From Non-Convex Aggregates to Monotone Aggregates in ASP*

Mario Alviano
University of Calabria, Italy
alviano@mat.unical.it

Wolfgang Faber
University of Huddersfield, UK
wf@wfaber.com

Martin Gebser
University of Potsdam, Germany
gebser@cs.uni-potsdam.de

Abstract

In answer set programming, knowledge involving sets of objects collectively is naturally represented by aggregates, which are rewritten into simpler forms known as monotone aggregates by current implementations. However, there is a complexity gap between general and monotone aggregates. In this paper, this gap is filled by means of a polynomial, faithful, and modular translation function, which can introduce disjunction in rule heads. The translation function is now part of the recent version 4.5 of the grounder GRINGO. This paper focuses on the key points of the translation function, and in particular on the mapping from non-convex sums to monotone sums.

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; 1991]. In a nutshell, stable models are minimal models of a program reduct obtained by deleting rules whose body is false, and by fixing the interpretation of negative literals in the remaining rules. Hence, stable models are those models that contain only necessary atoms under the assumption they provide for default negated formulas.

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 collectively, and are widely used for example to enforce *functional dependencies*, where a rule of the form

$$\bot \leftarrow R'(\mathbf{X}),\ \text{COUNT}[\mathbf{Y} : R(\mathbf{X}, \mathbf{Y}, \mathbf{Z})] > 1$$

constrains relation R to satisfy the functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$, where $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}$ is the set of attributes of R, and R' is the projection of R on $\mathbf{X}$. 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 (where we often leave "F-" implicit in the following). Interestingly, under this restriction F-stable models can be still defined as minimal models of a program reduct obtained by deleting rules whose body is false, and by fixing the interpretation of negative literals in the remaining rules [Alviano *et al.*, 2015], exactly as in the aggregate-free case.

Non-convex aggregations are of particular interest because they may arise in several contexts while modeling complex knowledge [Eiter *et al.*, 2008; 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 a non-convex aggregate of the form $\text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \neq 5$, which was not correctly handled by any ASP system before 2015. In fact, despite the fact that F-stable models were first proposed more than a decade ago, a *complete* implementation of non-convex aggregates was achieved only last year in the paper that this one is based on [Alviano *et al.*, 2015].

Supporting non-convex aggregates was a challenge because typical ASP systems implement a two phases computation: first, the *grounding* phase instantiates the input program; then, the stable models of the ground program are computed by the *solving* phase. The two phases thus handle different input formats: while the grounding phase can

*This work is based on a paper presented at the 31st International Conference on Logic Programming (ICLP 2015) [Alviano *et al.*, 2015]. Mario Alviano was partially supported by the Italian Ministry of University and Research under PON project "Ba2Know (Business Analytics to Know) Service Innovation - LAB", No. PON03PE_00001_1, and by Gruppo Nazionale per il Calcolo Scientifico (GNCS-INdAM).

process any program written by the user, the solving phase expects to read ground programs having a limited structure. In particular, the solving phase expects aggregates to be of a simple form 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]. This observation already evidences that all rewriting techniques introduced before 2015 are applicable 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]. In order to bridge the complexity gap between monotone and non-convex aggregates, introducing disjunction in rule heads is necessary, and it is actually the missing tile of the puzzle, discovered in [Alviano *et al.*, 2015].

The translation is however non-trivial because in general non-convex aggregates have to be replaced by disjunctions of monotone aggregates, so that an auxiliary atom representing the truth value of a non-convex aggregate can be derived whenever some of the monotone aggregates in the disjunction is true. In logic programming, the conventional approach for encoding such a disjunction consists in a set of rules, each having the auxiliary atom in the head and one of the monotone aggregates in the body. However, often this conventional approach can only preserve models, but not stable models. In fact, recall that stable models are minimal models of a program reduct, and the program reduct only contains rules whose body is true. So how to guarantee that all rules resulting from the translation of a non-convex aggregate survive in a program reduct when the aggregate is true? In addition, the program reduct is such that the interpretation of negative literals is fixed. That is, how to guarantee that the interpretation of positive literals in the non-convex aggregate does not become fixed in program reducts? These and related "optimization" issues have been solved in our previous work [Alviano *et al.*, 2015]. The basic ideas and intuitions of the translation function presented at ICLP 2015 are given in this paper. In particular, the paper focuses on SUM, as all other aggregation functions considered in [Alviano *et al.*, 2015] can be translated to sums by using strong equivalences [Lifschitz *et al.*, 2001; Turner, 2003; Ferraris, 2011].

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 the following form:

$$\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \odot b \tag{1}$$

where $n \geq 0$, $b, w_1, \ldots, w_n$ are integers, $l_1, \ldots, l_n$ are propositional literals, $\odot \in \{<, \leq, \geq, >, =, \neq\}$, and $[w_1 : l_1, \ldots, w_n : l_n]$ is a multiset. (Note that this notation of propositional aggregates differs from ASP-Core-2 format[1] for ease of presentation.) A literal is either a propositional literal, or an aggregate. A rule r is of the following form:

$$p_1 \vee \cdots \vee p_m \leftarrow l_1 \wedge \cdots \wedge l_n \tag{2}$$

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 $At(\Pi)$.

Example 1. Consider the following program Π_1:

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

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 $\bot \notin I$, and associates literals with binary digits as follows: for $p \in \mathcal{V}$, $I(p)$ equals 1 if $p \in I$, and 0 otherwise; $I(\sim l) := 1 - I(l)$; $I(\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \odot b)$ equals 1 if $\sum_{i \in [1..n]} w_i I(l_i) \odot b$, and 0 otherwise.

Relation $\models$ is inductively defined as follows: for a literal l, $I \models l$ if $I(l) = 1$; 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; $F(I, \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \odot b) := \text{SUM}[w_1 : F(I, l_1), \ldots, w_n : F(I, l_n)] \odot b$; 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 and Π' are equivalent with respect to a context $V \subseteq \mathcal{V}$, denoted $\Pi \equiv_V \Pi'$, if both $|SM(\Pi)| = |SM(\Pi')|$ and $\{I \cap V \mid I \in SM(\Pi)\} = \{I \cap V \mid I \in SM(\Pi')\}$.

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

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

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

[1] https://www.mat.unical.it/aspcomp2013/ASPStandardization/

$F(\{x_2, unequal, y_1, y_2\}, \Pi_1)$ is

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

and $\{x_2, y_2\}$ is a model of the above program. Similarly, it can be checked that $\{unequal, y_1, y_2\}$ and $\{x_1, x_2, unequal, y_1, y_2\}$ are not stable models of Π_1. ■

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 4. Resorting again to Example 2, note that the aggregate $\text{SUM}[1:x_1, 2:x_2, 2:y_1, 3:y_2] \neq 5$ is non-convex. Replacing $\neq$ with $>$ (resp. $<$) would result into a monotone (resp. convex) aggregate. ■

3 Compilation

Current ASP solvers (as opposed to grounders) only accept a limited set of aggregates, essentially sums of the form (1) such that $b, w_1, \ldots, w_n$ 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 $\Pi \equiv_{At(\Pi)} \Pi'$. Such a function was given in [Alviano *et al.*, 2015], and is briefly recalled in this section. Before, however, it is instructive to consider a few examples that show why other approaches fail to preserve stable models.

Example 5. 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:

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

The two programs differ only subtly: 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 $I \models \text{SUM}[1:x_1, 2:x_2, 2:y_1, 3:y_2] > 5$ or $I \models \text{SUM}[1:x_1, 2:x_2, 2:y_1, 3:y_2] < 5$. However, the two programs are not equivalent. Indeed, it can be checked that Π_2 has no stable model, and in particular $\{x_1, unequal, y_1, y_2\}$ is not stable because $F(\{x_1, unequal, y_1, y_2\}, \Pi_2)$ is

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

and $\{x_1\}$ is one of its models. ■

Example 5 is essentially the origin of the first question posed in the introduction: How to guarantee that all rules resulting from the translation of a non-convex aggregate survive in a program reduct when the aggregate is true? Another obstacle for previous translation functions is represented by negative integers, whose substitution may change the semantics of programs.

Example 6. Let Π_3 be the following program:

$$p \leftarrow \text{SUM}[1:p, -1:q] \geq 0 \qquad p \leftarrow q \qquad q \leftarrow p$$

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 this example is

$$p \leftarrow \text{SUM}[1:p, 1:{\sim}q] \geq 1 \qquad p \leftarrow q \qquad q \leftarrow p$$

which has no stable models. In particular, $\{p, q\}$ is not stable because in the program reduct the aggregate is replaced by $\text{SUM}[1:p, 1:\bot] \geq 1$, and therefore $\emptyset$ is a smaller model. ■

Example 6 is thus the origin of the second question posed in the introduction: How to guarantee that the interpretation of positive literals in the non-convex aggregate does not become fixed in program reducts? Actually, the fact that rewritings such as those hinted in the above examples do not preserve stable models in general can be also explained via complexity arguments. Indeed, 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, unless the polynomial hierarchy collapses to its first level, a polynomial-time compilation must possibly introduce disjunction when rewriting recursive programs. This intuition is used in Section 3.2. Before, in Section 3.1, the structure of input programs is simplified by rewriting sums so that comparison operators are either $\geq$ or $\neq$; after this simplification, negative integers and $\neq$ 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.

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) $\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \leq b \equiv_{SE}$
$\text{SUM}[-w_1 : l_1, \ldots, -w_n : l_n] \geq -b$

(B) $\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] < b \equiv_{SE}$
$\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \leq b - 1$

(C) $\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] > b \equiv_{SE}$
$\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \geq b + 1$

(D) $\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] = b \equiv_{SE}$
$\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \geq b \wedge$
$\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \leq b$

(E) $\text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \odot b \equiv_{SE}$
$\text{SUM}[w : l \mid l \in \{l_1, \ldots, l_n\} \setminus \{\bot\}, w := \sum_{i \in [1..n], l_i = l} w_i, w \neq 0]$

For example, (A) is based on the fact that the following statements are equivalent, for every interpretation I:

1. $I \models \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \leq b$;
2. $\sum_{i \in [1..n]} w_i I(l_i) \leq b$;
3. $\sum_{i \in [1..n]} -w_i I(l_i) \geq -b$;
4. $I \models \text{SUM}[-w_1 : l_1, \ldots, -w_n : l_n] \geq -b$.

Similar observations apply to (B)–(E). The application of (A)–(E), from the last to the first, to a program Π gives an equivalent program Π' with sums over distinct literals associated with integers different from zero, and whose comparison operators are $\geq$ and $\neq$.

3.2 Eliminating non-monotone aggregates

The structure of program Π' can be further simplified by eliminating non-monotone aggregates. (To ease the presentation, in the following we call an aggregate A non-monotone if it contains a negative integer, or if $\odot$ is $\neq$, thus disregarding special cases in which A would still be monotone or convex.) For an aggregate A of the form (1) such that $\odot$ is $\geq$, we define a rule with a fresh propositional atom aux, representing the truth value of A, as head and a monotone aggregate as body:

$$aux \leftarrow \text{SUM}\left(\begin{array}{l}[w_i : l_i \mid i \in [1..n], w_i > 0] \cup \\ [-w_i : l_i^F \mid i \in [1..n], w_i < 0, l_i \in \mathcal{V}] \cup \\ [-w_i : {\sim} l_i \mid i \in [1..n], w_i < 0, l_i \notin \mathcal{V}]\end{array}\right) \geq b - \sum_{i \in [1..n], w_i < 0} w_i \quad (3)$$

Note that any $(w_i : l_i)$ such that $w_i < 0$ and $l_i \notin \mathcal{V}$ is replaced by $(-w_i : {\sim} l_i)$, thus rewarding the falsity of l_i rather than penalizing l_i, which is in turn compensated by adding $-w_i$ to the bound b; such a replacement preserves models even if condition $l_i \notin \mathcal{V}$ is removed [Simons *et al.*, 2002], while this is not the case for stable models [Ferraris and Lifschitz, 2005]. For this reason, for any $l_i \in \mathcal{V}$ associated with a negative weight in A, (3) introduces a fresh, hidden propositional atom l_i^F [Eiter *et al.*, 2005; Janhunen and Niemelä, 2012], and the following rules are also added to the rewritten program:

$$l_i^F \leftarrow {\sim} l_i \quad (4)$$

$$l_i^F \leftarrow aux \quad (5)$$

$$l_i \vee l_i^F \leftarrow {\sim}{\sim} aux \quad (6)$$

Intuitively, l_i^F must be true whenever l_i is false, but also when aux is true, so to implement the concept of *saturation* [Eiter and Gottlob, 1995]. Rules (4) and (5) encode such an intuition. Moreover, rule (6) guarantees that at least one of l_i and l_i^F belongs to any model of reducts obtained from interpretations I containing aux. In fact, l_i^F represents the falsity of l_i in the reduct of rule (3) with respect to I in order to test the satisfaction of the monotone aggregate in (3) relative to subsets of I.

Example 7. Consider Π_3 from Example 6, and the following program Π_3' obtained by applying the rewriting above:

$$\begin{array}{lll} p \leftarrow aux & aux \leftarrow \text{SUM}[1 : p, 1 : q^F] \geq 1 & \\ p \leftarrow q & q \leftarrow p & \\ q^F \leftarrow {\sim} q & q^F \leftarrow aux & q \vee q^F \leftarrow {\sim}{\sim} aux \end{array}$$

Since $SM(\Pi_3')$ consists of the stable model $\{p, q, aux, q^F\}$ only, we have $\Pi_3 \equiv_{\{p,q\}} \Pi_3'$. That is, the two programs are equivalent with respect to the visible atoms $\{p, q\}$. ■

Note that so far the rewriting of non-monotone aggregates provided an answer to the second question posed in the introduction, i.e., the interpretation of positive literals is not fixed in program reducts thanks to the addition of auxiliary atoms that allow for checking all possible assignments. In order to also answer the first question posed in the introduction, the rewriting has to be extended to an aggregate $A := \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \neq b$. In fact, we are going to consider two cases based on splitting A into $A_> := \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] \geq b + 1$ and $A_< := \text{SUM}[-w_1 : l_1, \ldots, -w_n : l_n] \geq -b + 1$. (Note that $A_> \equiv_{SE} \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] > b$, and $A_< \equiv_{SE} \text{SUM}[w_1 : l_1, \ldots, w_n : l_n] < b$.) In a nutshell, all occurrences of A are replaced by a fresh atom aux, which is also the head of two new rules of the form (3), one for $A_>$ and one for $A_<$. Moreover, rules of the form (4)–(6) are introduced for each $i \in [1..n]$ such that $l_i \in \mathcal{V}$, so that all possible assignments can still be checked in program reducts.

Example 8. Consider Π_1 from Example 1, and the following program Π_1' obtained by applying the rewriting above:

$$\begin{array}{lll} x_1 \leftarrow {\sim}{\sim} x_1 & x_2 \leftarrow {\sim}{\sim} x_2 & \\ y_1 \leftarrow unequal & y_2 \leftarrow unequal & \bot \leftarrow {\sim} unequal \\ unequal \leftarrow aux & & \\ aux \leftarrow \text{SUM}[1 : x_1, 2 : x_2, 2 : y_1, 3 : y_2] \geq 6 & & \\ aux \leftarrow \text{SUM}[1 : x_1^F, 2 : x_2^F, 2 : y_1^F, 3 : y_2^F] \geq 4 & & \end{array}$$

where rules of the form (4)–(6) for all $l_i \in \{x_1, x_2, y_1, y_2\}$ are omitted for brevity. Since the only stable model of Π_1' is $\{x_1, unequal, y_1, y_2, aux, x_1^F, x_2^F, y_1^F, y_2^F\}$, we have $\Pi_1 \equiv_{\{x_1, x_2, unequal, y_1, y_2\}} \Pi_1'$. ■

It is important to note that the rewritten $A_>$ and $A_<$ are both true in the stable model above, which is eventually the answer to the first question posed in the introduction. In fact, the rewriting process described in this section can be iterated to eliminate all non-monotone aggregates from a program Π, where $rew(\Pi)$ denotes the resulting LPARSE-like program.

Theorem 1. *For any program Π, we have $\Pi \equiv_{At(\Pi)} rew(\Pi)$.*

4 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 problem solving, 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 any polynomial rewriting requires the introduction of disjunction if recursion is not limited to convex aggregates. Even though the translation presented in this paper is not particularly optimized towards actual recursion, an experiment reported in [Alviano, 2015] shows that it can pave the way to new applications of ASP. In fact, among 46 tested instances of Generalized Subset Sum, 38 were solved by GRINGO+CLASP within a timeout of 900 seconds, while the SMT solver Z3 could only solve 14 of these instances.

References

[Abseher *et al.*, 2014] M. Abseher, B. Bliem, G. Charwat, F. Dusberger, and S. Woltran. Computing secure sets in graphs using answer set programming. In D. Inclezan and M. Maratea, editors, *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP)*, 2014.

[Alviano and Faber, 2013] M. Alviano and W. Faber. The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In P. Cabalar and T. Son, editors, *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 8148 of *LNCS*, pages 67–72. Springer, 2013.

[Alviano *et al.*, 2015] M. Alviano, W. Faber, and M. Gebser. Rewriting recursive aggregates in answer set programming: Back to monotonicity. *Theory and Practice of Logic Programming*, 15(4-5):559–573, 2015.

[Alviano, 2015] M. Alviano. Evaluating answer set programming with non-convex recursive aggregates. In S. Bistarelli, A. Formisano, and M. Maratea, editors, *RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA)*, volume 1451 of *CEUR Workshop Proceedings*, pages 1–15. CEUR-WS.org, 2015.

[Bartholomew *et al.*, 2011] M. Bartholomew, J. Lee, and Y. Meng. First-order semantics of aggregates in answer set programming via modified circumscription. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning (SS-11-06)*, pages 16–22. AAAI, 2011.

[Berman *et al.*, 2002] P. Berman, M. Karpinski, L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *Journal of Computer and System Sciences*, 65(2):332–350, 2002.

[Brewka *et al.*, 2011] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[Eiter and Gottlob, 1995] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[Eiter *et al.*, 2005] T. Eiter, H. Tompits, and S. Woltran. On solution correspondences in answer set programming. In L. Kaelbling and A. Saffiotti, editors, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 97–102. Professional Book Center, 2005.

[Eiter *et al.*, 2008] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.

[Eiter *et al.*, 2012] T. Eiter, M. Fink, T. Krennwallner, and C. Redl. Conflict-driven ASP solving with external sources. *Theory and Practice of Logic Programming*, 12(4-5):659–679, 2012.

[Faber *et al.*, 2008] W. Faber, G. Pfeifer, N. Leone, T. Dell'Armi, and G. Ielpa. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming*, 8(5-6):545–580, 2008.

[Faber *et al.*, 2011] W. Faber, G. Pfeifer, and N. Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

[Ferraris and Lifschitz, 2005] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5(1-2):45–74, 2005.

[Ferraris, 2011] P. Ferraris. Logic programs with propositional connectives and aggregates. *ACM Transactions on Computational Logic*, 12(4):25:1–25:44, 2011.

[Gebser *et al.*, 2012] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *International Conference and Symposium of Logic Programming (ICLP)*, pages 1070–1080. MIT Press, 1988.

[Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3):365–386, 1991.

[Gelfond and Zhang, 2014] M. Gelfond and Y. Zhang. Vicious circle principle and logic programs with aggregates. *Theory and Practice of Logic Programming*, 14(4-5):587–601, 2014.

[Janhunen and Niemelä, 2012] T. Janhunen and I. Niemelä. Applying visible strong equivalence in answer-set program transformations. In E. Erdem, J. Lee, Y. Lierler, and D. Pearce, editors, *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *LNCS*, pages 363–379. Springer, 2012.

[Lifschitz *et al.*, 2001] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[Liu and Truszczyński, 2006] L. Liu and M. Truszczyński. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research*, 27:299–334, 2006.

[Liu *et al.*, 2010] L. Liu, E. Pontelli, T. Son, and M. Truszczyński. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence*, 174(3-4):295–315, 2010.

[Simons *et al.*, 2002] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

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