pi, A))$ to atoms $p \in rec(\Pi, A)$ occurring in A.

Example 9

The dependency graph of program Π_3 from Example 5 is shown in Fig. 1, where the first rule of Π_3 is identified with $p \leftarrow \text{SUM}[1 : p, -1 : q] > -1$. Let A denote the aggregate $\text{SUM}[1 : p, -1 : q] > -1$. First of all, note that there is no arc connecting A to q because $(w : q) \notin wLit^+(A)$. However, A reaches q in $\mathcal{G}_{\Pi_3}$ via p, and since also q reaches A via p, we have that $rec(\Pi_3, A) = \{p, q\}$, and thus $rew(\Pi_3, A, rec(\Pi_3, A)) = rew(\Pi_3, A, \mathcal{V})$.

Now consider $\Pi_3' := \Pi_3 \setminus \{q \leftarrow p\}$, whose dependency graph is obtained by removing arc (q, p) from $\mathcal{G}_{\Pi_3}$, i.e., the dashed arc in Fig. 1. Note that q does not reach A in $\mathcal{G}_{\Pi_3'}$, and therefore $rec(\Pi_3', A) = \{p\}$. In this case, $rew(\Pi_3', A, rec(\Pi_3', A)) = \{aux \leftarrow \text{SUM}[1 : p, 1 : {\sim}q] > 0, p \leftarrow aux,\ p \leftarrow q\}$, where $SM(rew(\Pi_3', A, rec(\Pi_3', A))) = \{\{p, aux\}\}$ and $SM(\Pi_3') = \{\{p\}\}$ yield that $\Pi_3' \equiv_{\{p,q\}} rew(\Pi_3', A, rec(\Pi_3', A))$. ■

Example 10

Program Π_1 from Example 1 has the components $\{x_1\}$, $\{x_2\}$, and $\{unequal, y_1, y_2, A\}$ for $A := \text{SUM}[1 : x_1; 2 : x_2; 2 : y_1; 3 : y_2] \neq 5$. Thus, $rew(\Pi_1, A, rec(\Pi_1, A))$ comprises the following rules:

$$
\begin{array}{llll}
x_1 \leftarrow {\sim}{\sim}x_1 & x_2 \leftarrow {\sim}{\sim}x_2 & y_1 \leftarrow unequal & y_2 \leftarrow unequal \\
\bot \leftarrow {\sim}unequal & aux \leftarrow \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5 & & \\
unequal \leftarrow aux & aux \leftarrow \text{SUM}[1 : {\sim}x_1, 2 : {\sim}x_2, 2 : y_1^F, 3 : y_2^F] > 3 & & \\
\quad y_1^F \leftarrow {\sim}y_1 & \quad y_2^F \leftarrow {\sim}y_2 & & \\
\quad y_1^F \leftarrow aux & \quad y_2^F \leftarrow aux & & \\
y_1 \vee y_1^F \leftarrow {\sim}{\sim}aux & y_2 \vee y_2^F \leftarrow {\sim}{\sim}aux & &
\end{array}
$$

In contrast to $rew(\Pi_1, A, \mathcal{V})$ in Example 8, x_1 and x_2 are mapped to ${\sim}x_1$ and ${\sim}x_2$, rather than x_1^F and x_2^F, in the rule $aux \leftarrow \text{SUM}[1 : {\sim}x_1, 2 : {\sim}x_2, 2 : y_1^F, 3 : y_2^F] > 3$ from $pos(\text{SUM}[-1 : x_1, -2 : x_2, -2 : y_1, -3 : y_2] > -5, rec(\Pi_1, A))$. Hence, the reduct of $rew(\Pi_1, A, rec(\Pi_1, A))$ with respect to $\{x_1, unequal, y_1, y_2, aux, y_1^F, y_2^F\}$ includes $aux \leftarrow \text{SUM}[1 : \bot, 2 : \top, 2 : y_1^F, 3 : y_2^F] > 3$ as well as $x_1 \leftarrow \top$, $y_1 \vee y_1^F \leftarrow \top$, and $y_2 \vee y_2^F \leftarrow \top$. As a consequence, any model containing y_1^F or y_2^F entails aux, and $aux \leftarrow \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5$ yields aux when y_1 and y_2 are both true. In fact, $\{x_1, unequal, y_1, y_2, aux, y_1^F, y_2^F\}$ is the only stable model of $rew(\Pi_1, A, rec(\Pi_1, A))$, so that $\Pi_1 \equiv_{\{x_1, x_2, unequal, y_1, y_2\}} rew(\Pi_1, A, rec(\Pi_1, A))$. ■

The above examples illustrate that saturation can be restricted to atoms p sharing the same component of Π with an aggregate A, where a fresh propositional atom p^F is introduced in $pos(A, rec(\Pi, A))$ when p has a negative or non-zero weight in A, depending on whether the comparison operator of A is $>$ or $\neq$, respectively. That is, the refined rewriting uses disjunction only if A is a recursive non-monotone aggregate. In turn, when A is non-recursive or stratified (Faber et al. 2011), the corresponding subprogram $pos(A, \emptyset)$ does not introduce disjunction or any fresh propositional atom different from aux.

3.4 Properties

Our first result generalizes a property of models of reducts to programs with aggregates.

Proposition 2

Let Π be a program, I be a model of Π, and $J \subset I$ be a model of $F(I, \Pi)$. Then, there is some component C of Π such that $I \cap (C \setminus J) \neq \emptyset$ and $I \setminus (C \setminus J) \models F(I, \Pi)$.

In other words, when any strict subset J of a model I of Π satisfies $F(I, \Pi)$, then there is a model K of $F(I, \Pi)$ such that $J \subseteq K \subset I$ and $I \setminus K \subseteq C$ for some component C of Π. For instance, the model $\{x_1\}$ of $F(\{x_1, unequal, y_1, y_2\}, \Pi_2)$, given in Example 4, is such that $\{x_1, unequal, y_1, y_2\} \setminus \{x_1\} \subseteq C$ for the component $C := \{unequal, y_1, y_2, \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] > 5\}$ of Π_2.

For a program Π and A of the form (3) such that $\odot \in \{>, \neq\}$, rewritings $rew(\Pi, A, \mathcal{V})$ and $rew(\Pi, A, rec(\Pi, A))$ have been investigated above. In order to establish their correctness, we show that $\Pi \equiv_{At(\Pi)} rew(\Pi, A, V)$ holds for all subsets V of $\mathcal{V}$ such that $rec(\Pi, A) \subseteq V$. To this end, let $At^F(A, V) := \{p^F \mid (p^F \leftarrow aux) \in pos(A, V)\}$ denote the fresh, hidden atoms p^F introduced in $pos(A, V)$. Given an interpretation I (such that $I \cap (\{aux\} \cup At^F(A, V)) = \emptyset$) and $J \subseteq I$, we define an *extension* of J relative to I by:

$$ext(J, I) := \begin{cases} J \cup \{p^F \in At^F(A, V) \mid p \notin I\} & \text{if } I \not\models A \\ J \cup \{p^F \in At^F(A, V) \mid p \notin J\} & \text{if } I \models A \text{ and } J \not\models F(I, A) \\ J \cup \{aux\} \cup At^F(A, V) & \text{if } I \models A \text{ and } J \models F(I, A) \end{cases}$$

For instance, considering $A := \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$, $V := \{unequal, y_1, y_2\}$, $I := \{x_2, unequal, y_1, y_2\}$, and $J := \{x_2, y_2\}$, in view of $I \models A$ and $J \not\models F(I, A)$, we obtain $ext(I, I) = I \cup \{aux, y_1^F, y_2^F\}$ and $ext(J, I) = J \cup \{y_1^F\}$.

For I and J as above, the following technical lemma yields $ext(I, I)$ as the subset-minimal model of reducts $F(I', pos(A, V))$ with respect to models I' of the subprogram $pos(A, V)$ that extend I. Under the assumption that a nonempty difference $I \setminus J$ remains local to a component C of Π such that some atom in C depends on A, $ext(J, I)$ further constitutes the subset-minimal extension of J to a model of $F(ext(I, I), pos(A, V))$.

Lemma 1
Let Π be a program, A be an aggregate, and V be a set of propositional atoms such that $rec(\Pi, A) \subseteq V$. Let I be an interpretation such that $I \cap (\{aux\} \cup At^F(A, V)) = \emptyset$ and $J \subseteq I$. Then, the following conditions hold:

1. For any model I' of $pos(A, V)$ such that $I' \setminus (\{aux\} \cup At^F(A, V)) = I$, we have that $ext(I, I) \subseteq I'$ and $ext(I, I) \models F(I', pos(A, V))$.
2. If $J = I$ or $I \setminus J \subseteq C$ for some component C of Π such that there is a rule $r \in \Pi$ with $H(r) \cap C \neq \emptyset$ and $A \in B(r)$, then $ext(J, I) \models F(ext(I, I), pos(A, V))$ and $ext(J, I) \subseteq J'$ for any model J' of $F(ext(I, I), pos(A, V))$ such that $J' \setminus (\{aux\} \cup At^F(A, V)) = J$.

With the auxiliary result describing the formation of models of $pos(A, V)$ and its reducts at hand, we can show the main result of this paper that the presented rewritings preserve the stable models of a program Π.

Theorem 1
Let Π be a program, A be an aggregate, and V be a set of propositional atoms such that $rec(\Pi, A) \subseteq V$. Then, it holds that $\Pi \equiv_{At(\Pi)} rew(\Pi, A, V)$.

The second objective is establishing the properties of a *polynomial, faithful, and modular* translation (Janhunen 2006), i.e., a mapping that is polynomial-time computable, preserves stable models (when auxiliary atoms are ignored), and can be computed independently on parts of an input program. The faithfulness of $rew(\Pi, A, V)$ for any $rec(\Pi, A) \subseteq V \subseteq \mathcal{V}$ is stated in Theorem 1. Moreover, since at most $3 \cdot n$ additional rules (5)–(7) are introduced in $pos(A, V)$ for A of the form (3), it is clear that $rew(\Pi, A, V)$ is polynomial-time computable. This also holds when applying the strong equivalences (A)–(O) to replace aggregates by conjunctions, where the worst cases (N) and (O) yield a quadratic blow-up.

Hence, the final condition to be addressed is modularity. Given that the refined rewriting $rew(\Pi, A, rec(\Pi, A))$ refers to the components of an entire program Π, this rewriting cannot be done in parts. The unoptimized rewriting $rew(\Pi, A, \mathcal{V})$, however, consists of the subprogram $pos(A, \mathcal{V})$, which is independent of Π, and otherwise merely replaces A by aux in Π. Thus, under the assumption that A does not occur outside of Π (where it cannot be replaced by aux), $rew(\Pi, A, \mathcal{V})$ complies with the modularity condition.

Proposition 3
Let Π, Π' be programs and A be an aggregate such that $A \notin Ag(\Pi')$. Then, it holds that $rew(\Pi \cup \Pi', A, \mathcal{V}) = rew(\Pi, A, \mathcal{V}) \cup \Pi'$.

Note that $A \notin Ag(\Pi')$ is not a restriction, given that an element $w : \bot$ with an arbitrary weight w can be added for obtaining a new aggregate A' that is strongly equivalent to A. In practice, however, one would rather aim at reusing a propositional atom aux that represents the satisfaction of A instead of redoing the rewriting with another fresh atom aux'.

4 Related work

Several semantics were proposed in the literature for interpreting ASP programs with aggregates. Among them, F-stable model semantics (Faber et al. 2011; Ferraris 2011) was considered in this paper because it is implemented by widely-used ASP solvers (Faber et al. 2008; Gebser et al. 2012). Actually, the definition provided in Section 2 is slightly different than those in (Faber et al. 2011; Ferraris 2011). In particular, the language considered in (Ferraris 2011) has a broader syntax allowing for arbitrary nesting of propositional formulas. The language considered in (Faber et al. 2011), instead, does not explicitly allow the use of double negation, which however can be simulated by means of auxiliary atoms. For example, in (Faber et al. 2011) a rule $p \leftarrow {\sim}{\sim}p$ must be modeled by using a fresh atom p^F and the following subprogram: $\{p \leftarrow {\sim}p^F,\ p^F \leftarrow {\sim}p\}$. Moreover, in (Faber et al. 2011) aggregates cannot contain negated literals, which can be simulated by auxiliary atoms as well. On the other hand, negated aggregates are permitted in (Faber et al. 2011), while they are not considered in this paper. Actually, programs with negated aggregates are those for which (Ferraris 2011) and (Faber et al. 2011) disagree. As a final remark, the reduct of (Faber et al. 2011) does not remove negated literals from satisfied bodies, which however are necessarily true in all counter-models because double negation is not allowed.

Techniques to rewrite logic programs with aggregates into equivalent programs with simpler aggregates were investigated in the literature right from the beginning (Simons et al. 2002). In particular, rewritings into LPARSE-like programs, which differ from those presented in this paper, were considered in (Liu and You 2013). As a general comment, since disjunction is not considered in (Liu and You 2013), all aggregates causing a jump from the first to the second level of the polynomial hierarchy are excluded a priori. This is the case for aggregates of the form $\textsc{sum}(S) \neq b$, $\textsc{avg}(S) \neq b$, and $\textsc{count}(S) \neq b$, as noted in (Son and Pontelli 2007), but also for comparators other than $\neq$ when negative weights are involved. In fact, in (Liu and You 2013) negative weights are eliminated by a rewriting similar to the one in (4), but negated literals are introduced instead of auxiliary atoms, which may lead to counterintuitive results (Ferraris and Lifschitz 2005). A different rewriting was presented in (Ferraris 2011), whose output are programs with nested expressions, a construct that is not supported by current ASP systems. Other relevant rewriting techniques were proposed in (Bomanson and Janhunen 2013; Bomanson et al. 2014), and proved to be quite efficient in practice. However, these rewritings produce aggregate-free programs preserving F-stable models only in the stratified case, or if recursion is limited to convex aggregates. On the other hand, it is interesting to observe that the rewritings of (Bomanson and Janhunen 2013; Bomanson et al. 2014) are applicable to the output of the rewritings presented in this paper in order to completely eliminate aggregates, thus preserving F-stable models in general.

The rewritings given in Section 3 do not apply to other semantics whose stability checks are not based on minimality (Pelov et al. 2007; Son and Pontelli 2007; Shen et al. 2014), or whose program reducts do not contain aggregates (Gelfond and Zhang 2014). They also disregard DL (Eiter et al. 2008) and HEX (Eiter et al. 2014) atoms, extensions of ASP for interacting with external knowledge bases, possibly expressed in different languages, that act semantically similar to aggregate functions.

As a final remark, the notion of a (positive) dependency graph given in Section 3.3 refines the concept of recursion through aggregates. In fact, many works (Alviano et al. 2011; Faber et al. 2011; Simons et al. 2002; Son and Pontelli 2007) consider an aggregate as recursive as soon as aggregated literals depend on the evaluation of the aggregate. According to this simple definition, the aggregate in the following program is deemed to be recursive: $\{p \leftarrow \textsc{sum}[-1 : q] > -1,\ q \leftarrow p\}$. However, (negative) recursion through a rule like $p \leftarrow {\sim}q$ is uncritical for the computation of F-stable models, as it cannot lead to circular support. In fact, the dependency graph introduced in Section 3.3 does not include arcs for such non-positive dependencies, so that strongly connected components render potential circular support more precisely. For example, the aforementioned program has three components, namely $\{p\}$, $\{q\}$, and $\{\textsc{sum}[-1 : q] > -1\}$. If the dependency of $\textsc{sum}[-1 : q] > -1$ on q were mistakenly considered as positive, the three components would be joined into one, thus unnecessarily extending the scope of stability checks.

5 Conclusion

The representation of knowledge in ASP is eased by the availability of several constructs, among them aggregation functions. As it is common in combinatorial problems, the structure of input instances is simplified in order to improve the efficiency of low-level reasoning. Concerning aggregation functions, the simplified form processed by current ASP solvers is known as monotone, and by complexity arguments faithfulness of current rewritings is subject to specific conditions, i.e., input programs can only contain convex aggregates. The (unoptimized) translation presented in this paper is instead polynomial, faithful, and modular for all common aggregation functions, including non-convex instances of SUM, AVG, and COUNT. Moreover, the rewriting approach extends to aggregation functions such as MIN, MAX, EVEN, and ODD. The proposed rewritings are implemented in a prototype system and also adopted in the recent version 4.5 of the grounder GRINGO.

Acknowledgement

Mario Alviano was partially supported by MIUR within project "SI-LAB BA2KNOW – Business Analitycs to Know", by Regione Calabria, POR Calabria FESR 2007-2013 within project "ITravel PLUS" and project "KnowRex", by the National Group for Scientific Computation (GNCS-INDAM), and by Finanziamento Giovani Ricercatori UNICAL. Martin Gebser was supported by AoF (grant #251170) within the Finnish Centre of Excellence in Computational Inference Research (COIN).

References

ABSEHER, M., BLIEM, B., CHARWAT, G., DUSBERGER, F., AND WOLTRAN, S. 2014. Computing secure sets in graphs using answer set programming. In *Seventh International Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2014)*, D. Inclezan and M. Maratea, Eds.

ALVIANO, M., CALIMERI, F., FABER, W., LEONE, N., AND PERRI, S. 2011. Unfounded sets and well-founded semantics of answer set programs with aggregates. *Journal of Artificial Intelligence Research 42*, 487–527.

ALVIANO, M., DODARO, C., AND RICCA, F. 2014. Anytime computation of cautious consequences in answer set programming. *Theory and Practice of Logic Programming 14,* 4-5, 755–770.

ALVIANO, M. AND FABER, W. 2013. The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In *Twelfth International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR 2013)*, P. Cabalar and T. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 67–72.

ALVIANO, M., FABER, W., LEONE, N., PERRI, S., PFEIFER, G., AND TERRACINA, G. 2010. The disjunctive Datalog system DLV. In *First International Workshop on Datalog 2.0 (Datalog Reloaded)*, O. de Moor, G. Gottlob, T. Furche, and A. Sellers, Eds. Lecture Notes in Computer Science, vol. 6702. Springer, 282–301.

BARTHOLOMEW, M., LEE, J., AND MENG, Y. 2011. First-order semantics of aggregates in answer set programming via modified circumscription. In *AAAI 2011 Spring Symposium on Logical Formalizations of Commonsense Reasoning (SS-11-06)*, E. Davis, P. Doherty, and E. Erdem, Eds. AAAI, 16–22.

BERMAN, P., KARPINSKI, M., LARMORE, L., PLANDOWSKI, W., AND RYTTER, W. 2002. On the complexity of pattern matching for highly compressed two-dimensional texts. *Journal of Computer and System Sciences 65,* 2, 332–350.

BOMANSON, J., GEBSER, M., AND JANHUNEN, T. 2014. Improving the normalization of weight rules in answer set programs. In *Fourteenth European Conference on Logics in Artificial Intelligence (JELIA 2014)*, E. Fermé and J. Leite, Eds. Lecture Notes in Computer Science, vol. 8761. Springer, 166–180.

BOMANSON, J. AND JANHUNEN, T. 2013. Normalizing cardinality rules using merging and sorting constructions. In *Twelfth International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR 2013)*, P. Cabalar and T. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 187–199.

BREWKA, G., EITER, T., AND TRUSZCZYŃSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

EITER, T., FINK, M., KRENNWALLNER, T., AND REDL, C. 2012. Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming 12,* 4-5, 659–679.

EITER, T., FINK, M., KRENNWALLNER, T., REDL, C., AND SCHÜLLER, P. 2014. Efficient HEX-program evaluation based on unfounded sets. *Journal of Artificial Intelligence Research 49*, 269–321.

EITER, T. AND GOTTLOB, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence 15,* 3-4, 289–323.

EITER, T., IANNI, G., LUKASIEWICZ, T., SCHINDLAUER, R., AND TOMPITS, H. 2008. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence 172,* 12-13, 1495–1539.

EITER, T., TOMPITS, H., AND WOLTRAN, S. 2005. On solution correspondences in answer set programming. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, L. Kaelbling and A. Saffiotti, Eds. Professional Book Center, 97–102.

FABER, W., PFEIFER, G., AND LEONE, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175,* 1, 278–298.

FABER, W., PFEIFER, G., LEONE, N., DELL'ARMI, T., AND IELPA, G. 2008. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming 8,* 5-6, 545–580.

FERRARIS, P. 2011. Logic programs with propositional connectives and aggregates. *ACM Transactions on Computational Logic 12,* 4, 25:1–25:44.

FERRARIS, P. AND LIFSCHITZ, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5,* 1-2, 45–74.

GEBSER, M., KAMINSKI, R., KÖNIG, A., AND SCHAUB, T. 2011. Advances in gringo series 3. In *Eleventh International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR 2011)*, J. Delgrande and W. Faber, Eds. Lecture Notes in Computer Science, vol. 6645. Springer, 345–351.

GEBSER, M., KAUFMANN, B., AND SCHAUB, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187/188*, 52–89.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Fifth International Conference and Symposium on Logic Programming (ICLP 1988)*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3-4, 365–385.

GELFOND, M. AND ZHANG, Y. 2014. Vicious circle principle and logic programs with aggregates. *Theory and Practice of Logic Programming 14,* 4-5, 587–601.

GIUNCHIGLIA, E., LIERLER, Y., AND MARATEA, M. 2006. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning 36,* 4, 345–377.

JANHUNEN, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics 16,* 1-2, 35–86.

JANHUNEN, T. AND NIEMELÄ, I. 2012. Applying visible strong equivalence in answer-set program transformations. In *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, E. Erdem, J. Lee, Y. Lierler, and D. Pearce, Eds. Lecture Notes in Computer Science, vol. 7265. Springer, 363–379.

LIFSCHITZ, V., PEARCE, D., AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2,* 4, 526–541.

LIU, G. AND YOU, J. 2013. Relating weight constraint and aggregate programs: Semantics and representation. *Theory and Practice of Logic Programming 13,* 1, 1–31.

LIU, L., PONTELLI, E., SON, T., AND TRUSZCZYŃSKI, M. 2010. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence 174,* 3-4, 295–315.

LIU, L. AND TRUSZCZYŃSKI, M. 2006. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research 27*, 299–334.

PELOV, N., DENECKER, M., AND BRUYNOOGHE, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming 7,* 3, 301–353.

SHEN, Y., WANG, K., EITER, T., FINK, M., REDL, C., KRENNWALLNER, T., AND DENG, J. 2014. FLP answer set semantics without circular justifications for general logic programs. *Artificial Intelligence 213*, 1–41.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

SON, T. AND PONTELLI, E. 2007. A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming 7,* 3, 355–375.

TURNER, H. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming 3,* 4-5, 609–622.

Appendix A Proofs

Proposition 4
The strong equivalences stated in (A)–(O) hold.

Proof
Let I be an interpretation. Given that the reducts with respect to I of expressions on both sides of $\equiv_{\mathcal{V}}$ in (A)–(O) fix the same (negative) literals, it is sufficient to show equivalence.

(A) $\mathrm{SUM}[w_1 : l_1, \dots, w_n : l_n] < b$ is true in I if and only if
$\sum_{i\in[1..n],I\models l_i} w_i < b$ if and only if
$\sum_{i\in[1..n],I\models l_i} -w_i > -b$ if and only if
$\mathrm{SUM}[-w_1 : l_1, \dots, -w_n : l_n] > -b$ is true in I.

(B) $\mathrm{SUM}[w_1 : l_1, \dots, w_n : l_n] \leq b$ is true in I if and only if
$\sum_{i\in[1..n],I\models l_i} w_i \leq b$ if and only if
$\sum_{i\in[1..n],I\models l_i} -w_i > -b-1$ if and only if
$\mathrm{SUM}[-w_1 : l_1, \dots, -w_n : l_n] > -b-1$ is true in I.

(C) $\mathrm{SUM}[w_1 : l_1, \dots, w_n : l_n] \geq b$ is true in I if and only if
$\sum_{i\in[1..n],I\models l_i} w_i \geq b$ if and only if
$\sum_{i\in[1..n],I\models l_i} w_i > b-1$ if and only if
$\mathrm{SUM}[w_1 : l_1, \dots, w_n : l_n] > b-1$ is true in I.

(D) $\mathrm{SUM}[w_1 : l_1, \dots, w_n : l_n] = b$ is true in I if and only if
$\sum_{i\in[1..n],I\models l_i} w_i = b$ if and only if
$\sum_{i\in[1..n],I\models l_i} w_i > b-1$ and $\sum_{i\in[1..n],I\models l_i} -w_i > -b-1$ if and only if
$\mathrm{SUM}[w_1 : l_1, \dots, w_n : l_n] > b-1 \wedge \mathrm{SUM}[-w_1 : l_1, \dots, -w_n : l_n] > -b-1$ is true in I.

(E) $\mathrm{AVG}[w_1 : l_1, \dots, w_n : l_n] \odot b$ is true in I if and only if
$m := |\{i \in [1..n] \mid I \models l_i\}|$, $m \geq 1$, and $\sum_{i\in[1..n],I\models l_i} \frac{w_i}{m} \odot b$ if and only if
$m := |\{i \in [1..n] \mid I \models l_i\}|$, $m \geq 1$, and $\sum_{i\in[1..n],I\models l_i} w_i \odot m \cdot b$ if and only if
$\{i \in [1..n] \mid I \models l_i\} \neq \emptyset$ and $\sum_{i\in[1..n],I\models l_i} (w_i - b) \odot 0$ if and only if
$\mathrm{SUM}[w_1 - b : l_1, \dots, w_n - b : l_n] \odot 0 \wedge \mathrm{SUM}[1 : l_1, \dots, 1 : l_n] > 0$ is true in I.

(F) $\mathrm{MIN}[w_1 : l_1, \dots, w_n : l_n] < b$ is true in I if and only if
$\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) < b$ if and only if
$\{i \in [1..n] \mid w_i < b, I \models l_i\} \neq \emptyset$ if and only if
$\mathrm{SUM}[1 : l_i \mid i \in [1..n], w_i < b] > 0$ is true in I.

(G) $\mathrm{MIN}[w_1 : l_1, \dots, w_n : l_n] \leq b$ is true in I if and only if
$\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) \leq b$ if and only if
$\{i \in [1..n] \mid w_i \leq b, I \models l_i\} \neq \emptyset$ if and only if
$\mathrm{SUM}[1 : l_i \mid i \in [1..n], w_i \leq b] > 0$ is true in I.

(H) $\mathrm{MIN}[w_1 : l_1, \dots, w_n : l_n] \geq b$ is true in I if and only if
$\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) \geq b$ if and only if
$\{i \in [1..n] \mid w_i < b, I \models l_i\} = \emptyset$ if and only if
$\mathrm{SUM}[-1 : l_i \mid i \in [1..n], w_i < b] > -1$ is true in I.

(I) $\mathrm{MIN}[w_1 : l_1, \dots, w_n : l_n] > b$ is true in I if and only if
$\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) > b$ if and only if
$\{i \in [1..n] \mid w_i \leq b, I \models l_i\} = \emptyset$ if and only if
$\mathrm{SUM}[-1 : l_i \mid i \in [1..n], w_i \leq b] > -1$ is true in I.

(J) $\text{MIN}[w_1 : l_1, \dots, w_n : l_n] = b$ is true in I if and only if
$\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) = b$ if and only if
$\{i \in [1..n] \mid w_i = b, I \models l_i\} \neq \emptyset$ and $\{i \in [1..n] \mid w_i < b, I \models l_i\} = \emptyset$ if and only if
$\text{SUM}[1 - n \cdot (b - w_i) : l_i \mid i \in [1..n], w_i \leq b] > 0$ is true in I.

(K) $\text{MIN}[w_1 : l_1, \dots, w_n : l_n] \neq b$ is true in I if and only if
$\min(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\}) \neq b$ if and only if
$\{i \in [1..n] \mid w_i = b, I \models l_i\} = \emptyset$ or $\{i \in [1..n] \mid w_i < b, I \models l_i\} \neq \emptyset$ if and only if
$\text{SUM}[n \cdot (b - w_i) - 1 : l_i \mid i \in [1..n], w_i \leq b] > -1$ is true in I.

(L) $\text{MAX}[w_1 : l_1, \dots, w_n : l_n] \odot b$ is true in I if and only if
$\max(\{w_i \mid i \in [1..n], I \models l_i\} \cup \{-\infty\}) \odot b$ if and only if
$\min(\{-w_i \mid i \in [1..n], I \models l_i\} \cup \{+\infty\})\ f(\odot)\ {-b}$ if and only if
$\text{MIN}[-w_1 : l_1, \dots, -w_n : l_n]\ f(\odot)\ {-b}$ is true in I,
where $< \stackrel{f}{\mapsto} >, \leq \stackrel{f}{\mapsto} \geq, \geq \stackrel{f}{\mapsto} \leq, > \stackrel{f}{\mapsto} <, = \stackrel{f}{\mapsto} =$, and $\neq \stackrel{f}{\mapsto} \neq$.

(M) $\text{COUNT}[l_1, \dots, l_n] \odot b$ is true in I if and only if
$|\{i \in [1..n] \mid I \models l_i\}| \odot b$ if and only if
$\text{SUM}[1 : l_1, \dots, 1 : l_n] \odot b$ is true in I.

(N) $\text{EVEN}[l_1, \dots, l_n]$ is true in I if and only if
$|\{i \in [1..n] \mid I \models l_i\}|$ is an even number if and only if
$|\{i \in [1..n] \mid I \models l_i\}| \neq 2 \cdot i' - 1$ for all $i' \in [1..\lceil n/2 \rceil]$ if and only if
$\bigwedge_{i \in [1..\lceil n/2 \rceil]} \text{SUM}[1 : l_1, \dots, 1 : l_n] \neq 2 \cdot i - 1$ is true in I.

(O) $\text{ODD}[l_1, \dots, l_n]$ is true in I if and only if
$|\{i \in [1..n] \mid I \models l_i\}|$ is an odd number if and only if
$|\{i \in [1..n] \mid I \models l_i\}| \neq 2 \cdot i'$ for all $i' \in [0..\lfloor n/2 \rfloor]$ if and only if
$\bigwedge_{i \in [0..\lfloor n/2 \rfloor]} \text{SUM}[1 : l_1, \dots, 1 : l_n] \neq 2 \cdot i$ is true in I.

□

Proposition 2

Let Π be a program, I be a model of Π, and $J \subset I$ be a model of $F(I, \Pi)$. Then, there is some component C of Π such that $I \cap (C \setminus J) \neq \emptyset$ and $I \setminus (C \setminus J) \models F(I, \Pi)$.

Proof

For any components $C_1 \neq C_2$ of Π, the existence of some path from $\alpha_1 \in C_1$ to $\beta_2 \in C_2$ in $\mathcal{G}_\Pi$ implies that there is no path from any $\alpha_2 \in C_2$ to $\beta_1 \in C_1$ in $\mathcal{G}_\Pi$. Hence, since $J \subset I$ and $\mathcal{G}_\Pi$ is finite, there is some component C of Π such that $I \cap (C \setminus J) \neq \emptyset$ and $\beta \in C \setminus J$ for any path from $\alpha \in C$ to $\beta \in I \setminus J$ in $\mathcal{G}_\Pi$. Consider any rule $r \in F(I, \Pi)$ such that $H(r) \cap (I \setminus (C \setminus J)) = \emptyset$. Then, $I \models B(r)$ and $I \models r$ yield that $H(r) \cap I \neq \emptyset$, so that $H(r) \cap C \neq \emptyset$. On the other hand, since $J \subseteq I \setminus (C \setminus J)$, we have that $H(r) \cap J = \emptyset$, which together with $J \models r$ implies that $J \not\models B(r)$. That is, there is some positive literal $\beta \in B(r)$ such that $I \models \beta$, $J \not\models F(I, \beta)$, some $\alpha \in C$ has an arc to β in $\mathcal{G}_\Pi$, and one of the following three cases applies:

1. If $\beta \in I \setminus J$, then $\beta \in C \setminus J$, so that $I \setminus (C \setminus J) \not\models B(r)$ and $I \setminus (C \setminus J) \models r$.
2. If β is an aggregate of the form (3) such that $\odot$ is $>$, for any $p \in I \setminus J$ such that

$(w : p) \in wLit^+(\beta)$, we have that $p \in C \setminus J$ because some $\alpha \in C$ has a path to p in $\mathcal{G}_\Pi$. Along with $J \subset I$, this in turn yields that

$$\sum_{(w:p)\in wLit^+(\beta), p\in I\setminus(C\setminus J)} w = \sum_{(w:p)\in wLit^+(\beta), p\in J} w.$$

Moreover, $J \subseteq I \setminus (C \setminus J)$ implies that

$$\sum_{(w:p)\in wLit^-(\beta), p\in I\setminus(C\setminus J)} w \leq \sum_{(w:p)\in wLit^-(\beta), p\in J} w,$$

so that

$$\begin{aligned} & \sum_{(w:p)\in wLit^*(\beta), p\in I\setminus(C\setminus J)} w \\ = & \sum_{(w:p)\in wLit^+(\beta), p\in I\setminus(C\setminus J)} w + \sum_{(w:p)\in wLit^-(\beta), p\in I\setminus(C\setminus J)} w \\ \leq & \sum_{(w:p)\in wLit^+(\beta), p\in J} w \qquad + \sum_{(w:p)\in wLit^-(\beta), p\in J} w \\ = & \sum_{(w:p)\in wLit^*(\beta), p\in J} w. \end{aligned}$$

In view of $J \not\models F(I, \beta)$, we further conclude that

$$\begin{aligned} & \sum_{(w:l)\in wLit^*(\beta), I\setminus(C\setminus J)\models F(I,l)} w \\ = & \sum_{(w:p)\in wLit^*(\beta), p\in I\setminus(C\setminus J)} w + \sum_{(w:l)\in wLit^*(\beta), l\notin\mathcal{V}, I\models l} w \\ \leq & \sum_{(w:p)\in wLit^*(\beta), p\in J} w \qquad + \sum_{(w:l)\in wLit^*(\beta), l\notin\mathcal{V}, I\models l} w \\ = & \sum_{(w:l)\in wLit^*(\beta), J\models F(I,l)} w \\ \leq & \, b. \end{aligned}$$

That is, $I \setminus (C \setminus J) \not\models F(I, \beta)$, so that $I \setminus (C \setminus J) \not\models B(r)$ and $I \setminus (C \setminus J) \models r$.

3. If β is an aggregate of the form (3) such that $\odot$ is $\neq$, for any $p \in I \setminus J$ such that $(w : p) \in wLit^*(\beta)$, we have that $p \in C \setminus J$ because some $\alpha \in C$ has a path to p in $\mathcal{G}_\Pi$. Along with $J \subset I$, this in turn yields that

$$\sum_{(w:p)\in wLit^*(\beta), p\in I\setminus(C\setminus J)} w = \sum_{(w:p)\in wLit^*(\beta), p\in J} w.$$

In view of $J \not\models F(I, \beta)$, we further conclude that

$$\begin{aligned} & \sum_{(w:l)\in wLit^*(\beta), I\setminus(C\setminus J)\models F(I,l)} w \\ = & \sum_{(w:p)\in wLit^*(\beta), p\in I\setminus(C\setminus J)} w + \sum_{(w:l)\in wLit^*(\beta), l\notin\mathcal{V}, I\models l} w \\ = & \sum_{(w:p)\in wLit^*(\beta), p\in J} w \qquad + \sum_{(w:l)\in wLit^*(\beta), l\notin\mathcal{V}, I\models l} w \\ = & \sum_{(w:l)\in wLit^*(\beta), J\models F(I,l)} w \\ = & \, b. \end{aligned}$$

That is, $I \setminus (C \setminus J) \not\models F(I, \beta)$, so that $I \setminus (C \setminus J) \not\models B(r)$ and $I \setminus (C \setminus J) \models r$.

Since $I \setminus (C \setminus J) \models r$ also holds for any rule $r \in F(I, \Pi)$ such that $H(r) \cap (I \setminus (C \setminus J)) \neq \emptyset$, we have shown that $I \setminus (C \setminus J) \models F(I, \Pi)$. $\square$

Lemma 1

Let Π be a program, A be an aggregate, and V be a set of propositional atoms such that $rec(\Pi, A) \subseteq V$. Let I be an interpretation such that $I \cap (\{aux\} \cup At^F(A, V)) = \emptyset$ and $J \subseteq I$. Then, the following conditions hold:

1. For any model I' of $pos(A, V)$ such that $I' \setminus (\{aux\} \cup At^F(A, V)) = I$, we have that $ext(I, I) \subseteq I'$ and $ext(I, I) \models F(I', pos(A, V))$.

2. If $J = I$ or $I \setminus J \subseteq C$ for some component C of Π such that there is a rule $r \in \Pi$ with $H(r) \cap C \neq \emptyset$ and $A \in B(r)$, then $ext(J, I) \models F(ext(I, I), pos(A, V))$ and $ext(J, I) \subseteq J'$ for any model J' of $F(ext(I, I), pos(A, V))$ such that $J' \setminus (\{aux\} \cup At^F(A, V)) = J$.

Proof

1. Let $I' \models pos(A, V)$ such that $I' \setminus (\{aux\} \cup At^F(A, V)) = I$. Then, in view of $\{p^F \leftarrow {\sim}p \mid p^F \in At^F(A, V)\} \subseteq pos(A, V)$, we have that $\{p^F \in At^F(A, V) \mid p \notin I\} \subseteq I'$, so that $I' \models \{p \vee p^F \leftarrow {\sim}{\sim}aux \mid p^F \in At^F(A, V)\}$. Moreover, when $aux \in I'$, $\{p^F \leftarrow aux \mid p^F \in At^F(A, V)\} \subseteq pos(A, V)$ yields that $At^F(A, V) \subseteq I'$.
   ($>$) For A of the form (3) such that $\odot$ is $>$, let A' be the body of rule (4) from $pos(A, V)$. Then, we have that $I' \models A'$ if and only if
   $$\begin{aligned} &\textstyle\sum_{(w:l)\in wLit^+(A), I\models l} w - \sum_{(w:l)\in wLit^-(A), l\notin V, I\not\models l} w \\ &- \textstyle\sum_{(w:p)\in wLit^-(A), p\in V, p^F\in I'} w > b - \sum_{(w:l)\in wLit^-(A)} w. \end{aligned} \tag{A1}$$
   By adding $\sum_{(w:l)\in wLit^-(A)} w$ on both sides, (A1) yields
   $$\begin{aligned} &\textstyle\sum_{(w:l)\in wLit^+(A), I\models l} w + \sum_{(w:l)\in wLit^-(A), l\notin V, I\models l} w \\ &+ \textstyle\sum_{(w:p)\in wLit^-(A), p\in V, p^F\notin I'} w > b. \end{aligned} \tag{A2}$$
   Since $\{p \in V \mid (w : p) \in wLit^-(A), p^F \notin I'\} \subseteq I$, $I \models A$ implies that (A2) holds and $\{aux\} \cup At^F(A, V) \subseteq I'$, but (A2) does not hold for $I' = I \cup \{p^F \in At^F(A, V) \mid p \notin I\}$ otherwise. In either case, we have that $ext(I, I) \subseteq I'$ and $ext(I, I) \models F(I', pos(A, V))$, where $ext(I, I) \models A'$ if and only if $I \models A$.
   ($\neq$) For A of the form (3) such that $\odot$ is $\neq$, (A2) holds when $I \models A_>$, where $A_> := \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] > b$. Moreover, for $A_< := \text{SUM}[-w_1 : l_1, \ldots, -w_n : l_n] > -b$, let $A'_<$ be the body of rule (4) from $pos(A_<, V)$. Then, we have that $I' \models A'_<$ if and only if
   $$\begin{aligned} &\textstyle\sum_{(w:l)\in wLit^+(A), l\notin V, I\not\models l} w - \sum_{(w:l)\in wLit^-(A), I\models l} w \\ &+ \textstyle\sum_{(w:p)\in wLit^+(A), p\in V, p^F\in I'} w > \sum_{(w:l)\in wLit^+(A)} w - b. \end{aligned} \tag{A3}$$
   By subtracting $\sum_{(w:l)\in wLit^+(A)} w$ on both sides and multiplying with -1, (A3) yields
   $$\begin{aligned} &\textstyle\sum_{(w:l)\in wLit^+(A), l\notin V, I\models l} w + \sum_{(w:l)\in wLit^-(A), I\models l} w \\ &+ \textstyle\sum_{(w:p)\in wLit^+(A), p\in V, p^F\notin I'} w < b. \end{aligned} \tag{A4}$$
   Since $\{p \in V \mid (w : p) \in wLit^+(A), p^F \notin I'\} \subseteq I$, $I \models A_<$ implies that (A4) holds and $\{aux\} \cup At^F(A, V) \subseteq I'$, but (A4) does not hold for $I' = I \cup \{p^F \in At^F(A, V) \mid p \notin I\}$ otherwise. In view of $I \models A$ if and only if $I \models A_>$ or $I \models A_<$, we further conclude that $ext(I, I) = I \cup \{aux\} \cup At^F(A, V) \subseteq I'$ when $I \models A$, while neither (A2) nor (A4) holds for $I' = I \cup \{p^F \in At^F(A, V) \mid p \notin I\} = ext(I, I)$ when $I \not\models A$. In either case, we have that $ext(I, I) \subseteq I'$ and $ext(I, I) \models F(I', pos(A, V))$.
2. Assume that $J = I$ or $I \setminus J \subseteq C$ for some component C of Π such that there is a rule $r \in \Pi$ with $H(r) \cap C \neq \emptyset$ and $A \in B(r)$, and let $J' \models F(ext(I, I), pos(A, V))$ such that $J' \setminus (\{aux\} \cup At^F(A, V)) = J$. Then, in view of $\{p^F \leftarrow \top \mid p^F \in At^F(A, V), p \notin I\} \subseteq F(ext(I, I), pos(A, V))$, we have that $\{p^F \in At^F(A, V) \mid p \notin I\} \subseteq J'$.

When $I \not\models A$, then $F(ext(I,I), pos(A,V)) = \{p^F \leftarrow \top \mid p^F \in At^F(A,V), p \notin I\}$, $ext(J,I) = J \cup \{p^F \in At^F(A,V) \mid p \notin I\} \subseteq J'$, and $ext(J,I) \models F(ext(I,I), pos(A,V))$. Below assume that $I \models A$, so that $\{p \vee p^F \leftarrow \top \mid p^F \in At^F(A,V)\} \subseteq F(ext(I,I), pos(A,V))$ implies $\{p^F \in At^F(A,V) \mid p \notin J\} \subseteq J'$.

($>$) For A of the form (3) such that $\odot$ is $>$, let A' be the body of rule (4) from $pos(A,V)$. Then, (A2) yields that $J' \models F(ext(I,I), A')$ if and only if

$$\begin{aligned}&\textstyle\sum_{(w:l)\in wLit^+(A),l\notin\mathcal{V},I\models l} w + \sum_{(w:l)\in wLit^-(A),l\notin V,I\models l} w \\ +&\textstyle\sum_{(w:p)\in wLit^+(A),p\in J} w \quad + \sum_{(w:p)\in wLit^-(A),p\in V,p^F\notin J'} w > b.\end{aligned} \tag{A5}$$

Moreover, $\{p \in V \mid (w:p) \in wLit^-(A), p^F \notin J'\} \subseteq J \subseteq I$ implies that

$$\begin{aligned}\textstyle\sum_{(w:p)\in wLit^-(A),p\in V,p^F\notin J'} w &\geq \textstyle\sum_{(w:p)\in wLit^-(A),p\in V\cap J} w \\ &\geq \textstyle\sum_{(w:p)\in wLit^-(A),p\in V\cap I} w.\end{aligned} \tag{A6}$$

In view of the prerequisite that $J = I$ or $I \setminus J \subseteq C$ for some component C of Π such that there is a rule $r \in \Pi$ with $H(r) \cap C \neq \emptyset$ and $A \in B(r)$, if $J \subset I$, some $\alpha \in C$ has an arc to A in $\mathcal{G}_\Pi$. Along with $rec(\Pi, A) \subseteq V$, this yields that $\{p \in I \setminus J \mid (w:p) \in wLit^+(A)\} = \emptyset$ or $\{p \in I \setminus J \mid (w:p) \in wLit^-(A)\} \subseteq V$. In the former case, $I \models A$, $\sum_{(w:p)\in wLit^+(A),p\in J} w = \sum_{(w:p)\in wLit^+(A),p\in I} w$, $\sum_{(w:l)\in wLit^-(A),l\notin V,J\models F(I,l)} w \geq \sum_{(w:l)\in wLit^-(A),l\notin V,I\models l} w$, and (A6) imply that $J \models F(I,A)$ and (A5) hold. Moreover, if $\{p \in I \setminus J \mid (w:p) \in wLit^-(A)\} \subseteq V$, then $\sum_{(w:l)\in wLit^-(A),l\notin V,J\models F(I,l)} w = \sum_{(w:l)\in wLit^-(A),l\notin V,I\models l} w$ and (A6) yield that (A5) holds when $J \models F(I,A)$, but (A5) does not hold for $J' = J \cup \{p^F \in At^F(A,V) \mid p \notin J\}$ otherwise. In view of $ext(I,I) \models A'$ if and only if $I \models A$, we further conclude that $ext(J,I) = J \cup \{aux\} \cup At^F(A,V) \subseteq J'$ when $J \models F(I,A)$, while (A5) does not hold for $J' = J \cup \{p^F \in At^F(A,V) \mid p \notin J\} = ext(J,I)$ when $J \not\models F(I,A)$. In either case, we have that $ext(J,I) \subseteq J'$ and $ext(J,I) \models F(ext(I,I), pos(A,V))$.

($\neq$) For A of the form (3) such that $\odot$ is $\neq$, (A5) holds when $J \models F(I, A_>)$, where $A_> := \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] > b$. Moreover, for $A_< := \text{SUM}[-w_1 : l_1, \ldots, -w_n : l_n] > -b$, let $A'_<$ be the body of rule (4) from $pos(A_<, V)$. Then, (A4) yields that $J' \models F(ext(I,I), A'_<)$ if and only if

$$\begin{aligned}&\textstyle\sum_{(w:l)\in wLit^+(A),l\notin V,I\models l} w \quad + \sum_{(w:l)\in wLit^-(A),l\notin\mathcal{V},I\models l} w \\ +&\textstyle\sum_{(w:p)\in wLit^+(A),p\in V,p^F\notin J'} w + \sum_{(w:p)\in wLit^-(A),p\in J} w < b.\end{aligned} \tag{A7}$$

Dual to (A6) above, $\{p \in V \mid (w:p) \in wLit^+(A), p^F \notin J'\} \subseteq J$ implies that

$$\textstyle\sum_{(w:p)\in wLit^+(A),p\in V,p^F\notin J'} w \leq \sum_{(w:p)\in wLit^+(A),p\in V\cap J} w. \tag{A8}$$

In view of the prerequisite regarding $I \setminus J$ and since $rec(\Pi, A) \subseteq V$, we have that $\{p \in I \setminus J \mid (w:p) \in wLit^*(A)\} \subseteq V$. Hence, $\sum_{(w:l)\in wLit^+(A),l\notin V,J\models F(I,l)} w = \sum_{(w:l)\in wLit^+(A),l\notin V,I\models l} w$ and (A8) yield that (A7) holds when $J \models F(I, A_<)$, but (A7) does not hold for $J' = J \cup \{p^F \in At^F(A,V) \mid p \notin J\}$ otherwise. Moreover, note that $ext(I,I) \not\models A'_<$ implies that

$$
\begin{aligned}
&\textstyle\sum_{(w:l)\in wLit^+(A),J\models F(I,l)} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),J\models F(I,l)} w\\
\geq &\textstyle\sum_{(w:l)\in wLit^+(A),l\notin V,J\models F(I,l)} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),J\models F(I,l)} w\\
= &\textstyle\sum_{(w:l)\in wLit^+(A),l\notin V,I\models l} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),J\models F(I,l)} w\\
\geq &\textstyle\sum_{(w:l)\in wLit^+(A),l\notin V,I\models l} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),I\models l} w\\
\geq &\, b,
\end{aligned}
$$

so that $J \not\models F(I, A_<)$. Similarly, for the body $A'_>$ of rule (4) from $pos(A_>, V)$, since $\sum_{(w:l)\in wLit^-(A),l\notin V,J\models F(I,l)} w = \sum_{(w:l)\in wLit^-(A),l\notin V,I\models l} w$, $ext(I, I) \not\models A'_>$ yields

$$
\begin{aligned}
&\textstyle\sum_{(w:l)\in wLit^+(A),J\models F(I,l)} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),J\models F(I,l)} w\\
\leq &\textstyle\sum_{(w:l)\in wLit^+(A),J\models F(I,l)} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),l\notin V,J\models F(I,l)} w\\
= &\textstyle\sum_{(w:l)\in wLit^+(A),J\models F(I,l)} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),l\notin V,I\models l} w\\
\leq &\textstyle\sum_{(w:l)\in wLit^+(A),I\models l} w &&+ \textstyle\sum_{(w:l)\in wLit^-(A),l\notin V,I\models l} w\\
\leq &\, b,
\end{aligned}
$$

so that $J \not\models F(I, A_>)$. In turn, $J \models F(I, A_>)$ implies $ext(I, I) \models A'_>$, and $ext(I, I) \models A'_<$ follows from $J \models F(I, A_<)$. In view of $J \models F(I, A)$ if and only if $J \models F(I, A_>)$ or $J \models F(I, A_<)$, $J \models F(I, A)$ yields that $ext(I, I) \models A'_>$ and (A5) or $ext(I, I) \models A'_<$ and (A7) hold, so that $ext(J, I) = J \cup \{aux\} \cup At^F(A, V) \subseteq J'$, while neither (A5) nor (A7) holds for $J' = J \cup \{p^F \in At^F(A, V) \mid p \notin J\} = ext(J, I)$ when $J \not\models F(I, A)$. In either case, we have that $ext(J, I) \subseteq J'$ and $ext(J, I) \models F(ext(I, I), pos(A, V))$.
□

Theorem 1
Let Π be a program, A be an aggregate, and V be a set of propositional atoms such that $rec(\Pi, A) \subseteq V$. Then, it holds that $\Pi \equiv_{At(\Pi)} rew(\Pi, A, V)$.

Proof
($\Rightarrow$) Let $I \in SM(\Pi)$. Then, by Lemma 1, $ext(I, I) \models F(ext(I, I), pos(A, V))$ and $ext(I, I) \subseteq I'$ as well as $ext(I, I) \models F(I', pos(A, V))$ for any model I' of $pos(A, V)$ such that $I' \setminus (\{aux\} \cup At^F(A, V)) = I$. Since $I \models \Pi$ and $aux \in ext(I, I)$ if and only if $I \models A$, this yields $ext(I, I) \models rew(\Pi, A, V)$ as well as $ext(I, I) \models F(I', rew(\Pi, A, V))$ for any model I' of $rew(\Pi, A, V)$ such that $I' \setminus (\{aux\} \cup At^F(A, V)) = I$, so that $I' \in SM(rew(\Pi, A, V))$ implies $I' = ext(I, I)$.

Let $J' \subset ext(I, I)$ such that $ext(I, I) \setminus J' \subseteq C'$ for some component C' of $rew(\Pi, A, V)$, and assume that $J' \models (F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))) \cup F(ext(I, I), pos(A, V))$. For $J := J' \setminus (\{aux\} \cup At^F(A, V))$, note that any path from $\alpha \in I \setminus J$ to $\beta \in I \setminus J$ in $\mathcal{G}_{rew(\Pi,A,V)}$ that does not include aux is a path in $\mathcal{G}_\Pi$ as well, while it maps to a path in $\mathcal{G}_\Pi$ (that includes A) otherwise. Hence, $I \setminus J \subseteq C$ for some component C of Π, so that $J' \models F(ext(I, I), pos(A, V))$ yields $J \subset I$ by Lemma 1. Since $I \in SM(\Pi)$, we have that $J \not\models F(I, \Pi)$, while $J' \models F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))$ implies $J \models F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))$ because $At(F(I, \Pi)) \cap (\{aux\} \cup At^F(A, V)) = \emptyset$. That is, $J \not\models F(I, \Pi) \setminus F(ext(I, I), rew(\Pi, A, V))$, so that $I \models B(r)$, $J \models B(F(I, r))$, and $H(r) \cap J = \emptyset$ for some rule $r \in \Pi \setminus rew(\Pi, A, V)$. For such a rule r, we have that $A \in B(r)$, and $I \models r$ yields $H(r) \cap (I \setminus J) \neq \emptyset$. Hence, by Lemma 1,

$ext(J, I) \subseteq J'$, where $A \in B(r)$ together with $I \models B(r)$ and $J \models B(F(I, r))$ imply $aux \in ext(J, I)$. This means that $J' \models (B(F(I, r)) \setminus \{A\}) \cup \{aux\}$, while $H(r) \cap J' = H(r) \cap J = \emptyset$, so that $F(ext(I, I), r') \in F(ext(I, I), rew(\Pi, A, V))$ and $J' \not\models F(ext(I, I), r')$ for the rule $r' \in rew(\Pi, A, V)$ that replaces A in r by aux. We thus conclude that $J' \not\models F(ext(I, I), rew(\Pi, A, V))$ and $\{I' \in SM(rew(\Pi, A, V)) \mid I' \setminus (\{aux\} \cup At^F(A, V)) = I\} = \{ext(I, I)\}$.

($\Leftarrow$) Let $I' \in SM(rew(\Pi, A, V))$ and $I := I' \setminus (\{aux\} \cup At^F(A, V))$. Then, by Lemma 1, we have that $ext(I, I) \subseteq I'$ and $ext(I, I) \models F(I', pos(A, V))$, which yields $ext(I, I) \models F(I', rew(\Pi, A, V))$ and $I' = ext(I, I)$. Moreover, $I \models \Pi$ holds because $At(\Pi) \cap (\{aux\} \cup At^F(A, V)) = \emptyset$ and $aux \in ext(I, I)$ if and only if $I \models A$.

Let $J \subset I$ for some component C of Π, and assume that $J \models F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))$. For $J' := ext(I, I) \setminus (I \setminus J)$, since $ext(I, I) \in SM(rew(\Pi, A, V))$ and $J' \subset ext(I, I)$, we have that $J' \not\models F(ext(I, I), rew(\Pi, A, V))$, while $J \models F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))$ implies $J' \models F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))$ because $At(F(I, \Pi)) \cap (\{aux\} \cup At^F(A, V)) = \emptyset$ and $J' \setminus (\{aux\} \cup At^F(A, V)) = J$. Moreover, $\emptyset \subset H(r) \cap (\{aux\} \cup At^F(A, V)) \subseteq J'$ holds for any $r \in F(ext(I, I), pos(A, V))$, so that $J' \models F(ext(I, I), pos(A, V))$. That is, $J' \not\models F(ext(I, I), rew(\Pi, A, V)) \setminus (F(I, \Pi) \cup F(ext(I, I), pos(A, V)))$, so that $ext(I, I) \models B(r')$, $J' \models B(F(ext(I, I), r'))$, and $H(r') \cap J' = \emptyset$ for some rule $r' \in rew(\Pi, A, V) \setminus (\Pi \cup pos(A, V))$. For such a rule r', we have that $aux \in B(r')$, and $ext(I, I) \models r'$ yields $H(r') \cap (I \setminus J) \neq \emptyset$. Thus, $H(r) \cap (I \setminus J) \neq \emptyset$ and $A \in B(r)$ for the rule $r \in \Pi$ such that r' replaces A in r by aux. Hence, by Lemma 1, $ext(J, I) \models F(ext(I, I), pos(A, V))$, but $ext(J, I) \not\models F(ext(I, I), rew(\Pi, A, V))$ in view of $ext(J, I) \subset ext(I, I)$. Since $ext(J, I) \models F(I, \Pi) \cap F(ext(I, I), rew(\Pi, A, V))$ because $At(F(I, \Pi)) \cap (\{aux\} \cup At^F(A, V)) = \emptyset$ and $ext(J, I) \setminus (\{aux\} \cup At^F(A, V)) = J$, this means that $aux \in ext(J, I)$, $I \models A$, $J \models F(I, A)$, $ext(J, I) \not\models F(ext(I, I), r')$, and $J \not\models F(I, r)$ for rules $r' \in rew(\Pi, A, V) \setminus (\Pi \cup pos(A, V))$ and $r \in \Pi$ as above. We thus conclude that $J \not\models F(I, \Pi)$ and $I \in SM(\Pi)$. $\square$

Proposition 3

Let Π, Π' be programs and A be an aggregate such that $A \notin Ag(\Pi')$. Then, it holds that $rew(\Pi \cup \Pi', A, \mathcal{V}) = rew(\Pi, A, \mathcal{V}) \cup \Pi'$.

Proof

The claim follows immediately by observing that $rew(\Pi, A, \mathcal{V}) \cup \Pi' \subseteq rew(\Pi \cup \Pi', A, \mathcal{V})$ and $rew(\Pi \cup \Pi', A, \mathcal{V}) \setminus rew(\Pi, A, \mathcal{V}) \subseteq \Pi'$. $\square$