On Logic Programming with Aggregates

Tran Cao Son, Enrico Pontelli, Islam Elkabani
Computer Science Department
New Mexico State University
Las Cruces, NM 88003, USA
{tson,epontell,ielkaban}@cs.nmsu.edu

Abstract

In this paper, we present a translational semantics for normal logic programs with aggregates. We propose two different translations of logic programs with aggregates into normal logic programs, whose answer set semantics is used to defined the semantics of the original programs. Differently from many of the earlier proposals in this area, our semantics does not impose any syntactic restrictions on the aggregates and the programs. The semantics naturally extends the traditional answer set semantics for normal logic programs, and it subsumes many of the previous proposals in this area, yet it overcomes several drawbacks of those proposals, e.g., by disallowing non-minimal answer sets. We also discuss how the proposed approach can be extended to logic programs with aggregates in the head, a class of programs that has rarely been considered.

The new semantics is natural and intuitive, and it can be directly implemented using available answer set solvers. We describe a system, called $\mathbb{ASP}^A$, that is capable of computing answer sets of programs with arbitrary (possibly recursively defined) aggregates. We also present a preliminary comparison of $\mathbb{ASP}^A$ with another system for computing answer sets of programs with aggregates, $\mathtt{DLV}^{\mathcal{A}}$.

1 Background and Motivation

The handling of aggregates in Logic Programming (LP) has been the subject of intense studies in the late 80s and early 90's [17, 21, 28, 34, 35]. Most of these proposals focused on the theoretical foundations and computational properties of aggregate functions in LP. The recent development of the *answer set programming* paradigm, whose underlying theoretical foundation is the answer set semantics [11], has renewed the interest in the treatment of aggregates in LP and led to a number of new proposals [2, 3, 7, 9, 13, 24, 26]. Unlike many of the earlier proposals, these new efforts provide a sensible semantics for programs with *recursive* aggregates. Most of these new efforts build on the spirit of answer set semantics for LP, and some have found their way in concrete implementations. For example, the current release (built BEN/2/23/05)[1] of $\mathtt{DLV}^{\mathcal{A}}$ handles aggregate-stratified programs [2], and the system described in [7] supports recursive aggregates according to the semantics described in [17]. The ASET-Prolog system supports recursive aggregates, but its implementation [15] is still prototypical.

[1] `http://www.dbai.tuwien.ac.at/proj/dlv`

Answer set semantics [11] for LP has been one of the most widely adopted semantics for general logic programs—i.e., logic programs that allow negation as failure in the body. Answer set semantics is a natural extension of the minimal model semantics for positive logic programs to the case of general logic programs. A set of atoms S is an *answer set* of the program P if S is the minimal model of the positive program P^S (the *reduct of* P *w.r.t.* S), obtained by

(i) removing from P all the rules whose body contains a negation as failure literal $not\ b$ which is false in S (i.e., $b \in S$); and

(ii) removing all the negation as failure literals from the remaining rules.

The transformation P^S is often referred to as the *Gelfond-Lifschitz transformation.*

This definition satisfies several key properties. In particular, answer sets are

(Pr_1) *closed*, i.e., if an answer set satisfies the body of a rule r then it also satisfies its head;

(Pr_2) *supported*—i.e., for each member p of an answer set S there exists a rule $r \in P$ such that p is the head of the rule and the body of r is true in S;

(Pr_3) *minimal*—i.e., no proper subset of an answer set satisfies (Pr_1) and (Pr_2) (w.r.t. P^S).

It should be emphasized that a set of atoms satisfying the three properties (Pr_1)-(Pr_3) might not be an answer set of P. For example, the set $\{p\}$ is not an answer set of the program $\{p \leftarrow p, \quad q \leftarrow not\ p\}$, even though it satisfies the three properties. Nevertheless, these properties constitute the main principles that guided several extensions of the answer set semantics to different classes of logic programs, such as extended and disjunctive logic programs [12], programs with weight constraint rules [22], and programs with aggregates (e.g., [2, 17]).

As evident from the literature, a straightforward extension of the Gelfond-Lifschitz transformation to programs with aggregates leads to the loss of some of the properties (Pr_1)-(Pr_3) (e.g., presence of non-minimal answer sets [17]). Sufficient conditions that characterize classes of programs with aggregates for which the properties (Pr_1)-(Pr_3) of answer sets are guaranteed have been investigated, such as aggregate-stratification and monotonicity (e.g., [21]). Alternatively, researchers have either accepted the loss of some of the (Pr_1)-(Pr_3) properties (e.g., acceptance of non-minimal models [7, 13, 17]) or have *explicitly* introduced minimality or analogous properties as requirements in the definition of answer sets in presence of aggregates (e.g., [9]).

In this paper, we propose a new semantics for logic programming with aggregates, with the following properties:

- It applies to *arbitrary* programs with aggregates (e.g., no syntactic restrictions).
- It is as intuitive as the traditional answer set semantics and it extends traditional answer set semantics (i.e., it has the same behavior on programs without aggregates).

- It *does not* require explicit mention of the properties (Pr_1)-(Pr_3), but the answer sets resulting from the new definition naturally satisfy such properties.
- It can naturally handle aggregates as program facts and as head of program rules.
- It can be implemented by integrating the definition directly in state-of-the-art answer set solvers, such as SMODELS [23], DLV [6], CMODELS [18], ASSAT [19], etc. In particular, it requires only the addition of a module to determine the *"solutions"* of an aggregate,[2] without any modifications to the mechanisms to compute answer sets.

We achieve these objectives by defining a transformation, called *unfolding*, from logic programs with aggregates to normal logic programs. The key idea facilitating this transformation is the generalization of the principle (Pr_2) to the case of aggregates. More precisely, our transformation ensures that, if an aggregate literal is satisfied by a model M, then M supports at least one of its solutions. Solutions of aggregates can be precomputed, and an answer set solver for LP with aggregates can be implemented using standard answer set solvers.

The notion of unfolding has been widely used in various areas of logic programming (e.g., [27, 29, 33]). The inspiration for the approach used in handling aggregates in this paper comes from the methodology proposed in various works on constructive negation (e.g., [1, 4, 32])—in particular, from the idea of unfolding *intensional sets* into sets of solutions, employed to handle intensional sets in [4].

The rest of this paper is organized as follows. Section 2 presents the syntax of our logic programming language with aggregates. Section 3 describes our semantics, investigates some of its properties, and describes an implementation. Section 4 introduces an alternative characterization of the same semantics, which is useful for extending the use of aggregates to the head of program rules. Section 5 compares our approach with the relevant literature. Section 6 discusses some issues related to our approach to semantics of aggregates. Finally, Section 7 presents the conclusions and the future work.

2 A Logic Programming Language with Aggregates

Let us consider a signature $\Sigma_L = \langle \mathcal{F}_L, \mathcal{V}, \Pi_L \cup \Pi_{Agg} \rangle$, where $\mathcal{F}_L$ is a collection of constants, $\mathcal{V}$ is a denumerable collection of variables, Π_L is a collection of predicate symbols, and Π_{Agg} is a collection of unary predicate symbols (*aggregate predicates*). In the rest of this paper, we will assume that $\mathbb{Z}$ is a subset of $\mathcal{F}_L$—i.e., there are distinct constants representing the integer numbers. We will refer to Σ_L as the *ASP signature*. We will also refer to $\Sigma_P = \langle \mathcal{F}_P, \mathcal{V}, \Pi_P \cup \Pi_{Agg} \rangle$ as the *program signature*, where $\mathcal{F}_P \subseteq \mathcal{F}_L$, $\Pi_P \subseteq \Pi_L$, and $\mathcal{F}_P$ is finite. We will denote with $\mathcal{H}_P$ the Σ_P-Herbrand universe, containing the ground terms built using symbols of $\mathcal{F}_P$—and with $\mathcal{B}_P$ the corresponding Σ_P-Herbrand base. We will refer to an atom of the form $p(t_1, \ldots, t_n)$, where $t_i \in \mathcal{F}_P \cup \mathcal{V}$ and $p \in \Pi_P$, as an ASP-atom; an ASP-literal is either an ASP-atom or the negation as failure (*not* A) of an ASP-atom.

[2]This concept is formalized later in the paper.

Definition 1 *An* extensional set *has the form* $\{t_1, \ldots, t_k\}$, *where* t_i *are terms of* Σ_P. *An* extensional multiset *has the form* $\{\{t_1, \ldots, t_k\}\}$ *where* t_i *are (possibly repeated) terms of* Σ_P.

Definition 2 *An* intentional set *is of the form*

$$\{X \mid p(X_1, \ldots, X_k)\}$$

where X *is a variable,* X_i*'s are variables or constants, and* p *is a predicate in* Π_P *of arity* k. *Similarly, an* intensional multiset *is of the form*

$$\{\{X \mid \exists Z_1, \ldots, Z_r.\, p(X_1, \ldots, X_k, Z_1, \ldots, Z_r)\}\}$$

where X *is a variable,* X_i *and* Z_j*'s are variables or constants, and* $X \notin \{Z_1, \ldots, Z_r\}$. *Intuitively, we are collecting the values of* X *that satisfy the atom* $p(X_1, \ldots, X_k, Z_1, \ldots, Z_r)$, *under the assumption that the variables* Z_v *are locally and existentially quantified.*

We call X *and* p *the* grouped variable *and the* predicate *of the set/multiset, respectively.*

Definition 2 can be extended to allow more complex types of sets, e.g., sets with a tuple as the grouped variable, sets with conjunctions of atoms as property of the intensional construction, and intensional sets with existentially quantified variables.

Definition 3 *An* aggregate literal *has the form* $\mathcal{P}(s)$ *where* $\mathcal{P} \in \Pi_{Agg}$ *and* s *is an intensional set or multiset.*

This notation for aggregate literals is a bit more general than the one used in some previous works, and resembles the *abstract constraint atom* notation presented in [20].

In our examples, we will focus on the "standard" aggregate functions and predicates, e.g., Count, Sum, Min, Max, Avg applied to sets/multisets and predicates such as $=$, $\neq$, $\leq$, etc. Also, for the sake of readability, we will often use a more traditional notation when dealing with the standard aggregates; e.g., instead of writing $\text{Sum}^{\leq}_{7}(\{X \mid p(X)\})$ we will use the more common format $\text{Sum}(\{X \mid p(X)\}) \leq 7$.

An $\mathbb{ASP}^A$ rule is an expression of the form

$$A \leftarrow C_1, \ldots, C_m, A_1, \ldots, A_n, not\ B_1, \ldots, not\ B_k \tag{1}$$

where $A, A_1, \ldots, A_n, B_1, \ldots, B_k$ are ASP-atoms, and $C_1, \ldots, C_m$ are aggregate literals ($m \geq 0$, $n \geq 0$, $k \geq 0$). For simplicity, we will assume that the grouped variables in $C_1, \ldots, C_m$ are pairwise distinct and do not occur in $A, A_1, \ldots, A_n, B_1, \ldots, B_k$. Facts—i.e., rules with an empty body—will be represented by writing only the head of the rule (and omitting the "$\leftarrow$" part). An $\mathbb{ASP}^A$ program is a collection of $\mathbb{ASP}^A$ rules.

3 Aggregate Solutions and Unfolding Semantics

In this section we develop a semantics, based on answer sets, for the logic language with aggregates, study some of its properties, and investigate an implementation based on the Smodels system.

3.1 Semantics

We will use the notation $vars(\alpha)$ to denote the set of variables present in an arbitrary term or atom α. We will also write $\bar{X}$ to denote $X_1, \ldots, X_n$. Let t be an aggregate term $\{X \mid p(\bar{Y})\}$. The set of free variables of t, denoted by $fvars(t)$, is defined as $fvars(t) = vars(\bar{Y}) \setminus \{X\}$—i.e., the set of free variables of t contains all the free variables in $p(\bar{Y})$ which are different from X. Given a multiset t of the form, $\{\{X \mid \exists \bar{Z}.p(\bar{Y}, \bar{Z})\}\}$, the set $fvars(t)$ is defined as $fvars(t) = vars(\bar{Y}) \setminus (\{X\} \cup vars(\bar{Z}))$. The set of free variables in an aggregate literal $\mathcal{P}(s)$ is $fvars(s)$. For an ASP-atom p (ASP-literal *not* p), the set of free variables of p (*not* p) is defined as $fvars(p) = vars(p)$ ($fvars(\textit{not}\ p) = vars(p)$). A literal l is *ground* if $fvars(l) = \emptyset$.

Given an $\mathbb{ASP}^A$ rule r of the form (1), the set of free variables in r, denoted by $fvars(r)$, is the collection of all free variables occurring in any of the literals occurring in r. An $\mathbb{ASP}^A$ rule r is *ground* if $fvars(r) = \emptyset$. A ground substitution for r is given by $\{X/c \mid X \in fvars(r),\ c \in \mathcal{H}_P\}$.

For later use, we introduce some notations. For a ground $\mathbb{ASP}^A$ rule r of the form (1), with $head(r)$, $agg(r)$, $pos(r)$, and $neg(r)$ we denote A, $\{C_1, \ldots, C_m\}$, $\{A_1, \ldots, A_n\}$, and $\{B_1, \ldots, B_k\}$ respectively. Furthermore, $body(r)$ denotes the right hand side of the rule r. For a program P, $lit(P)$ denotes the set of all ASP-atoms present in P.

Given an $\mathbb{ASP}^A$ rule r and a ground substitution $\sigma = \{X_1/c_1, \ldots, X_t/c_t\}$ for r, $r\sigma$ is the ground rule obtained from r by simultaneously replacing every occurrence of X_i with c_i ($i = 1, \ldots, t$). $r\sigma$ is called a *ground instantiation* of r. By $ground(r)$ we denote the set of all ground instantiations of the rule r. For a program P, the collection of all ground instantiations of the rules in P, denoted by $ground(P)$, is called the ground instantiation of P, i.e., $ground(P) = \bigcup_{r \in P} ground(r)$.

Definition 4 *An interpretation of an* $\mathbb{ASP}^A$ *program* P *on* Σ_P *is a subset of* $\mathcal{B}_P$*.*

If p is a ground ASP-atom and M is an interpretation, then p is true in M ($M \models p$) if $p \in M$. Given an ASP-atom p, *not* p is true in M ($M \models \textit{not}\, p$) if $p \notin M$.

Let s be the ground intensional set term $\{X \mid p(\bar{X})\}$ and let M be an interpretation; the grounding of s w.r.t. M (denoted by s^M) is the ground extensional set term $\{a_1, \ldots, a_n\}$ where $M \models (p(\bar{X}))\{X/a_i\}$ holds for all and only $1 \leq i \leq n$.

Analogously, let s be the ground intensional multiset $\{\{X \mid \exists \bar{Z}.p(\bar{X}, \bar{Z})\}\}$ and let M be an interpretation, the grounding of s w.r.t. M (denoted by s^M) is the ground extensional multiset $\{\{a_1, \ldots, a_k\}\}$ where for each $1 \leq i \leq k$ there is a ground substitution η_i for $\bar{Z}$ such that $M \models p(\bar{X}, \bar{Z})\eta_i\{X/a_i\}$ and no other element has such property.

Each of the aggregate predicates $\mathcal{P}$ in Π_{Agg} is interpreted as a relation over the set of sets/multisets of constants from $\mathcal{H}_P$. Given $\mathcal{P} \in \Pi_{Agg}$ we denote with $\mathcal{P}^s$ and $\mathcal{P}^m$ the interpretations of $\mathcal{P}$:

$$\mathcal{P}^s \subseteq 2^{\mathcal{H}_P} \qquad\qquad \mathcal{P}^m \subseteq \mathcal{M}(\mathcal{H}_P)$$

($\mathcal{M}(S)$ contains all the finite multisets of elements from S). We will assume that the traditional aggregate functions and predicates are interpreted in the usual way. E.g., an aggregate such as $\textsc{Sum}^{=}_{10}$ is interpreted as the finite sets of numbers whose sum is 10.

Definition 5 (Aggregate Satisfaction) *Given an interpretation M and a ground aggregate literal c of the form $\mathcal{P}(s)$, where s is an intensional set (resp. multiset), we say that c is* satisfied *by M, denoted by $M \models c$, if $s^M \in \mathcal{P}^s$ (resp. $s^M \in \mathcal{P}^m$).*

This allows us to define when an $\mathbb{ASP}^A$rule is satisfied by an interpretation.

Definition 6 (Rule Satisfaction) *M* satisfies the body *of a ground rule r, denoted by $M \models body(r)$, if*

(i) $pos(r) \subseteq M$;
(ii) $neg(r) \cap M = \emptyset$;
(iii) $M \models c$ for every $c \in agg(r)$.

M satisfies a ground rule *r if $head(r) \in M$ or $M \not\models body(r)$.*

Having specified when a rule is satisfied by an interpretation, we can define the notion of model of a program as follows.

Definition 7 *Let P be an $\mathbb{ASP}^A$ program. An interpretation M is a* model *of P if M satisfies every rule in $ground(P)$.*

M is a minimal model *of P if M is a model of P and there is no proper subset of M which is also a model of P.*

We will now present a semantics for $\mathbb{ASP}^A$ programs. The key idea in this semantics is a new reduction of programs with aggregates and negation as failure into logic programs without aggregates. We call this transformation *unfolding*; this transformation is in the same spirit as the transformation semantics for intensional sets proposed in [4, 5].

Definition 8 (Aggregate Solution) *Let c be a ground aggregate literal and p be the predicate occurring in its aggregate term. An* aggregate solution *of c with respect to $\mathcal{B}_P$ is a pair $\langle S_1, S_2\rangle$ of disjoint sets of ground atoms of p[3] such that*

- *$S_1 \subseteq \mathcal{B}_P$ and $S_2 \subseteq \mathcal{B}_P$; and*
- *for every interpretation M, if $S_1 \subseteq M$ and $S_2 \cap M = \emptyset$ then $M \models c$.*

$\mathcal{SOLN}(c)$ denotes the set of all the solutions of the aggregate literal c with respect to $\mathcal{B}_P$.

Let $S = \langle S_1, S_2\rangle$ be an aggregate solution of c; we denote with $S.p$ and $S.n$ the two components S_1 and S_2 of the solution.

Example 1 *Let c be the aggregate literal* $\textsc{Sum}(\{X \mid p(X)\}){\neq}5$ *in a program with the Herbrand base $\{p(1), p(2), p(3)\}$. This literal has a total of 15 solutions of the form $\langle S_1, S_2\rangle$ such that $S_1, S_2 \subseteq \{p(1), p(2), p(3)\}$, $S_1 \cap S_2 = \emptyset$, and (i) either $p(1) \in S_1$; or (ii) $\{p(2), p(3)\} \cap S_2 \neq \emptyset$. These solutions are listed below.*

$\langle\{p(1)\}, \emptyset\rangle$	$\langle\{p(1)\}, \{p(2)\}\rangle$	$\langle\{p(1)\}, \{p(3)\}\rangle$
$\langle\{p(1)\}, \{p(2), p(3)\}\rangle$	$\langle\{p(1), p(2)\}, \emptyset\rangle$	$\langle\{p(1), p(2)\}, \{p(3)\}\rangle$
$\langle\{p(1), p(3)\}, \emptyset\rangle$	$\langle\{p(1), p(3)\}, \{p(2)\}\rangle$	$\langle\{p(2)\}, \{p(3)\}\rangle$
$\langle\{p(2)\}, \{p(3), p(1)\}\rangle$	$\langle\{p(3)\}, \{p(2)\}\rangle$	$\langle\{p(3)\}, \{p(2), p(1)\}\rangle$
$\langle\{p(1), p(2), p(3)\}, \emptyset\rangle$	$\langle\emptyset, \{p(2)\}\rangle$	$\langle\emptyset, \{p(3)\}\rangle$

[3]By *atoms of p* we mean atoms that have p as their predicate symbol.

Let c be the aggregate literal $\mathcal{P}(s)$ and p be the predicate of s. The following holds:

(i) if $M \models c$ then there exists some solution S_c of c such that $S_c.p \subseteq M$ and $S_c.n \cap M = \emptyset$;

(ii) if S_c is a solution of c then, for every set S' of ground atoms of p with $S' \cap (S_c.p \cup S_c.n) = \emptyset$, we have that $\langle S_c.p, S_c.n \cup S' \rangle$ and $\langle S_c.p \cup S', S_c.n \rangle$ are also solutions of c.

We will now define the *unfolding* of an aggregate literal, of a ground rule, and of a program. For simplicity, we use S (resp. *not* S) to denote the conjunction $\bigwedge_{a \in S} a$ (resp. $\bigwedge_{b \in S} not\ b$) when $S \neq \emptyset$; $\emptyset$ (*not* $\emptyset$) stands for $\top$ ($\bot$).[4]

Definition 9 (Unfolding of an Aggregate Literal) *Given a ground aggregate literal c and a solution $S \in \mathcal{SOLN}(c)$, the unfolding of c w.r.t. S, denoted by $c(S)$, is $S.p \wedge not\ S.n$.*

Definition 10 (Unfolding of a Rule) *Let r be a ground rule. A ground rule r' is an* unfolding *of r if there exists a sequence of solutions $\langle S_c \rangle_{c \in agg(r)}$ of the aggregate literals occurring in r such that*

1. *$head(r') = head(r)$,*
2. *$pos(r') = pos(r) \cup \bigcup_{c \in agg(r)} S_c.p$,*
3. *$neg(r') = neg(r) \cup \bigcup_{c \in agg(r)} S_c.n$, and*
4. *$agg(r') = \emptyset$.*

We say that r' is an unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$. *The set of all possible unfoldings of a rule r is denoted by $unfolding(r)$.*

For a $\mathbb{ASP}^A$ program P, $unfolding(P)$ denotes the set of unfolding rules of $ground(P)$. It is easy to see that $unfolding(P)$ is a normal logic program.

Answer sets of $\mathbb{ASP}^A$ programs are defined as follows.

Definition 11 *A set of atoms M is an $\mathbb{ASP}^A$-answer set of P iff M is an answer set of $unfolding(P)$.*

Example 2 *Let P_1 be the program:*[5]

$$
\begin{array}{lcl@{\qquad}lcl}
p(a) & \leftarrow & \textsc{Count}(\{X \mid p(X)\}) > 0. & p(b) & \leftarrow & not\ q. \\
q & \leftarrow & not\ p(b). & & &
\end{array}
$$

The aggregate literal $\textsc{Count}(\{X \mid p(X)\}) > 0$ *has five aggregate solutions with respect to $\mathcal{B}_{P_1} = \{p(a), p(b), q\}$:*

$$\langle \{p(a)\}, \emptyset \rangle \qquad \langle \{p(b)\}, \emptyset \rangle \qquad \langle \{p(a), p(b)\}, \emptyset \rangle \quad \langle \{p(a)\}, \{p(b)\} \rangle \qquad \langle \{p(b)\}, \{p(a)\} \rangle$$

[4]We follow the convention of denoting *true* with $\top$ and *false* with $\bot$.

[5]We would like to thank Vladimir Lifschitz for suggesting this example.

The unfolding of P_1 is the program

$$\begin{array}{llllllll} p(a) & \leftarrow & p(a). & p(a) & \leftarrow & p(b). & & \\ p(a) & \leftarrow & p(a), p(b). & p(a) & \leftarrow & p(a), not\, p(b). & p(a) \leftarrow & p(b), not\, p(a). \\ p(b) & \leftarrow & not\, q. & q & \leftarrow & not\, p(b). & & \end{array}$$

$M_1 = \{q\}$ and $M_2 = \{p(b), p(a)\}$ are the two answer sets of $unfolding(P_1)$, thus $\mathbb{ASP}^A$-answer sets of P_1.

Example 3 *Let P_2 be the program*

$$p(1). \qquad p(2). \qquad p(3). \qquad p(5) \leftarrow q. \qquad q \leftarrow \textsc{Sum}(\{X \mid p(X)\}) > 10.$$

The only aggregate solution of $\textsc{Sum}(\{X \mid p(X)\}) > 10$ with respect to $\mathcal{B}_{P_2} = \{p(1), p(2), p(3), p(5), q\}$: is $\langle\{p(1), p(2), p(3), p(5)\}, \emptyset\rangle$ and $unfolding(P_2)$ contains:

$$p(1). \qquad p(2). \qquad p(3). \qquad p(5) \leftarrow q. \qquad q \leftarrow p(1), p(2), p(3), p(5).$$

which has $M_1 = \{p(1), p(2), p(3)\}$ as its only answer set. Thus, M_1 is the only $\mathbb{ASP}^A$-answer set of P_2.

The next program with aggregates does not have answer sets, even though it does not contain any negation as failure literals.

Example 4 *Consider the program P_3:*

$$p(2). \qquad p(1) \leftarrow \textsc{Min}(\{X \mid p(X)\}) \geq 2.$$

The unique aggregate solution of the aggregate literal $\textsc{Min}(\{X \mid p(X)\}) \geq 2$ with respect to $\mathcal{B}_{P_3} = \{p(1), p(2)\}$ is $\langle\{p(2)\}, \{p(1)\}\rangle$. The unfolding of P_3 consists of the two rules:

$$p(2). \qquad p(1) \leftarrow p(2), not\, p(1).$$

and it does not have any answer sets. As such, P_3 does not have any $\mathbb{ASP}^A$-answer sets.

Observe that, in creating $unfolding(P)$, we use *every* solution of c in $\mathcal{SOLN}(c)$. Since the number of solutions of an aggregate literal can be exponential in the size of the Herbrand base, the size of $unfolding(P)$ can be exponential in the size of P. Fortunately, as we will show later (Theorem 2), this process can be simplified by considering only minimal solutions of c (Definition 13). In practice, for most common uses of aggregates, we have observed a small number of elements in the minimal solution set (typically linear in the extension of the predicate used in the intensional set).

3.2 Properties of $\mathbb{ASP}^A$-Answer Sets

It is easy to see that the notion of $\mathbb{ASP}^A$-answer sets extends the notion of answer sets of normal logic programs. Indeed, if P does not contain aggregate literals, then $unfolding(P) = ground(P)$. Thus, for a program without aggregates P, M is an $\mathbb{ASP}^A$-answer set of P if and only if M is an answer set of P with respect to the Gelfond-Lifschitz definition of answer sets.

We will now show that $\mathbb{ASP}^A$-answer sets satisfies the same properties of minimality, closedness, and supportedness as answer sets for normal logic programs.

Lemma 1 *Every model of $unfolding(P)$ is a model of P.*

Proof. Let M be a model of $unfolding(P)$, and let us consider a rule $r \in ground(P)$ such that M satisfies the body of r. This implies that there exists a sequence of solutions $\langle S_c \rangle_{c \in agg(r)}$ for the aggregates occurring in r, such that $S_c \in \mathcal{SOLN}(c)$, $S_c.p \subseteq M$, and $S_c.n \cap M = \emptyset$. Let r' be the unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$. We have that $pos(r') \subseteq M$ and $neg(r') \cap M = \emptyset$. In other words, M satisfies the body of $r' \in unfolding(P)$. This implies that $head(r') \in M$, i.e., $head(r) \in M$. $\Box$

Lemma 2 *Every model of P is a model of $unfolding(P)$.*

Proof. Let M be a model of P, and let us consider a rule $r' \in unfolding(P)$ such that M satisfies the body of r'. Since $r' \in unfolding(P)$, there exists $r \in ground(P)$ and a sequence of aggregate solutions $\langle S_c \rangle_{c \in agg(r)}$ for the aggregates in r such that M satisfies $S_c.p \wedge \neg S_c.n$ (for $c \in agg(r)$) and r' is the unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$. This means that $pos(r) \subseteq M$, $neg(r) \cap M = \emptyset$, and $M \models c$ for $c \in agg(r)$. In other words, M satisfies $body(r)$. Since M is a model of $ground(P)$, we have that $head(r) \in M$, which means that $head(r') \in M$. $\Box$

Theorem 1 *Let P be a program with aggregates and M be an $\mathbb{ASP}^A$-answer set of P. Then, M is closed, supported, and a minimal model of $ground(P)$.*

Proof. Since M is an $\mathbb{ASP}^A$-answer set of P, Lemma 1 implies that M is a model of $ground(P)$. Minimality of M follows from Lemma 2 and from the fact that M is a minimal model of $unfolding(P)$. Closedness is immediate from Lemma 1.

Supportedness can be derived from the fact that each atom p in M is supported by M (w.r.t. $unfolding(P)$) since M is an answer set of $unfolding(P)$. Thus, if p were not supported by M w.r.t. $ground(P)$, then this would mean that no rule in $unfolding(P)$ supports p, which would contradict the fact that M is an answer set of $unfolding(P)$. $\Box$

Observe that the converse of the above theorem does not hold, as illustrated by the following example.

Example 5 *Let P_4 be the program*

$$\begin{array}{lll}
p(1). & & \\
p(2) & \leftarrow & q. \\
q & \leftarrow & \textsc{Sum}(\{X \mid p(X)\}) \geq 2. \\
q & \leftarrow & \textsc{Sum}(\{X \mid p(X)\}) < 2.
\end{array}$$

It is easy to see that $M = \{p(1), p(2), q\}$ is a minimal model of this ground program—i.e., M is a minimal set of atoms, closed under the rules of $ground(P_4)$ and each atom of M is supported by a rule of $ground(P_4)$. On the other hand, $unfolding(P_4)$ consists of the following rules

$$\begin{array}{llllll}
p(1). & & & & & \\
p(2) & \leftarrow & q. & q & \leftarrow & p(1), p(2). \\
q & \leftarrow & p(2). & q & \leftarrow & p(2), not\, p(1). \\
q & \leftarrow & p(1), not\, p(2). & q & \leftarrow & not\, p(1), not\, p(2). \\
q & \leftarrow & not\, p(2). & & &
\end{array}$$

M is not an answer set of $unfolding(P_4)$. We can easily check that this program does not have an answer set according to Definition 11.

3.3 Implementation

In spite of the number of proposals dealing with aggregates in logic programming, only few implementations have been described. Dell'Armi et al. [2] describe an implementation of aggregates in the DLV engine, based on the semantics described in Section 5.8 (the current distribution is limited to aggregate-stratified programs[6]). Elkabani et al. [7] describe an integration of a Constraint Logic Programming engine (the ECLiPSe engine) and the SMODELS answer set solver; the integration is employed to implement aggregates, with respect to the semantics of Section 5.8. Some more restricted forms of aggregation, characterized according to the semantics of Section 5.8 have also been introduced in the ASET-PROLOG system [13]. Efficient algorithms for bottom-up computation of the perfect model of aggregate-stratified programs have been described in [16, 35].

In this section, we will describe an implementation of a system for computing $\mathbb{ASP}^A$-answer sets by computing the solutions of aggregate literals, unfolding the program, and computing the answer sets using an available answer set solver. We begin with a discussion of computing solutions of aggregates.

3.3.1 Computing the Solutions

As we have mentioned before, the size of the unfolding program $unfolding(P)$ can become unmanageable in some situations. One way to reduce the size of $unfolding(P)$ is to find a set of representative solutions for the aggregate literals occurring in P, whose size is—hopefully—much smaller than the size of the $\mathcal{SOLN}(c)$. Interestingly, in several situations, the number of representative solutions of an aggregate literal is small [30]. We say that a set of solutions is complete if it can be used to check the satisfiability of the aggregate literal in every interpretation of the program. First, we define when a solution *covers* another solution.

Definition 12 *A solution S of an aggregate literal c* covers *a solution T of c, denoted by $T \trianglelefteq_c S$, if, for all interpretations M,*

$$(\ M \models (T.p \wedge not\ T.n)\ ) \ \Rightarrow \ (\ M \models (S.p \wedge not\ S.n)\ )$$

This can be used to define a *complete* and *minimal* sets of solutions of an aggregate literal.

Definition 13 *A set $S(c)$ of solutions of c is* complete *if for every solution S_c of c, there exists $S'_c \in S(c)$ such that $S.c \trianglelefteq_c S'_c$.*

A solution set $S(c)$ is reducible *if there are two distinct solutions S and T in $S(c)$ such that $T \trianglelefteq_c S$. The set of solutions $S(c) \setminus \{T\}$ is then called a* reduction *of $S(c)$. A solution set $S(c)$ is* minimal *if it is complete and not reducible.*

[6]The concept of aggregate stratification is discussed in Subsection 5.3.

By definition, we have that $\mathcal{SOLN}(c)$ is complete. Because of the transitivity of the covering relationship, we can conclude that any minimal solution set of c is a reduction of $\mathcal{SOLN}(c)$. Given a ground program P, let $c_1, \ldots, c_k$ be the aggregate literals present in P, and let us denote with $\zeta(P, [c_1/S(c_1), \ldots, c_k/S(c_k)])$ the unfolding of P where c_i has been unfolded using only the solution set $S(c_i)$.

Theorem 2 *Given a ground program P containing the aggregate literals $c_1, \ldots, c_k$, and given a complete solution set $S(c_i)$ for each literal c_i, we have that M is an $\mathbb{ASP}^A$-answer set of P iff M is an answer set of $\zeta(P, [c_1/S(c_1), \ldots, c_k/S(c_k)])$.*

Proof. For an interpretation M, let $Q_1 = (\zeta(P, [c_1/S(c_1), \ldots, c_k/S(c_k)]))^M$ and $Q_2 = (\zeta(P, [c_1/\mathcal{SOLN}(c_1), \ldots, c_k/\mathcal{SOLN}(c_k)]))^M = (unfolding(P))^M$. We have that M is an $\mathbb{ASP}^A$-answer set of P iff M is an answer set of Q_2. Furthermore, $Q_1 \subseteq Q_2$, and for each rule $r \in Q_2$ there is a rule $r' \in Q_1$ with $head(r) = head(r')$ and $body(r') \subseteq body(r)$. Using this information, we can show that M is an answer set of Q_1 iff M is an answer set of Q_2, which proves the theorem. $\Box$

The above theorem shows that we can use any complete solution set (e.g., a minimal one) to unfold an aggregate literal.

We make use of the following observation to compute a complete solution set:

Theorem 3 *Let c be an aggregate literal and let $\langle S_1, S_2\rangle$, $\langle T_1, T_2\rangle$ be solutions of c. Then $\langle T_1, T_2\rangle \trianglelefteq_c \langle S_1, S_2\rangle$ iff $S_1 \subseteq T_1$ and $S_2 \subseteq T_2$.*

The abstract algorithm in Figure 1 computes a complete solution set $\mathcal{S}(c)$ for a given aggregate literal—when called with `Find_Solution`$(c, \langle\emptyset, \emptyset\rangle)$ and with $\mathcal{S}(c) = \emptyset$ initially. This algorithm is generic—i.e., can be used with arbitrary aggregate predicates, as long as a mechanism to perform the test in line 3 is provided. Observe also that more effective algorithms can be provided for specific classes of aggregates, by using properties of the aggregate predicates used in the aggregate literals [30].

```
1:  Procedure Find_Solution ($c$, $\langle T, F\rangle$)
2:  { assume $T = \{t_1, \ldots, t_k\}$ and $F = \{f_1, \ldots, f_h\}$ }
3:      if $t_1 \wedge \cdots \wedge t_k \wedge \neg f_1 \wedge \cdots \wedge \neg f_h \models c$ then
4:          Add $\langle T, F\rangle$ to $\mathcal{S}(c)$;
5:          return
6:      endif
7:      if $T \cup F = \mathcal{B}_P$ then return;
8:      endif
9:      forall ($p \in \mathcal{B}_P \setminus (T \cup F)$)
10:         Find_Solution($c$, $\langle T \cup \{p\}, F\rangle$);
11:         Find_Solution($c$, $\langle T, F \cup \{p\}\rangle$);
12:     endfor
```

Figure 1: Algorithm to compute solution set of an aggregate

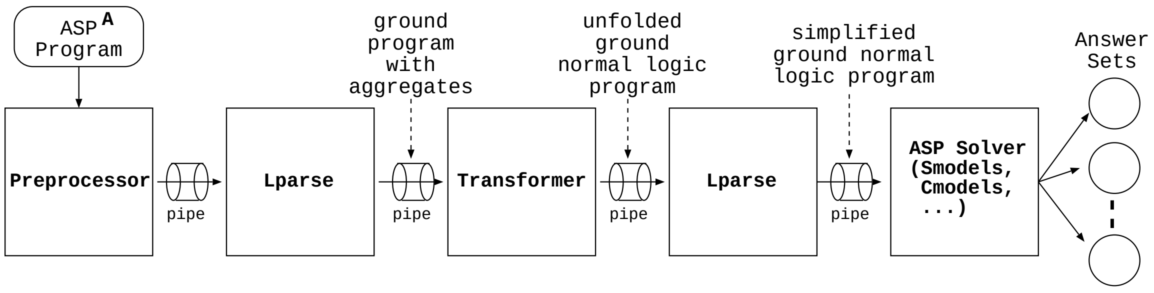
Given a program P containing the aggregate literals $c_1, \ldots, c_k$, we can replace P with $P' = \zeta(P, [c_1/\mathcal{S}(c_1), \ldots c_k/\mathcal{S}(c_k)])$. The program P' is a normal logic program without aggregates, whose answer sets can be computed using a standard answer set solver. The algorithm has been implemented in an extended version of LPARSE—using a combination of ad-hoc rules and an external constraint solver to compute line 3. Note that the **forall** in line 9 is a *non-deterministic* choice of p.

3.3.2 The $\mathbb{ASP}^A$ System

We will now describe the prototype we have constructed, called $\mathbb{ASP}^A$, for computing answer sets of programs with aggregates. The computation is performed following the semantics given in Definition 11, simplified by Theorem 2. In other words, to compute the answer set of a program P, we

1. Compute a complete (and possibly minimal) solution set for each aggregate literal occurring in P;
2. Unfold P using the computed solution sets;
3. Compute the answer sets of the unfolded program $unfolding(P)$ using a standard answer set solver (in our case we tried both SMODELS and CMODELS).

The overall structure of the system is shown in Figure 2.



ASP A Program
Preprocessor
pipe
Lparse
ground program with aggregates
pipe
Transformer
unfolded ground normal logic program
pipe
Lparse
simplified ground normal logic program
pipe
ASP Solver (Smodels, Cmodels, ...)
Answer Sets

Figure 2: Overall System Structure

The computation of answer sets is performed in five steps. In the first step, a preprocessor performs a number of simple syntactic transformations on the input program, which are aimed at rewriting the aggregate literals in a format acceptable by LPARSE. For example, the aggregate literal $\text{SUM}(\{X \mid p(X)\}) \geq 40$ is rewritten to "$agg"$(sum,$ "$x"$, p($"$x"$), 40, geq)$ and an additional rule

$$0\,\{\texttt{``\$agg''(sum,``\$x'', p(``\$x''), 40, geq)}\}\,1$$

is added to the program. The rewritten program is then grounded and simplified using LPARSE, in which aggregate literals are treated like standard (non-aggregate) literals.

The ground program is processed by the *transformer module*, detailed in Figure 3, in which the unfolded program is computed. This module performs the following operations:

1. Creation of the *atom table*, the *aggregate table*, and the *rule table*, used to store the ground atoms, aggregate atoms, and rules of the program, respectively. This is performed by the *Reader* component in Figure 3.

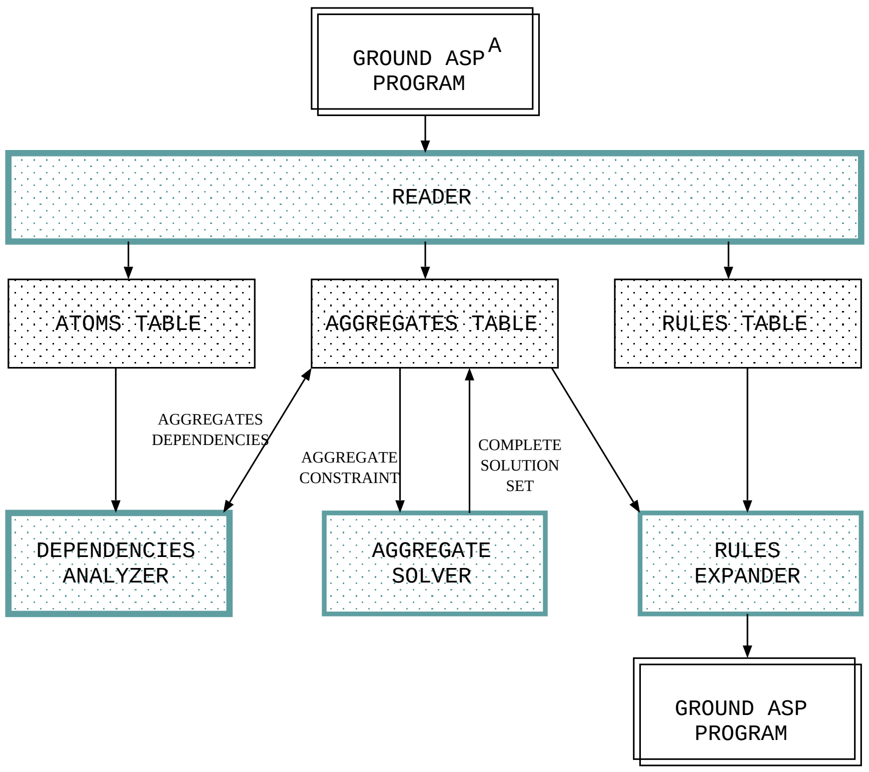


Figure 3: Transformer Module

2. Identification of the dependencies between aggregate atoms and the atoms contributing to such atoms (done by the *Dependencies Analyzer*);
3. Computation of a complete solution set for each aggregate literal (done by the *Aggregate Solver*—as described in the previous subsection);
4. Creation of the unfolded program (done by the *Rule Expander*).

Note that the unfolded program is passed one more time through LPARSE, to avail of the simplifications and optimizations that LPARSE can perform on a normal logic program. The resulting program is a ground normal logic program, whose answer sets can be computed by a system like SMODELS or CMODELS.

3.3.3 Some Experimental Results

We have performed a number of tests using the $\mathbb{ASP}^A$ system. In particular, we selected benchmarks with aggregates presented in the literature. The benchmarks, drawn from various papers on aggregation, are:

- *Company Control:* Let $owns(X, Y, N)$ denotes the fact that company X owns a fraction N of the shares of the company Y. We say that a company X *controls* a company Y if the sum of the shares it owns in Y together with the sum of the shares owned in

Y by companies controlled by X is greater than half of the total shares of Y:

$$\begin{array}{lcl} \mathit{control\_shares}(X,Y,N) & \leftarrow & \mathit{owns}(X,Y,N). \\ \mathit{control\_shares}(X,Y,N) & \leftarrow & \mathit{control}(X,Z), \mathit{owns}(Z,Y,N). \\ \mathit{control}(X,Y) & \leftarrow & \textsc{Sum}(\{\{\, M \mid \mathit{control\_shares}(X,Y,M) \,\}\}) > 50. \end{array}$$

We explored different instances, with varying numbers of companies.

- *Shortest Path:* Suppose a weight-graph is given by relation *arc*, where $\mathit{arc}(X,Y,W)$ means that there is an arc in the graph from node X to node Y of weight W. We represent the shortest path (minimal weight) relation *spath* using the following rules

$$\begin{array}{lcl} \mathit{path}(X,Y,C) & \leftarrow & \mathit{arc}(X,Y,C). \\ \mathit{path}(X,Y,C) & \leftarrow & \mathit{spath}(X,Z,C1), \mathit{arc}(Z,Y,C2), C = C1 + C2. \\ \mathit{spath}(X,Y,C) & \leftarrow & \textsc{Min}(\{\{\, D \mid \mathit{path}(X,Y,D) \,\}\}) = C. \end{array}$$

  The instances explored make use of graphs with varying number of nodes.

- *Party Invitations:* The main idea of this problem is to send out party invitations considering that some people will not accept the invitation unless they know that at least k other people from their friends accept it too.

$$\begin{array}{lcl} \mathit{friend}(X,Y) & \leftarrow & \mathit{friend}(Y,X). \\ \mathit{coming}(X) & \leftarrow & \mathit{requires}(X,0). \\ \mathit{coming}(X) & \leftarrow & \mathit{requires}(X,K), \textsc{Count}(\{\, Y \mid \mathit{kc}(X,Y) \,\}) \geq K. \\ \mathit{kc}(X,Y) & \leftarrow & \mathit{friend}(X,Y), \mathit{coming}(Y). \end{array}$$

  The instances explored in our experiments have different numbers of people invited to the party.

- *Group Seating:* In this problem, we want to arrange the sitting of a group of n people in a restaurant, knowing that the number of tables times the number of seats on each table equals to n. The number of people that can sit at a table cannot exceed the number of chairs at this table, and each person can sit exactly at one table. In addition, people who like each other must sit together at the same table and those who dislike each other must sit at different tables.

$$\begin{array}{lcl} \mathit{at}(P,T) & \leftarrow & \mathit{person}(P), \mathit{table}(T), \mathit{not}\ \mathit{not\_at}(P,T). \\ \mathit{not\_at}(P,T) & \leftarrow & \mathit{person}(P), \mathit{table}(T), \mathit{not}\ \mathit{at}(P,T). \\ & \leftarrow & \mathit{table}(T), \mathit{nchairs}(C), \textsc{Count}(\{\, P \mid \mathit{at}(P,T) \,\}) > C. \\ & \leftarrow & \mathit{person}(P), \textsc{Count}(\{\, T \mid \mathit{at}(P,T) \,\}) \neq 1. \\ & \leftarrow & \mathit{like}(P1,P2), \mathit{at}(P1,T), \mathit{not}\ \mathit{at}(P2,T). \\ & \leftarrow & \mathit{dislike}(P1,P2), \mathit{at}(P1,T), \mathit{at}(P2,T). \end{array}$$

  The benchmark makes use of 16 guests, 4 tables, each having 4 chairs.

- *Employee Raise:* Assume that a manager decides to select a group of employees of size at most N to give them a raise. An employee is a good candidate for the raise if he has worked for at least K hours per week. A relation $emp(X, D, H)$ denotes that an employee X worked H hours during the day D.

$$\begin{array}{lll}
raised(X) & \leftarrow & empName(X), not\, notraised(X). \\
notraised(X) & \leftarrow & empName(X), not\, raised(X). \\
notraised(X) & \leftarrow & empName(X), nHours(K), \textsc{Sum}(\{\{H \mid emp(X, D, H)\}\}) < K. \\
 & \leftarrow & maxRaised(N), \textsc{Count}(\{X \mid raised(X)\}) > N.
\end{array}$$

The different experiments conducted are described by the two parameters M/N, where M is the number of employees and N the maximum number of individuals getting a raise.

- *NM1* and *NM2*: these are two synthetic benchmarks that compute large aggregates that are recursive and non-monotonic.

The code for the benchmarks can be found at: `www.cs.nmsu.edu/~ielkaban/asp-aggr.html`.

Table 1 presents the results obtained. The columns of the table have the following meaning:

- `Program` is the name of the benchmark.
- `Instance` describes the specific instance of the benchmark used in the test.
- `Smodels Time` is the time (in seconds) employed by Smodels to compute the answer sets of the unfolded program.
- `Cmodels Time` is the time (in seconds) employed by Cmodels to compute the answer sets of the unfolded program.
- `Transformer Time` is the time (in seconds) to preprocess and ground the program (i.e., compute the solutions of aggregates and perform the unfolding).
- $\texttt{DLV}^{\mathcal{A}}$ is the time employed by the $\texttt{DLV}^{\mathcal{A}}$ system to execute the same benchmark (where applicable, otherwise marked `N/A`)—observe that the current distribution of this system does not support recursion through aggregates.

All computations have been performed on a Pentium 4, 3.06 GHz machine with 512MB of memory under Linux 2.4.28 using GCC 3.2.1. The system is available for download at `www.cs.nmsu.edu/~ielkaban/asp-aggr.html`.

As we can see from the table, even this relatively simple implementation of aggregates can efficiently solve all benchmarks we tried, offering a coverage significantly larger than other existing implementations. Observed also that the overhead introduced by the computation of aggregate solutions is significant in relatively few cases.

4 An Alternative Semantics

The main advantage of the previously introduced definition of $\mathbb{ASP}^{A}$-answer sets is its simplicity, which allows an easy computation of answer sets of programs with aggregates

Program	Instance	Smodels Time	Cmodels Time	Transformer Time	$\mathtt{DLV}^{\mathcal{A}}$ Time
Company Control	20	0.010	0.00	0.080	N/A
Company Control	40	0.020	0.00	0.340	N/A
Company Control	80	0.030	0.00	2.850	N/A
Company Control	120	0.040	0.030	12.100	N/A
Shortest Path	20	0.220	0.05	0.740	N/A
Shortest Path	30	0.790	0.13	2.640	N/A
Shortest Path	50	3.510	0.51	13.400	N/A
Shortest Path (All Pairs)	20	6.020	1.15	35.400	N/A
Party Invitations	40	0.010	0.00	0.010	N/A
Party Invitations	80	0.020	0.01	0.030	N/A
Party Invitations	160	0.050	0.02	0.050	N/A
Seating	16/4/4	11.40	3.72	0.330	4.337
Employee Raise	15/5	0.57	0.87	0.140	2.750
Employee Raise	21/15	2.88	1.75	1.770	6.235
Employee Raise	25/20	3.42	8.38	5.20	3.95
NM1	125	1.10	0.07	1.00	N/A
NM1	150	1.60	0.18	1.30	N/A
NM2	125	1.44	0.23	0.80	N/A
NM2	150	2.08	0.34	1.28	N/A

Table 1: Computing Answer Sets of Benchmarks with Aggregates

using currently available answer set solvers. Following this approach, all we need to do to compute answer sets of a program P is to compute its unfolded program $unfolding(P)$ and then use an answer set solver to compute the answer sets of $unfolding(P)$. One disadvantage of this method lies in the fact that the size of the program $unfolding(P)$ can be exponential in the size of P—which could potentially become unmanageable. Theoretically, this is not a surprise, as the problem of determining the existence of answer sets for propositional programs with aggregates depends on the types of aggregates present in the program (see also [30] and Chapter 6 in [24]).

In what follows, we present an alternative characterization of the semantics for programs with aggregates, whose underlying principle is still the unfolding mechanism. This alternative characterization allows us to compute the answer sets of a program by using the generate-and-test procedure. The key difference is that the unfolding is now performed with respect to a *given interpretation.*

4.1 Unfolding w.r.t. an Interpretation

Let us start by specializing the notion of solution of an aggregate with respect to a fixed interpretation.

Definition 14 (M-solution) *For a ground aggregate literal c and an interpretation M, its M-solution set is*

$$\mathcal{SOLN}^*(c, M) = \{S_c \mid S_c \in \mathcal{SOLN}(c), S_c.p \subseteq M, S_c.n \cap M = \emptyset\}.$$

Intuitively, $\mathcal{SOLN}^*(c, M)$ is the set of solutions of c which are true in M. For a solution $S_c \in \mathcal{SOLN}^*(c, M)$, the unfolding of c in M w.r.t. S is the conjunction $\bigwedge_{a \in S_c.p} a$. We say that c' is an unfolding of c with respect to M if c' is an unfolding of c in M with respect to some $S_c \in \mathcal{S}OLN^*(c, M)$. When $\mathcal{S}OLN^*(c, M) = \emptyset$, we say that $\perp$ is the unfolding of c in M.

The unfolding of a rule $r \in ground(P)$ with respect to M is the set of rules $unfolding^*(r, M)$ and is defined as follows:

1. If $neg(r) \cap M \neq \emptyset$, or if there is a $c \in agg(r)$ such that $\perp$ is the unfolding of c in M, then $unfolding^*(r, M) = \emptyset$;
2. If $neg(r) \cap M = \emptyset$ and $\perp$ is not the unfolding of c for every $c \in agg(r)$ then $r' \in unfolding^*(r, M)$ if
   (a) $head(r') = head(r)$
   (b) there exists a sequence of aggregate solutions $\langle S_c \rangle_{c \in agg(r)}$ of aggregates in $agg(r)$ such that $S_c \in \mathcal{S}OLN^*(c, M)$ for every $c \in agg(r)$ and $pos(r') = pos(r) \cup \bigcup_{c \in agg(r)} S_c.p$.

Given a program P, we denote with $unfolding^*(P, M)$ the set

$$unfolding^*(P, M) = \bigcup_{r \in ground(P)} unfolding^*(r, M)$$

We define M to be an $\mathbb{ASP}^A$-answer set of P iff M is an answer set of $unfolding^*(P, M)$.

Example 6 *Consider the program P_1 presented earlier and consider the interpretation $M = \{p(a), p(b)\}$. If c is the aggregate* $\text{COUNT}(\{X \mid p(X)\}) > 0$*, then we have that*

$$\mathcal{S}OLN^*(c, M) = \{\langle \{p(a)\}, \emptyset \rangle, \langle \{p(b)\}, \emptyset \rangle, \langle \{p(a), p(b)\}, \emptyset \rangle\}$$

The $unfolding^*(P_1, M)$ is:

$$\begin{array}{lllllllll} p(a) & \leftarrow & p(a). & p(a) & \leftarrow & p(b). & & & \\ p(a) & \leftarrow & p(a), p(b). & p(b) & \leftarrow & not\ q. & q & \leftarrow & not\ p(b). \end{array}$$

Observe that M is indeed an answer set of $unfolding^*(P_1, M)$.

Example 7 *Consider the program P_2 presented earlier and let us consider $M = \{p(1), p(2), p(3), p(5), q\}$. Observe that, if we consider the aggregate c of the form* $\text{SUM}(\{X \mid p(X)\}) > 10$ *then*

$$\mathcal{S}OLN^*(c, M) = \{\langle \{p(1), p(2), p(3), p(5)\}, \emptyset \rangle\}$$

The $unfolding^*(P_2, M)$ is:

$$p(1). \qquad p(2). \qquad p(3). \qquad p(5) \leftarrow q. \qquad q \leftarrow p(1), p(2), p(3), p(5).$$

This program has the unique answer set $\{p(1), p(2), p(3)\}$ which is different from M; thus M is not an answer set of P_2.

The next theorem proves that this new definition is equivalent to the one in Section 3.

Theorem 4 *For any $\mathbb{ASP}^A$ program P, the interpretation M of P is an answer set of $unfolding(P)$ iff M is an answer set of $unfolding^*(P, M)$.*

Proof. Let $R = unfolding^*(P, M)$ and $Q = (unfolding(P))^M$. We have that R and Q are definite programs. We will prove by induction on k that if M is an answer set of Q then $T_Q \uparrow k = T_R \uparrow k$ for every $k \geq 0$. The equation holds trivially for $k = 0$. Let us consider the case for $k > 0$, assuming that $T_Q \uparrow l = T_R \uparrow l$ for $0 \leq l < k$.

- Consider $p \in T_Q \uparrow k$. This means that there exists some rule $r' \in Q$ such that $head(r') = p$ and $body(r') \subseteq T_Q \uparrow k - 1$. From the definition of the Gelfond-Lifschitz reduction and the definition of the unfolded program, we can conclude that there exists some rule $r \in ground(P)$ and a sequence of aggregate solutions $\langle S_c \rangle_{c \in agg(r)}$ for the aggregates in $body(r)$ such that $pos(r') = pos(r) \cup \bigcup_{c \in agg(r)} S_c.p$, and $(neg(r) \cup \bigcup_{c \in agg(r)} S_c.n) \cap M = \emptyset$. In other words, r' is the Gelfond-Lifschitz reduction with respect to M of the unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$. These conditions imply that $r' \in R$. Together with the inductive hypothesis, we can conclude that $p \in T_R \uparrow k$.
- Consider $p \in T_R \uparrow k$. Thus, there exists some rule $r' \in R$ such that $head(r') = p$ and $body(r') \subseteq T_R \uparrow k - 1$. From the definition of R, we can conclude that there exists some rule $r \in ground(P)$ and a sequence of aggregate solutions $\langle S_c \rangle_{c \in agg(r)}$ for the aggregates in $body(r)$ such that $pos(r') = pos(r) \cup \bigcup_{c \in agg(r)} S_c.p$, and $(neg(r) \cup \bigcup_{c \in agg(r)} S_c.n) \cap M = \emptyset$. Thus, $r' \in Q$. Together with the inductive hypothesis, we can conclude that $p \in T_Q \uparrow k$.

This shows that, if M is an answer set of Q, then M is an answer set of R.

Similar arguments can be used to show that if M is an answer set of R, $T_Q \uparrow k = T_R \uparrow k$ for every $k \geq 0$, which means that M is an answer set of Q. □

The above theorem shows that we can compute answer sets of aggregate programs in the same generate–and–test order as in normal logic programs. Given a program P and an interpretation M, instead of computing the Gelfond-Lifschitz's reduct P^M we compute the $unfolding^*(P, M)$. This method of computation might yield better performance but requires modifications of the answer set solver.

Another advantage of this alternative characterization is its suitability to handle aggregates as heads of program rules, as discussed next.

4.2 Aggregates in the Head of Rules

As in most earlier proposals, with the exception of Smodels-weight constraint [22] and logic programs with abstract constraint atoms [20], the language discussed in Section 2 does not allow aggregates as facts (or as head of a rule). To motivate the need for aggregates as rule heads, let us consider the following example. Let us have a set of three students who have taken an exam, and let us assume that at least two got 'A'. This can be encoded as the Smodels program with the set of facts about students and the weight constraint

$$2 \leq \{gotA(X) \mid student(X)\}.$$

If aggregates were allowed in the head, we could encode this problem as the following $\mathbb{ASP}^A$ program

$$\textsc{Count}(\{X \mid gotA(X)\}) \geq 2.$$

along with a constraint stating that if $gotA(X)$ is true then $student(X)$ must be true as well (which can be encoded using the constraint $\perp \leftarrow gotA(X), not\ student(X)$). This program should have four answer sets, each representing a possible grade distribution, in which either one of the students does not receive the 'A' grade or all the three students receive 'A'. This suggests that aggregates in the head of a rule are convenient.

We will next discuss how the semantics in the previous subsection can be extended to allow for aggregate literals in the head of rules. The unfolding is done in two steps. In the first step, we replace a rule with aggregate in the head by a rule without aggregate in the head. The second step is done exactly as in Subsection 4.1.

Let P be a program with aggregates in the head and M be an interpretation of P. Let r be one of the rules where $head(r)$ is an aggregate literal. We define $r^{\perp} = \{\perp \leftarrow body(r)\}$ and $r^{\langle S_1, S_2 \rangle} = \{l \leftarrow body(r) \mid l \in S_1\}$.

For an interpretation M, let P' be a program obtained from P by replacing each rule $r \in P$ whose head is an aggregate literals with either

(a) $r^{\perp}$ if $\mathcal{SOLN}^*(head(r), M) = \emptyset$; or

(b) $r^{\langle S_1, S_2 \rangle}$ for some $\langle S_1, S_2 \rangle \in \mathcal{SOLN}^*(head(r), M)$.

P' is called an *aggregate-free head reduct* of P with respect to M.

A logic program Q is an unfolding of P with respect to M if $Q = unfolding^*(P', M)$ for some aggregate-free head reduct of P with respect to M. We then can say that a set of atoms M is an answer set of P iff M is an answer set of one of the unfoldings of P with respect to M. Observe that, because of aggregates in the head, an $\mathbb{ASP}^A$-answer set might be non minimal.

Example 8 *Consider the* SMODELS *program*

$$\begin{array}{lll} student(a). & student(b). & student(c). \\ 2\ \{gotA(X) : student(X)\}. & & \end{array}$$

which could be represented using aggregates as follows (program P_5):

$$\begin{array}{l} student(a). \quad student(b). \quad student(c). \\ 2 \leq \text{COUNT}(\{X \mid gotA(X)\}). \\ \leftarrow gotA(X), not\ student(X). \end{array}$$

Let us unfold P_5 with respect to two different interpretations

$$M_1 = \{student(a), student(b), student(c), gotA(a)\}$$

$$M_2 = \{student(a), student(b), student(c), gotA(a), gotA(b)\}.$$

Let c denote the aggregate literal $2 \leq \text{COUNT}(\{X \mid gotA(X)\})$.

For M_1, we can check that c is not satisfied by M_1, and hence, the unfolding of the fourth rule of P_5 is the set of rules $\{\perp\}$, i.e., the unfolding of P_5 w.r.t. M_1 is the following program:

$$\begin{array}{l} student(a). \quad student(b). \quad student(c). \\ \perp. \\ \leftarrow gotA(X), not\ student(X). \end{array}$$

This program does not have any answer set. Thus, M_1 is not an $\mathbb{ASP}^A$-answer set of P_5.

For M_2, we have that c is satisfied by M_2 and

$$\mathcal{SOLN}^*(c, M_2) = \{\langle\{gotA(a), gotB(b)\}, \emptyset\rangle, \langle\{gotA(a), gotB(b)\}, \{gotA(c)\}\rangle\}.$$

Since $r^{\langle\{gotA(a),gotB(b)\},\emptyset\rangle} = r^{\langle\{gotA(a),gotB(b)\},\{gotA(c)\}\rangle}$ and contains only two facts $gotA(a)$ and $gotA(b)$, we have that the unfolding of P_5 w.r.t. M_2 is the program

$$\begin{array}{l} student(a). \quad student(b). \quad student(c). \\ gotA(a). \quad gotA(b). \\ \leftarrow gotA(X), not\ student(X). \end{array}$$

which has M_2 as an answer set. Therefore, M_2 is an $\mathbb{ASP}^A$-answer set of P_5.

5 Related Work

In this section, we relate our definition of $\mathbb{ASP}^A$-answer sets to several formulations of aggregates proposed in the literature. We begin with a comparison of the unfolding semantics with the two most recently proposed semantics, the *ultimate stable model semantics* [24–26] and the *minimal answer set semantics* [9]. We then relate our work to earlier proposals, such as perfect models of aggregate-stratified programs (e.g., [21]), fixpoint answer set of monotonic programs [17], and programs with weight constraints [22]. Finally, we briefly discuss the relation of $\mathbb{ASP}^A$-answer sets to other proposals.

5.1 Pelov's Approximation Semantics for Logic Program with Aggregates

The doctoral thesis of Pelov [24] contains a nice generalization of several semantics of logic programs to logic programs with aggregates. The key idea in his work is the use of approximation theory in defining the semantics of logic programs with aggregates. He also developed a translation of logic programs with aggregates to normal logic programs, denoted by tr, which was first given in [25] and then in [24]. It is interesting to note that the translation in [25] and the unfolding proposed in Section 2 have similarities.[7] We will now provide a more detailed comparison between the two translations. For the completeness of the paper, we will review the basics of the translation of [25], expressed using our notation.

Given a logic program with aggregates P, $tr(P)$ denotes the normal logic program obtained after the translation. The translation begins with the translation of each aggregate literal $\ell = \mathcal{P}(s)$ into a disjunction $tr(\ell) = \bigvee F^{At(s)}_{(s_1,s_2)}$ where $At(s)$ is the set of atoms of p—the predicate of s—in $\mathcal{B}_P$, (s_1, s_2) belongs to an index set, $s_1 \subseteq s_2 \subseteq At(s)$, and each $F^{At(s)}_{(s_1,s_2)}$ is a conjunction of the form

$$\bigwedge_{l \in s_1} l \wedge \bigwedge_{l \in s \setminus s_2} not\ l$$

The construction of $tr(\ell)$ considers only pairs of (s_1, s_2) satisfying the following condition: every interpretation I such that $s_1 \subseteq I$ and $s \setminus s_2 \cap I = \emptyset$ also satisfies ℓ. $tr(P)$ is then

[7] We would like to thank a reviewer of an earlier version of this paper who provided us with the pointers to these works. Our translation builds on the previous work of the authors on semantics of logic programming with sets and aggregates [4, 5, 7] and was concurrently and independently developed as [25].

created by rewriting rules with disjunction in the body by a set of rules in a straightforward way. For example, the rule

$$a \leftarrow (b \vee c), d.$$

is replaced by the two rules

$$\begin{array}{l} a \leftarrow b, d. \\ a \leftarrow c, d. \end{array}$$

We can prove a lemma that connects $unfolding(P)$ and $tr(P)$.

Lemma 3 *For every aggregate literal $\ell = \mathcal{P}(s)$, S is a solution of ℓ if and only if $F^{At(s)}_{(S.p, S.p \cup (At(s) \setminus S.n))}$ is a disjunct in $tr(\ell)$.*

Proof. The result is a trivial consequence of the definition of a solution and the definition of $tr(\ell)$. □

This lemma allows us to prove the following relationship between $unfolding(P)$ and $tr(P)$.

Corollary 5.1 *For every program P, A is an $\mathbb{ASP}^A$-answer set of P if and only if A is an exact stable model of P with respect to [26].*

Proof. The result is a trivial consequence of the fact that $unfolding(P) = tr(P)$ and $tr(P)$ has the same set of partial stable models as P [25]. □

5.2 $\mathbb{ASP}^A$-Answer Sets and Minimality Condition

In this subsection, we investigate the relationship between $\mathbb{ASP}^A$-answer sets and the notion of answer set defined by *Faber et al.* in [9]. The notion of answer set proposed in [9] is based on a new notion of reduct, defined as follows. Given a program P and a set of atoms S, the *reduct of P with respect to S*, denoted by ${}^{S}P$, is obtained by removing from $ground(P)$ those rules whose body is not satisfied by S. In other words,

$${}^{S}P = \{r \mid r \in ground(P), S \models body(r)\}.$$

The novelty of this reduct is that it *does not* remove aggregates and negation-as-failure literals satisfied by S.

Definition 15 (FLP-answer set, [9]) *For a program P, S is a* FLP-answer set *of P if it is a minimal model of ${}^{S}P$.*

Observe that the definition of answer set in this approach *explicitly* requires answer sets to be minimal, thus requiring the ability to determine minimal models of a program with aggregates. In the following propositions, we will show that $\mathbb{ASP}^A$-answer sets of a program P are FLP-answer sets and that FLP-answer sets of P are minimal models of $unfolding(P)$, but not necessary $\mathbb{ASP}^A$-answer sets.

Theorem 5 *Let P be a program with aggregates. If M is an $\mathbb{ASP}^A$-answer set, then M is a FLP-answer set of P.*

If M is a FLP-answer set of P then M is a minimal model of $unfolding(P)$.

Proof. Let $Q = unfolding(P)$. Since M is an $\mathbb{ASP}^A$-answer set, we have that M is an answer set of Q Lemma 2 shows that M is a model of $ground(P)$ and hence is a model of $R = {}^M(ground(P))$.

Let us assume that M is not a minimal model of R. This means that there exists $M' \subsetneq M$ such that M' is a model of ${}^M(P)$.

We will show that M' is a model of $Q' = Q^M$ where Q^M is the result of the Gelfond-Lifschitz transformation of the program Q with respect to M.

Consider a rule $r_2 \in Q'$ such that $M' \models body(r_2)$, i.e., $pos(r_2) \subseteq M'$. From the definition of the Gelfond-Lifschitz transformation, we conclude that there exists some $r' \in Q$ such that $pos(r') = pos(r_2)$ and $neg(r') \cap M = \emptyset$. This implies that there is a rule $r \in ground(P)$ and a sequence of solutions $\langle S_c \rangle_{c \in agg(r)}$ of aggregates in r such that r' is the unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$ and for every $c \in agg(r)$, $S_c.p \subseteq M'$ and $S_c.n \cap M = \emptyset$. Since $M' \subseteq M$, we can conclude that $M \models body(r)$, i.e., $r \in R$. Furthermore, $M' \models body(r)$ because $pos(r) \subseteq pos(r') = pos(r_2) \subseteq M'$, $neg(r) \subseteq neg(r')$ and $neg(r') \cap M = \emptyset$, and for every $c \in agg(r)$, $S_c.p \subseteq M'$ and $S_c.n \cap M' = \emptyset$. Since M' is a model of R, we have that $head(r) \in M'$. Since $head(r_2) = head(r') = head(r)$, we have that M' satisfies r_2. This holds for every rule of Q'. Thus, M' is a model of Q. This contradicts the fact that M is an answer set of Q. □

The next example shows that FLP-answer sets might not be $\mathbb{ASP}^A$-answer sets.[8]

Example 9 *Consider the program P_6 where*

$$\begin{array}{lcl} p(1) & \leftarrow & \textsc{Sum}(\{X \mid p(X)\}) \geq 0. \\ p(1) & \leftarrow & p(-1). \\ p(-1) & \leftarrow & p(1). \end{array}$$

The interpretation $M = \{p(1), p(-1)\}$ is a FLP-answer set of P_6. We will show next that P_6 does not have an answer set according to our definition. It is possible to show[9] that the aggregate literal $\textsc{Sum}(\{X \mid p(X)\}) \geq 0$ has the following solutions with respect to $\mathcal{B}_P = \{p(1), p(-1)\}$: $\langle \emptyset, \{p(1), p(-1)\} \rangle$, $\langle \{p(1)\}, \{p(-1)\} \rangle$, $\langle \{p(1)\}, \emptyset \rangle$, and $\langle \{p(1), p(-1)\}, \emptyset \rangle$. The unfolding of P_6, $unfolding(P_6)$, consists of the following rules:

$$\begin{array}{lcl} p(1) & \leftarrow & not\, p(1), not\, p(-1). \\ p(1) & \leftarrow & p(1), not\, p(-1). \\ p(1) & \leftarrow & p(1). \\ p(1) & \leftarrow & p(1), p(-1). \\ p(1) & \leftarrow & p(-1). \\ p(-1) & \leftarrow & p(1). \end{array}$$

It is easy to see that $unfolding(P_6)$ does not have answer sets. Thus, P_6 does not have $\mathbb{ASP}^A$-answer sets. □

[8] We would like to thank an anonymous reviewer of an earlier version of this paper who suggested this example.

[9] We follow the common practice that the sum of an empty set is equal to 0.

Remark 1 *If we replace in P_6 the rule $p(1) \leftarrow \textsc{Sum}(\{X \mid p(X)\}) \geq 0$ with the intuitively equivalent* Smodels *weight constraint rule*

$$p(1) \leftarrow 0[p(1) = 1, p(-1) = -1].$$

we obtain a program that does not have answer sets in Smodels.

The above example shows that our characterization of programs with aggregates differs from the proposal in [9]. Apart from the lack of support for aggregates in the heads of rules, the semantics of [9] might accept answer sets that are not $\mathbb{ASP}^A$-answer sets. Remark 1 shows that our definition is closer to Smodels understanding of aggregates. As shown earlier, our definition allows the use of aggregates in the head of rules, and thus is applicable for logic programs with weight constraints, whereas it is unclear how to modify the definition in [9] to accommodate this class of programs.

5.3 Stratified Programs

Various forms of stratification (e.g., lack of recursion through aggregates) have been proposed to syntactically identify classes of programs that admit a unique minimal model, e.g., local stratification [21], modular stratification [21], and XY-stratification [35]. Efficient evaluation strategies for some of these classes have been investigated (e.g., [14, 16]). Let us show that the simpler notion of aggregate stratification leads to a unique $\mathbb{ASP}^A$ answer set. The program with aggregates P is aggregate-stratified if there is a function $lev : \Pi_P \mapsto \mathbb{N}$ such that, for each rule $H \leftarrow L_1, \ldots, L_k$ in P,

- $lev(pred(H)) \geq lev(pred(L_i))$ if L_i is an ASP-atom;
- $lev(pred(H)) > lev(pred(A_i))$ if L_i is the ASP-literal *not* A_i; and
- $lev(pred(H)) > lev(p)$ if $L_i = \mathcal{P}(s)$ is an aggregate literal with p as the predicate of s.

The notion of *perfect model* is defined as follows.

Definition 16 (Perfect Model, [21]) *The* perfect model *of an aggregate-stratified program P is the minimal model M such that*

- *if M' is another model of P, then the extension of each predicate p of level* 0 *in M is a subset of the extension of p in M'*
- *if M' is another model of P such that M and M' agree on the predicates of all levels up to i, then the extension of each predicate at level $i+1$ in M is a subset of the extension of the same predicate in M'*

From [17, 21] we learn that each aggregate-stratified program has a unique perfect model. We will show next that $\mathbb{ASP}^A$-answer sets for aggregate-stratified programs are perfect models.

Theorem 6 *Let P be an aggregate-stratified program P. The following holds:*

1. *If M is an $\mathbb{ASP}^A$-answer set of P then M is the perfect model of P.*

2. The perfect model of P is an $\mathbb{ASP}^A$-answer set of P.

Proof. The proof is fairly mechanical and it can be found in [31]. □

The following corollary follows directly from the fact that an aggregate-stratified program has a unique perfect model and the above theorem.

Corollary 5.2 *Every aggregate-stratified program admits a unique $\mathbb{ASP}^A$-answer set.*

We believe this equivalence can be easily proved for other forms of aggregate-stratification.

5.4 Monotonic Programs

The notion of *monotonic programs* has been introduced in [21], and later elaborated by other researchers (e.g., [17, 28]), as another class of programs for which the existence of a unique intended model is guaranteed, even in presence of recursion through aggregation. The notion of monotonic programs, defined only for programs with aggregates and without negation, is as follows.

Definition 17 (Monotonic Programs, [17]) *Let F be a collection of base predicates and B be an interpretation of F. A program P is* monotonic *with respect to B if, for each rule r in $ground(P)$ where $pred(head(r)) \notin F$, and for all interpretations I and I', where $B \subseteq I \subseteq I'$, we have that $I \models body(r)$ implies $I' \models body(r)$.*

We will follow the convention used in [28] of fixing the set of base predicates F to be equal to the set of EDB predicates, i.e., it contains only predicates which do not occur in the head of rules of P. This will also mean that B is fixed and B is true in every interpretation of the program P. As such, instead of saying that P is monotonic with respect to B, we will often say that P is monotonic whenever there is no confusion.

For a monotonic program P with respect to the interpretation B of a set of base predicates F, the fixpoint operator, denoted by T_P^B, is extended to include B as follows:

$$T_P^B(I) = \{head(r) \mid r \in ground(P),\ pred(head(r)) \notin F,\ I \cup B \models body(r)\}.$$

It can be shown that T_P^B is monotonic and hence has a unique least fixpoint, denoted by $lfp(T_P^B)$. We will next prove that monotonicity also implies uniqueness of $\mathbb{ASP}^A$-answer sets. First, we prove a simple observation characterizing aggregate solutions in monotonic programs.

Proposition 1 *Let P be a monotonic program with respect to B and r be a rule in $ground(P)$. Assume that $c \in agg(r)$ and S_c is a solution of c. Then, $\langle S_c.p, \emptyset \rangle$ is also a solution of c.*

Proof. Due to the monotonicity of P we have that $M \models c$ for every interpretation M satisfying the condition $S_c.p \subseteq M$. This implies that $\langle S_c.p, \emptyset \rangle$ is a solution of c. □

Theorem 7 *Let P' be a monotonic program w.r.t. B and let $P = P' \cup B$. Then $lfp(T_P^B)$ is an $\mathbb{ASP}^A$-answer set.*

Proof. Let $M = lfp(T_P^B)$, $Q = (unfolding(P))^M$, and $M' = T_Q \uparrow \omega$. We will prove that $M = M'$. First of all, observe that $B \subseteq M \cap M'$, since the elements of B are present as facts in P. Since the predicates used in B do not appear as head of any other rule in P', in the rest we can focus on the elements of M, M' which are distinct from B.

- $M' \subseteq M$: we prove by induction on k that $T_Q \uparrow k \subseteq M$. The result is obvious for $k = 0$. Assume that $T_Q \uparrow k \subseteq M$ and consider $p \in T_Q(T_Q \uparrow k)$. This implies that there is a rule $r' \in Q$ such that $p = \mathit{head}(r')$ and $\mathit{pos}(r') \subseteq T_Q \uparrow k \subseteq M$. This means that there exists a rule $r \in \mathit{ground}(P)$ and a sequence of aggregate solutions $\langle S_c \rangle_{c \in agg(r)}$ such that r' is obtained from r'', which is the unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$, by removing $\mathit{neg}(r'')$ from its body, i.e., $\mathit{neg}(r'') \cap M = \emptyset$. This implies that

  - $\mathit{head}(r) = \mathit{head}(r')$
  - $\mathit{pos}(r') = \mathit{pos}(r'') = \mathit{pos}(r) \cup \bigcup_{c \in agg(r)} S_c.p$ and $\mathit{pos}(r') \subseteq T_Q \uparrow k \subseteq M$
  - $\mathit{neg}(r'') = \bigcup_{c \in agg(r)} S_c.n$ and $\mathit{neg}(r'') \cap M = \emptyset$.

  This implies that $M \models c$ for every $c \in agg(r)$ and $\mathit{pos}(r) \subseteq M$. This allows us to conclude that $M \models \mathit{body}(r)$. By the definition of $T_{P'}^B$, we have that $p = \mathit{head}(r) \in M$.

- $M \subseteq M'$: we will show that $T_P^B \uparrow k \subseteq M'$ for $k \geq 0$. We prove this by induction on k. The result is obvious for $k = 0$. Assume that $T_P^B \uparrow k \subseteq M'$. Consider $p \in T_P^B(T_P^B \uparrow k)$. This implies the existence of a rule $r \in \mathit{ground}(P)$ such that $\mathit{head}(r) = p$ and $T_P^B \uparrow k \models \mathit{body}(r)$. This means that $\mathit{pos}(r) \subseteq T_P^B \uparrow k \subseteq M'$ and $T_P^B \uparrow k \models c$ for every $c \in agg(r)$. From Proposition 1, we know that there exists a sequence of aggregate solutions $\langle S_c \rangle_{c \in agg(r)}$ such that $S_c.n = \emptyset$ and $S_c.p \subseteq T_P^B \uparrow k$. This implies that r', the unfolding of r with respect to $\langle S_c \rangle_{c \in agg(r)}$, is a rule in Q and $\mathit{body}(r') \subseteq M'$. Hence, $p = \mathit{head}(r) = \mathit{head}(r') \in M'$.

The above results allow us to conclude that $M = M'$. □
Since $lfp(T_P^B)$ is unique, we have the following.

Corollary 5.3 *Every monotonic program admits exactly one* $\mathbb{ASP}^A$*-answer set.*

5.5 Logic Programs with Weight Constraints

Let us consider the weight constraints employed by SMODELS and let us describe a translation method to convert them into our language with aggregates. To start with, we will focus on weight constraint that are used in the body of rules (see Sect. 4.2 for aggregates in the heads of rules). For simplicity, we will also focus on weight constraints with non-negative weights (the generalization can be obtained through algebraic manipulations, as described in [22]). A *ground* weight constraint c has the form:[10]

$$L \leq \{p_1{=}w_1, \ldots, p_n{=}w_n, \mathit{not}\, r_1{=}v_1, \ldots, \mathit{not}\, r_m{=}v_m\} \leq U$$

where p_i, r_j are ground atoms, and w_i, v_j, L, U are numeric constants. p_i's and $\mathit{not}\ r_j$'s are called literals of c. $lit(c)$ denotes the set of literals of c. The local weight function

[10]Note that grounding removes SMODELS' conditional literals.

of a constraint c, $w(c)$, returns the weight of its literals. For example, $w(c)(p_i) = w_i$ and $w(c)(not\ r_i) = v_i$. The weight of a weight constraint c in a model S, denoted by $W(c, S)$, is given by

$$W(c, S) = \sum_{p \in lit(c),\, p \in S} w(c)(p) + \sum_{not\ q \in lit(c),\, q \notin S} w(c)(not\, p).$$

We will now show how weight constraints in Smodels can be translated into aggregates in our language. For each weight constraint c, let agg_c^+ and agg_c^- be two new predicates which do not belong to the language of P. Let $r(c)$ be the set of following rules:

$$\begin{array}{lcl} agg_c^+(1, w_1) \leftarrow p_1. & \cdots & agg_c^+(n, w_n) \leftarrow p_n. \\ agg_c^-(1, v_1) \leftarrow r_1. & \cdots & agg_c^-(m, v_m) \leftarrow r_m. \end{array}$$

Intuitively, agg_c^+, agg_c^- assign a specific weight to each literal originally present in the weight constraint. The weight constraint itself is replaced by a conjunction $\tau(c)$:

$$\tau(c) = \begin{array}{c} \text{Sum}(\{\{X \mid \exists Y.agg_c^+(Y, X)\}\}) = S^+ \wedge \\ \text{Sum}(\{\{X \mid \exists Y.agg_c^-(Y, X)\}\}) = S^- \wedge \\ L \leq S^+ + \sum_{i=1}^m v_i - S^- \leq U \end{array}$$

where Sum is an aggregate function with its usual meaning.

Given a Smodels program P, let $\tau(P)$ be the program obtained from P by replacing every weight constraint c in P with $\tau(c)$ and adding the set of rules $r(P)$ to P where $r(P) = \bigcup_{c \text{ is an weight constraint in } P} r(c)$. For each set of atoms S, let us denote with $\hat{S} = S \cup T_{r(P)}(S)$.[11] We have that

$$\hat{S} = \begin{array}{ll} S & \cup\ \{agg_c^+(i, w_i) \mid c \text{ is a weight constraint in } P, p_i = w_i \in c, p_i \in S\} \\ & \cup\ \{agg_c^-(i, v_i) \mid c \text{ is a weight constraint in } P, not\ q_i = v_i \in c, q_i \in S\}. \end{array} \qquad (2)$$

This implies the following lemma.

Lemma 4 *Let S be a set of atoms and c be a weight constraint. For $\hat{S} = S \cup T_{r(P)}(S)$,*

$$W(c, S) = \text{Sum}(\{\{X \mid \exists Y.agg_c^+(Y, X)\}\}^{\hat{S}}) + \sum_{i=1}^m v_i - \text{Sum}(\{\{X \mid \exists Y.agg_c^-(Y, X)\}\}^{\hat{S}})$$

Proof. Follows directly from Equation 2 and the definition of $W(c, S)$. □

Corollary 5.4 *Given a set of atoms S and a weight constraint c, $S \models c$ iff $\hat{S} \models \tau(c)$.*

The next theorem relates P and $\tau(P)$.

Theorem 8 *Let P be a ground* Smodels *program with weight constraints only in the body and with no negative literals in the weight constraints. Let $\tau(P)$ be its translation to aggregates. It holds that*

1. *if S is a* Smodels *answer set of P then $\hat{S}$ is an $\mathbb{ASP}^A$-answer set of $\tau(P)$;*

[11] $T_{r(P)}$ is the immediate consequence operator of program $r(P)$.

2. *if $\hat{S}$ is an $\mathbb{ASP}^A$-answer set of $\tau(P)$ then $\hat{S} \cap lit(P)$ is a* minimal SMODELS *answer set of P.*

Proof. Since negation-as-failure literals can be replaced by weight constraints, without the lost of generality, we can assume that P is a positive program with weight constraints. Let S be a set of atoms and R be the SMODELS reduct of P with respect to S. Furthermore, let $Q = (unfolding(\tau(P)))^{\hat{S}}$. Using Corollary 5.4, we can prove by induction on k that if S is a SMODELS answer set of P (resp. $\hat{S}$ is an answer set of $\tau(P)$) then

1. $T_Q \uparrow k \subseteq (\widehat{T_R \uparrow k})$ for $k \geq 0$
2. $T_R \uparrow k \subseteq (T_Q \uparrow k) \cap lit(P)$ for $k \geq 0$

This proves the two items of the theorem. □

The following example, used in [26] to show that SMODELS-semantics for weight constraints is counter-intuitive in some cases, shows that the equivalence does not hold when negative literals are allowed in the weight constraint.

Example 10 *Let us consider the* SMODELS *program P_6*

$$p(0) \leftarrow \{not\, p(0) = 1\}0.$$

According to the semantics described in [22], we can observe that, for $S = \emptyset$, the reduct P_6^S is $\emptyset$ making it an answer set of P_6. For $S = \{p(0)\}$, the reduct P_6^S is

$$p(0).$$

thus making $\{p(0)\}$ an answer set of P_6.

On the other hand, the intuitively equivalent program using aggregates (we make use of the obvious extension that allows negations in the aggregate) is:

$$p(0) \leftarrow \text{COUNT}(\{X \mid not\, p(X)\}) \leq 0.$$

The unfolding of this program is

$$p(0) \leftarrow p(0).$$

which has the single answer set $\emptyset$.

5.6 Logic Program with Abstract Constraint Atoms

A very general semantic characterization of programs with aggregates has been proposed by Marek et al. in [20]. The framework offers a model where general aggregates can be employed both in the body and in the head of rules. The authors introduce the notion of abstract constraint atom, $C(X)$, where C is a set of sets of atoms (the solutions of the aggregate) and X is a set of atoms. An interpretation M satisfies $C(X)$ if $X \cap M \in C$. In particular, the focus is only on *monotonic constraints*, i.e., constraints $C(X)$ where if $A \in C$ then all supersets of A are also in C. A program is composed of rules of the form

$$B_0(X) \;\text{:-}\; B_1(X_1), \ldots, B_n(X_n), not\; B_{n+1}(Y_1), \ldots, not\; B_{n+m}(Y_m)$$

where each B_i $(i \geq 0)$ is an abstract constraint atom. The semantics of this language is developed as a generalization of answer set semantics [20].

It is possible to show that the semantics of [20] differs from our proposal (and it is, instead, closer to the semantics of [9]). Consider the set of atoms $At = \{p(1), p(-1)\}$ and the abstract aggregate constraints $C = \{\emptyset, \{p(1)\}, \{p(1), p(-1)\}\}$ and $U_{At} = \{X \subseteq At \mid X \neq \emptyset\}$ which are clearly monotonic. We define the program

$$\begin{array}{lll} U_{At}(p(1)) & :- & C(\{p(1), p(-1)\}). \\ U_{At}(p(1)) & :- & U_{At}(p(-1)). \\ U_{At}(p(-1)) & :- & U_{At}(p(1)). \end{array}$$

This program is intuitively equivalent to the program of Example 9. Let us consider $M = \{p(1), p(-1)\}$. The reduct of this program corresponds to the program itself, since there are no negated atoms. Let us show that M is a derivable model according to [20]; let us consider the P-computation:

$$\begin{array}{lllll} X_0 & = & \emptyset & & T_P^{nd}(\emptyset) = \{\{p(1)\}, \{p(1), p(-1)\}\} \\ X_1 & = & \{p(1)\} & & T_P^{nd}(\{p(1)\}) = \{\{p(1), p(-1)\}\} \\ X_2 & = & \{p(1), p(-1)\} & & T_P^{nd}(\{p(1), p(-1)\}) = \{\{p(1), p(-1)\}\} \\ X_3 & = & \{p(1), p(-1)\} & & \end{array}$$

Thus $M = \{p(1), p(-1)\}$ is a derivable model of the program, and thus it is also a stable model of P according to [20].

5.7 Answer Sets for Propositional Theories

The proposal of Ferraris [10] applies a novel notion of reduct and answer sets, developed for propositional theories, to the case of aggregates containing arbitrary formulae. The intuition behind the notion of satisfaction of an aggregate relies on translating the aggregate to a propositional formula that guarantees that all cases where the aggregate is false are ruled out. In particular, for an aggregate of the form $F(\{\alpha_1 = w_1, \ldots, \alpha_k = w_k\}) \odot R$, where α_i are propositional formulae, w_j and R are real numbers, F is a function from multisets of real numbers to $\mathbb{R} \cup \{+\infty, -\infty\}$, and $\odot$ is a relational operator (e.g., $\leq$, $\neq$), the transformation leads to the propositional formula:

$$\bigwedge_{\substack{I \subseteq \{1,\ldots,k\} \\ F(\{w_i \mid i \in I\}) \not\odot R}} \left(\left(\bigwedge_{i \in I} \alpha_i \right) \Rightarrow \left(\bigvee_{i \in \{1,\ldots,k\} \setminus I} \alpha_i \right) \right)$$

The results in [10] show that the new notion of reduct, along with this translation for aggregates, applied to the class of logic programs with aggregates of [9], captures exactly the class of FLP-answer sets.

5.8 Other Proposals

Another semantic characterization of aggregates that has been adopted by several researchers [2, 7, 13, 17] can be simply described as follows. Given a program P and an interpretation M, let $G(M, P)$ be the program obtained by:

(i) removing all the rules with an aggregate or a negation-as-failure literal which is false in M; and

(ii) removing all the remaining aggregates and negation-as-failure literals.

M is a *stable set* of P if M is the least model of $G(M, P)$. We can prove the following result.

Theorem 9 *Let P be a program with aggregates. If M is an $\mathbb{ASP}^A$-answer set of P, then M is a stable set of P.*

Proof: The proof can be found in [31]. $\Box$

The converse is not true in general, since stable sets could be not subset-minimal. For example, the program P_2 in Example 3 has $\{p(1), p(2), p(3), p(5), q\}$ as a stable set.

A somewhat different direction has been explored in [3]. The semantics proposed in [3] relies on approximation theory, and does not in general coincide with answer set semantics on normal programs. The work in [26] addresses this problem and provides a new semantics for aggregate programs which guarantees minimality of total answer sets, as in our case. We discussed the relationship between the work in [26] and ours in Subsection 5.1.

6 Discussions

In this section, we present a program with aggregates in which the unfolding transformation (as well as the translation discussed in [24]) is not applicable. We also briefly discuss the computational complexity issues related to the class of logic programs with aggregates.

6.1 A Limitation of the Unfolding Transformation

The key idea of our approach lies in that, if an aggregate literal is satisfied in an interpretation, one of its solutions must be satisfied. Since our main interest is in the class of programs whose answer sets can be computed by currently available answer set solvers, we are mainly concerned with finite programs and aggregate literals with finite solutions. Here, by a finite solution we mean a solution S whose components $S.p$ and $S.n$ are finite sets of atoms. Certain modifications to our approach might be needed to deal with programs with infinite domains which can give raise to infinite solutions. For example, consider the program P_8 which consists of the rules:

$$\begin{array}{lll} q & \leftarrow & \textsc{Sum}(X \mid p(X)) \geq 2. \\ p(X/2) & \leftarrow & p(X). \\ p(0). & & p(1). \end{array}$$

It is easy to see that the aggregate literal $c = \textsc{Sum}(X \mid p(X)) \geq 2$ has two aggregate solutions, $S = \langle Q, \emptyset \rangle$ and $T = \langle Q \setminus \{p(0)\}, \emptyset \rangle$, where $Q = \{p(1/(2^i)) \mid i = 0, 1, \ldots, \} \cup \{p(0)\}$. Both solutions are infinite. As such, the unfolded version of program P_8 is no longer a normal logic program—in the sense that it contains some rules whose body is not a *finite* set of ASP-literals. Presently, it is not clear how answer sets of such programs should be defined.

In [30], we provide an alternative (equivalent) definition of $\mathbb{ASP}^A$ answer sets which utilizes the notion of solutions but does not employ the unfolding transformation. This semantics yields the intuitive answer for P_8.

6.2 Computational Complexity

Our main goal in this paper is to develop a framework for dealing with aggregates in Answer Set Programming. As we have demonstrated in Section 3.3, the proposed semantics can be easily integrated to existing answer set solvers. In [30], we proved that the complexity of checking the existence of an answer set of a program with aggregates depends on the complexity of evaluating aggregate literals and on the complexity of checking aggregate solutions. In particular, we proved that there are large classes of programs, making use of the standard aggregate functions (e.g., SUM, MIN), for which the answer set checking problem is tractable and the problem of determining the existence of an answer set is in **NP**. These results are in line with similar results presented in [24].

7 Conclusions and Future Works

In this paper, we presented a novel characterization of aggregates in logic programming. Our definition is based on a translation process, which reduces programs with aggregates to normal logic programs. We showed that our approach naturally extends and subsumes many of the existing proposals. We also showed how our approach can be extended to deal with aggregates as heads of rules. We discussed the basic ideas for an implementation based on standard answer set solvers and described $\mathbb{ASP}^A$, a system capable of computing answer sets of program with aggregates.

As we noticed in this work, there are some subtle differences between distinct semantic characterizations recently proposed for logic programming with aggregates; as future work, we propose to investigate formalization of semantics of aggregates that can be parameterized in such a way to cover the most relevant proposals. Our immediate future work is also to investigate whether our alternative characterization for answer sets, based on unfolding w.r.t. a given interpretation, might be used to improve the performance of our implementation. We also plan to extend our implementation to consider the class of logic programs with aggregates in the heads of rules.

Acknowledgments

We would like to thank Vladimir Lifschitz and Michael Gelfond for the numerous discussions, related to the topics of this paper. We also wish to thanks the anonymous referees of a preliminary version of this work, for their insightful comments. The authors have been supported by the NSF grants CNS-0220590, CNS-0454066, and HRD-0420407. The description of the system $\mathbb{ASP}^A$ has been presented in [8].

References

[1] D. Chan. An Extension of Constructive Negation and its Application in Coroutining. In *North American Conference on Logic Programming*, pages 477–493. MIT Press, 1989.

[2] T. Dell'Armi, W. Faber, G. Ielpa, N. Leone, and G. Pfeifer. Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 847–852, 2003.

[3] M. Denecker, N. Pelov, and M. Bruynooghe. Ultimate Well-founded and Stable Semantics for Logic Programs with Aggregates. In *International Conference Logic Programming*, pages 212–226. Springer Verlag, 2001.

[4] A. Dovier, E. Pontelli, and G. Rossi. Constructive Negation and Constraint Logic Programming with Sets. *New Generation Computing*, 19(3):209–256, 2001.

[5] A. Dovier, E. Pontelli, and G. Rossi. Intensional Sets in CLP. In *International Conference on Logic Programming*, pages 284–299. Springer Verlag, 2003.

[6] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System `dlv`: Progress Report, Comparisons, and Benchmarks. In *Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 406–417, Morgan Kaufmann, 1998.

[7] I. Elkabani, E. Pontelli, and T. C. Son. Smodels with CLP and its Applications: a Simple and Effective Approach to Aggregates in ASP. In *International Conference on Logic Programming*, pages 73–89. Springer Verlag, 2004.

[8] I. Elkabani, E. Pontelli, and T. C. Son. Smodels^A – A System for Computing Answer Sets of Logic Programs with Aggregates. In *LPNMR*, pages 427–431. Springer Verlag, 2005.

[9] W. Faber, N. Leone, and G. Pfeifer. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *JELIA*, Springer Verlag, pages 200–212, 2004.

[10] P. Ferraris. Answer Sets for Propositional Theories. In *LPNMR*, Springer Verlag, pp. 119–131, 2005.

[11] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *International Conf. and Symp. on Logic Programming*, MIT Press, pages 1070–1080, 1988.

[12] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–387, 1991.

[13] M. Gelfond. Representing Knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond*, Springer Verlag, pages 413–451, 2002.

[14] S. Greco. Dynamic Programming in Datalog with Aggregates. *IEEE TKDE*, 11(2):265–283, 1999.

[15] M. Heidt. Developing an Inference Engine for ASET-Prolog. Master Thesis, University of Texas at El Paso, 2001.

[16] D. B. Kemp and K. Ramamohanarao. Efficient Recursive Aggregation and Negation in Deductive Databases. *IEEE TKDE*, 10(5):727–745, 1998.

[17] D. B. Kemp and P. J. Stuckey. Semantics of Logic Programs with Aggregates. In *ISLP*, MIT Press, pages 387–401, 1991.

[18] Y. Lierler and M. Maratea. Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In *LPNMR*, Springer Verlag, pages 346–350, 2004.

[19] F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of A Logic Program By SAT Solvers. In *AAAI*, 112–117, 2002.

[20] V.W. Marek and M. Truszczynski. Logic Programs with Abstract Constraint Atoms. In *AAAI*, pp. 86–91, 2004.

[21] I. S. Mumick, H. Pirahesh, and R. Ramakrishnan. The Magic of Duplicates and Aggregates. In *16th International Conference on Very Large Data Bases*, pages 264–277. Morgan Kaufmann, 1990.

[22] I. Niemelä and P. Simons. Extending the Smodels System with Cardinality and Weight Constraints. In *Logic-based Artificial Intelligence*, pages 491–521. Kluwer Academic Publishers, 2000.

[23] I. Niemelä and P. Simons. Smodels - An Implementation of the Stable Model and Well-founded Semantics for Normal Logic Programs. In *LPNMR*, Springer Verlag, pages 420–429, 1997.

[24] N. Pelov. *Semantic of Logic Programs with Aggregates.* PhD thesis, Katholieke Universiteit Leuven, 2004.

[25] N. Pelov, M. Denecker, and M. Bruynooghe. Translation of Aggregate Programs to Normal Logic Programs. In *ASP 2003, Answer Set Programming: Advances in Theory and Implementation), vol 78, CEUR Workshop*, pages 29–42, 2003.

[26] N. Pelov, M. Denecker, and M. Bruynooghe. Partial Stable Models for Logic Programs with Aggregates. In *International Conference on Logic Programming and Non-monotonic Reasoning*, pages 207–219. Springer Verlag, 2004.

[27] A. Pettorossi and M. Proietti. Transformation of Logic Programs. In *Handbook of Logic in Artificial Intelligence*, pages 697–787. Oxford University Press, 1998.

[28] K. A. Ross and Y. Sagiv. Monotonic Aggregation in Deductive Database. *J. Comput. Syst. Sci.*, 54(1):79–97, 1997.

[29] A. Roychoudhury, K. Kumark, C.R. Ramakrishnan, and I.V. Ramakrishnan. An Unfold/Fold Transformation Framework for Definite Logic Programs. *ACM Transactions on Programming Languages and Systems*, 26(3):464–509, 2004.

[30] T. C. Son and E. Pontelli. A Constructive Semantic Characterization of Aggregates in Answer Set Programming. Technical Report NMSU-CS-2005-007, New Mexico State University, 2005. `www.cs.nmsu.edu/~tson/papers/agg-007.pdf`

[31] T. C. Son, E. Pontelli, and I. Elkabani. A Translational Semantics for Aggregates in Logic Programming. Technical Report NMSU-CS-2005-005, New Mexico State University, 2005. `www.cs.nmsu.edu/~tson/papers/agg-005.pdf`

[32] P.J. Stuckey. Negation and Constraint Logic Programming. *Information & Computation*, 118(1):12–33, 1995.

[33] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In *International Conference on Logic Programming*, pages 127–138, 1984.

[34] A. Van Gelder. The well-founded semantics of aggregation. In *PODS*, 127–138. ACM Press, 1992.

[35] C. Zaniolo, N. Arni, and K. Ong. Negation and Aggregates in Recursive Rules: the LDL++ Approach. In *DOOD*, ACM Press, pages 204–221, 1993.