TPLP **23** (1): 157–194, 2023.
doi:10.1017/S1471068422000047 First published online 23 February 2022

*Aggregate Semantics for Propositional Answer Set Programs**

MARIO ALVIANO
University of Calabria, Rende, Italy
(*e-mail:* alviano@mat.unical.it)

WOLFGANG FABER
University of Klagenfurt, Klagenfurt, Austria
(*e-mail:* wolfgang.faber@aau.at)

MARTIN GEBSER
University of Klagenfurt, Klagenfurt, Austria
Graz University of Technology, Graz, Austria
(*e-mail:* martin.gebser@aau.at@aau.at)

submitted 28 July 2022; revised 20 September 2021; accepted 21 January 2022

Abstract

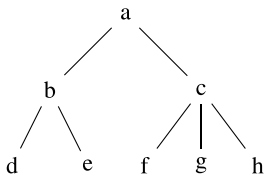
Answer set programming (ASP) emerged in the late 1990s as a paradigm for knowledge representation and reasoning. The attractiveness of ASP builds on an expressive high-level modeling language along with the availability of powerful off-the-shelf solving systems. While the utility of incorporating aggregate expressions in the modeling language has been realized almost simultaneously with the inception of the first ASP solving systems, a general semantics of aggregates and its efficient implementation have been long-standing challenges. Aggregates have been proposed and widely used in database systems, and also in the deductive database language Datalog, which is one of the main precursors of ASP. The use of aggregates was, however, still restricted in Datalog (by either disallowing recursion or only allowing monotone aggregates), while several ways to integrate unrestricted aggregates evolved in the context of ASP. In this survey, we pick up at this point of development by presenting and comparing the main aggregate semantics that have been proposed for propositional ASP programs. We highlight crucial properties such as computational complexity and expressive power, and outline the capabilities and limitations of different approaches by illustrative examples.

KEYWORDS: answer set programming, aggregate expressions, semantics, complexity and expressiveness

1 Introduction

Answer set programming (ASP) (Brewka *et al*. 2011; Gelfond and Leone 2002; Lifschitz 2002; Marek and Truszczyński 1999; Niemelä 1999) is a paradigm for knowledge rep-

*The work of Mario Alviano was partially supported by projects PRIN "Declarative Reasoning over Streams" (CUP: H24I17000080001), PON-MISE MAP4ID "Multipurpose Analytics Platform 4 Industrial Data" (CUP: B21B19000650008), lab LAIA (part of SILA), and GNCS-INdAM. Martin Gebser was partially supported by KWF project 28472, cms electronics GmbH, FunderMax GmbH, Hirsch Armbänder GmbH, incubed IT GmbH, Infineon Technologies Austria AG, Isovolta AG, Kostwein Holding GmbH, and Privatstiftung Kärntner Sparkasse. We are grateful to the reviewers for valuable and constructive comments helping to improve this paper.




```
tree(a).                                  nodes(a,8).
tree(b). tree(c).                         nodes(b,3).
tree(d). tree(e).                         nodes(c,4).
tree(f). tree(g). tree(h).                nodes(d,1).
                                          nodes(e,1).
child(a,b). child(a,c).                   nodes(f,1).
child(b,d). child(b,e).                   nodes(g,1).
child(c,f). child(c,g). child(c,h).       nodes(h,1).
```

Fig. 1. A tree, its relational representation, and the number of nodes in each (sub)tree.

resentation and reasoning. ASP knowledge bases are encoded by means of logic rules interpreted according to the stable model semantics (Gelfond and Lifschitz 1988; 1991), that is, models of ASP programs are required to satisfy an additional stability condition guaranteeing that all true atoms in a model are necessary. A strength of ASP is its high-level modeling language, capable of expressing all problems in the complexity classes NP and Σ_2^P by means of declarative statements (Dantsin *et al*. 2001; Schlipf 1995), depending on the availability of disjunctive rules or unrestricted aggregates. The attractiveness of ASP also builds on the availability of powerful off-the-shelf solving systems, among them CLINGO (Gebser *et al*. 2019), DLV (Alviano *et al*. 2017), and IDP (Bruynooghe *et al*. 2015).

The language of ASP offers several constructs to ease the representation of practical knowledge. Aggregate expressions received particular interest by ASP designers (Bartholomew *et al*. 2011; Denecker *et al*. 2001; Faber *et al*. 2011; Ferraris 2011; Gebser *et al*. 2015a; Gelfond and Zhang 2019; Liu *et al*. 2010; Marek and Remmel 2004; Pelov *et al*. 2007; Simons *et al*. 2002), and the utility of their incorporation in the modeling language has been realized almost simultaneously with the inception of the first ASP solving systems. In fact, aggregate expressions provide a natural syntax for expressing properties on sets of atoms collectively, which is often desired when modeling complex knowledge. For example, aggregate expressions are widely used to enforce collective conditions on guessed relations. In *graph k-colorability* (Garey and Johnson 1979), the guessed assignment of colors must be a total function, which can be enforced by means of the following constraint:

```
:- vertex(X), #count{C : assign_color(X,C)} != 1.
```

In *independent set* (Garey and Johnson 1979), the guessed set of nodes must not be smaller than a given bound $k \geq 1$, which can be enforced by the following constraint:

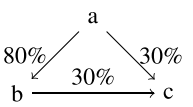
```
:- #count{X : in_independent_set(X)} < k.
```

Aggregate expressions in constraints, as in the examples above, are frequent and leave no ambiguity in their intended semantics. However, there are other frequent use cases in which aggregation functions are involved in inductive definitions, which open the possibility of using aggregation functions for reasoning about recursive data structures. For example, the number of nodes in a tree (and its subtrees) can be determined by the following rule:

```
nodes(T,C+1) :- tree(T), C = #sum{C',T' : child(T,T'), nodes(T',C')}.
```

Figure 1 shows a tree and the expected outcome for the `nodes` relation. Note that the sum is applied to a multiset, specifically to the multiset obtained by projecting on the first

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press




```
company(a). owns(a,b,80).
company(b). owns(a,c,30).
company(c). owns(b,c,30).
```

```
controls(a,b).
controls(a,c).
```

Fig. 2. Companies and their shares, their relational representation, and the control relationships.

element the following set of tuples: `{(C',T') | child(T,T') ∧ nodes(T',C')}`. For example, given `nodes(d,1)` and `nodes(e,1)`, for `T` = `b` the set is `{(1,d), (1,e)}`, and therefore the sum aggregation is applied to the multiset $[1, 1]$.

As another example, consider the *company controls* problem, originally proposed by Mumick *et al.* (1990), where a set of companies and their stock shares are given and the task is to determine the control relationships between companies. The following rule encodes such knowledge:

```
controls(A,B) :- company(A), company(B),
    #sum{S : own(A,B,S); S,C : controls(A,C), own(C,B,S), A != C} > 50.
```

Figure 2 shows an instance of this problem and the expected outcome. Non-deterministic variants of this problem naturally emerge if some of the stock shares are not fixed and associated with a cost, and a holding company wants to understand whether its companies can exert an indirect control over other companies of interest.*

The possibility to mix guesses, inductive definitions, and aggregation functions opens the gates for several alternative interpretations of the expected outcome, and thus a general semantics of aggregates and its efficient implementation have been long-standing challenges. Already in the propositional case, checking the satisfiability of an aggregate expression is an intractable problem in general, with the NP-complete *subset sum* and *subset product* problems (Garey and Johnson 1979) as special cases. The practical implication of such an intractability is the use of weakened propagation procedures, meaning that ASP solvers give up on the propagation of some deterministic inferences to achieve algorithmic efficiency, usually obtained by translational approaches (Alviano *et al.* 2015; Alviano and Leone 2015; Elkabani *et al.* 2004; Faber *et al.* 2008; Ferraris and Lifschitz 2005; Simons *et al.* 2002). For example, an aggregate expression involving a sum and the equality comparison is split into a conjunction of two aggregate expressions that are true when the sum is no more and no less than the original bound. In addition, satisfiability of aggregate expressions is in general a *non-convex* function: the expression `#sum{1 : a; -1 : b} = 0` is true for $\emptyset$, false for `{a}` and `{b}`, and true again for `{a,b}`, a pattern not shown by conjunctions of (possibly negated) atoms. In fact, satisfiability of conjunctions of positive literals is a *monotone* function (i.e., at most one transition from false to true, and no other transitions), satisfiability of conjunctions of negative literals is an *anti-monotone* function (i.e., at most one transition from true to false, and no other transitions), and satisfiability of conjunctions of literals in general is a *convex* function (i.e., at most one transition from false to true followed by at most one transition from true to false). The general picture for aggregate expressions is shown in Figure 3 and further detailed in Section 2.3.

Among the variety of semantics proposed for interpreting ASP programs with aggregates, two of them (Faber *et al.* 2011; Ferraris 2011) are implemented in widely used ASP

* https://www.mat.unical.it/aspcomp2011/FinalProblemDescriptions/CompanyControlsOptimize.

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

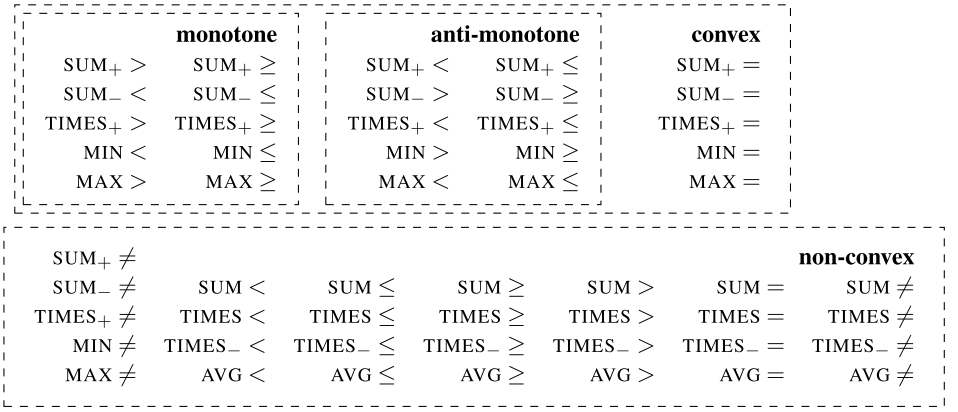


Fig. 3. Aggregate expressions classified according to the monotonicity of satisfiability. Subscript signs denote restrictions on the sign of weights; for example, SUM+ is SUM with positive weights.

solvers (Faber *et al*. 2008; Gebser *et al*. 2012). The two semantics agree for programs without negation, and are thus referred indistinctly as FFLP-answer sets in this survey. Under this syntactic restriction, FFLP-answer sets can be defined as $\subseteq$-minimal models of a program reduct obtained by deleting rules with false bodies, as in the aggregate-free case. Other prominent semantics (Denecker *et al*. 2001; Gelfond and Zhang 2019; Kemp and Stuckey 1991; Liu *et al*. 2010; Marek and Remmel 2004; Pelov *et al*. 2007; Son and Pontelli 2007) adopt constructive procedures for checking the provability of true atoms. While some of these semantics were originally defined via a notion of reduct in the non-disjunctive case, they can be viewed as semantics based on different extensions of the immediate consequence operator (van Emden and Kowalski 1976; Lloyd 1987). Although such extensions result in different answer sets in general, there are cases in which answer sets according to one definition are also answer sets according to other definitions; details on such relationships are given in Section 3.3.

Historically, aggregates appeared in database query languages and have been available almost since the inception of database systems. However, these aggregate constructs usually lacked generality and formal definitions. Notably, aggregates were not present in Relational Algebra and Relational Calculus of Codd's seminal papers (Codd 1970; 1972). It was only with Klug's work (Klug 1982) in the 1980s that aggregate expressions were cleanly integrated into Relational Algebra and Calculus. This integration was further refined by Özsoyoglu *et al*. (1987) a few years later, which also included the ability to specify set expressions. While none of these languages supported recursive definitions yet, the Relational Calculus in these papers did, however, feature a clean formalism for defining ranges and therefore avoid unsafe queries, which can be seen as a direct precursor to safety notions in the *ASP-Core-2* language specification (Calimeri *et al*. 2019).

Database query languages allowing for recursive definitions and aggregates were based on Datalog, essentially a deterministic fragment of ASP, in which every program is associated with a unique intended model, as for example the perfect model of Datalog programs with stratified negation (Apt *et al*. 1987). Preserving the unique model property is non-trivial in presence of aggregate expressions because of nonmonotonicity of

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

their satisfiability functions, which may lead to programs having no model, and also to programs having several equally justifiable models. Often in the literature, uniqueness of the intended model was guaranteed by imposing syntactic restrictions on the use of aggregates, usually resulting in some stratification of aggregations or monotonicity conditions; see for example the papers by Mumick *et al.* (1990) and Ross (1994). Other works in the literature extended the well-founded semantics to logic programs with aggregates (Kemp and Stuckey 1991; Pelov *et al.* 2007; Van Gelder 1992), hence considering a third truth value to reason on undefined expressions whose truth or falsity is taken as unknown. Semantics for aggregate expressions were also given in terms of translation into aggregate-free (sub)programs, and then applying the well-founded or stable model semantics to the resulting programs. Such translations first addressed extrema predicates only, that is, `#min` and `#max` aggregate expressions (Ganguly *et al.* 1995; Sudarshan and Ramakrishnan 1991), and were later also presented for monotonic versions of `#count` and `#sum` aggregate expressions (Mazuran *et al.* 2013; Zaniolo *et al.* 2017).

The first attempt to define a variant of the stable model semantics that includes aggregates was given by Kemp and Stuckey (1991). The proposed generalization of the Gelfond–Lifschitz reduct was to treat aggregate expressions in the same way as negated atoms. From today's point of view, this was of course bound to be problematic, as some aggregate expressions behave like positive rather than negative literals (and some aggregate expressions behave like neither of them). As a simple example, consider the program

```
a :- #sum{1 : a} > 0.
```

which should intuitively be equivalent to

```
a :- a.
```

but will have two answer sets according to Kemp and Stuckey (1991): the expected $\emptyset$ plus the unexpected `{a}` (as in that case the reduct results in `a.`). Still, this seminal work should be regarded as the starting point of what we discuss in the sequel.

The structure of this survey is as follows. Section 2 introduces the language of logic programs with aggregate expressions to be studied (Section 2.1), investigates the complexity of deciding satisfiability of aggregate expressions (Section 2.2), and classifies aggregate expressions according to their monotonicity (Section 2.3). Contrary to this introduction, which used examples of symbolic programs with object variables, the main part of the survey focuses on a propositional language, where object variables are assumed to be eliminated as usual by a grounding procedure. Section 3 discusses model-based and construction-based answer set semantics for logic programs containing aggregates (presented in Section 3.1 or Section 3.2, respectively), and outlines semantic correspondences obtained for programs with aggregate expressions of particular monotonicity (Section 3.3). The computational complexity of common reasoning tasks is studied in Section 4, addressing answer set checking (Section 4.1) and answer set existence (Section 4.2). Finally, Section 5 discusses related work beyond the presented aggregate semantics and properties, including first-order semantics for symbolic programs with aggregates over object variables, rewriting methods for turning programs containing (sophisticated) aggregates into simpler and often aggregate-free representations, and tools to extend logic programs by custom aggregates.

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

2 Rules and programs with aggregates

This section addresses basic syntactic and semantic concepts of propositional logic programs with aggregate expressions. Section 2.1 introduces the syntax of aggregate expressions as well as rules and logic programs including them, and further specifies the satisfaction of these constructs by interpretations over propositional atoms. In Section 2.2, we investigate the complexity of deciding whether an aggregate expression is satisfiable, that is, checking the existence of some interpretation satisfying the aggregate expression. Finally, Section 2.3 turns to monotonicity properties of aggregate expressions, which characterize their satisfaction w.r.t. evolving interpretations.

2.1 Syntax and satisfaction of aggregates

In order to characterize the similarities and differences between the main aggregate semantics proposed for propositional logic programs (Denecker *et al.* 2001; Faber *et al.* 2011; Ferraris 2011; Gelfond and Zhang 2019; Liu *et al.* 2010; Marek and Remmel 2004; Pelov *et al.* 2007), we consider a set $\mathcal{P}$ of *propositional atoms*. An *aggregate expression* has the form

$$\text{AGG}[w_1 : p_1, \ldots, w_n : p_n] \odot w_0, \tag{1}$$

where $n \geq 0$ and

- $\text{AGG} \in \{\text{SUM}, \text{TIMES}, \text{AVG}, \text{MIN}, \text{MAX}\}$ is the name of an *aggregation function*,
- $w_0, w_1, \ldots, w_n$ are integers, where w_i is a *weight*, for $1 \leq i \leq n$, and w_0 is a *bound*,
- $p_1, \ldots, p_n$ are propositional atoms from $\mathcal{P}$, and
- $\odot \in \{<, \leq, \geq, >, =, \neq\}$ is a comparison operator.

For an aggregate expression A as in (1), by $\text{AT}(A) = \{p_1, \ldots, p_n\}$ we denote the set of propositional atoms occurring in A.

Note that the common COUNT aggregation function is a special case of SUM such that each weight is 1. Similarly, EVEN and ODD are special cases of TIMES such that each weight is -1, the bound is 1, and $\odot$ is $=$ or $\neq$, respectively. More liberal aggregate concepts also account for rational or real-valued weights, formulas instead of propositional atoms only, as well as a second comparison operator and bound (Calimeri *et al.* 2019; Ferraris 2011). Concerning knowledge representation, Bartholomew *et al.* (2011), Gebser *et al.* (2015a), and Harrison and Lifschitz (2019) provide first-order generalizations of the aggregate semantics by Faber *et al.* (2011) and Ferraris (2011), while Gelfond and Zhang (2019) as well as Pelov *et al.* (2007) genuinely accommodate first-order aggregates. Specific aggregates in the form of cardinality and weight constraints are elaborated by Ferraris and Lifschitz (2005), Liu and You (2013), and Simons *et al.* (2002).

A *rule* has the form

$$p_1 \vee \cdots \vee p_m \leftarrow A_1 \wedge \cdots \wedge A_n, \tag{2}$$

where $m \geq 0$, $n \geq 0$, $p_1, \ldots, p_m$ are propositional atoms, and $A_1, \ldots, A_n$ are aggregate expressions. For a rule r as in (2), let $H(r) = \{p_1, \ldots, p_m\}$ and $B(r) = \{A_1, \ldots, A_n\}$ denote the *head* and *body* of r. We say that r is *non-disjunctive* if $|H(r)| \leq 1$, in which case the consequent of (2) is either of the form p_1 or $\bot$, the latter representing an empty disjunction, that is, a contradiction. The rule r is also called a *constraint* if $H(r) = \emptyset$, and a *fact* if $B(r) = \emptyset$, where an empty conjunction in the antecedent of (2) is denoted by

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

$\top$. While the bodies of rules do not explicitly allow for (negated) propositional atoms p, they can easily be represented by aggregate expressions of the form $\text{SUM}[1 : p] > 0$ or $\text{SUM}[1 : p] < 1$, so that the syntax at hand includes disjunctive rules (Eiter and Gottlob 1995; Gelfond and Lifschitz 1991). A *program* P is a finite set of rules, and we call P a *non-disjunctive* program if each $r \in P$ is non-disjunctive.

An *interpretation* $X \subseteq \mathcal{P}$ is represented by the set of its true propositional atoms. It allows us to map each aggregate expression A of the form (1) to a multiset $\omega(A, X) = [w_i \mid 1 \leq i \leq n, p_i \in X]$ of weights, and then to an expression $\alpha(A, X)$ where

- for $\text{AGG} = \text{SUM}$, $\alpha(A, X) = \sum_{w \in \omega(A,X)} w$,
- for $\text{AGG} = \text{TIMES}$, $\alpha(A, X) = \prod_{w \in \omega(A,X)} w$,
- for $\text{AGG} = \text{AVG}$, $\alpha(A, X) = \begin{cases} \sum_{w \in \omega(A,X)} w/|\omega(A, X)|, & \text{if } |\omega(A, X)| > 0, \\ \varepsilon & \text{, if } |\omega(A, X)| = 0, \end{cases}$
- for $\text{AGG} = \text{MIN}$, $\alpha(A, X) = \min\{w \mid w \in \omega(A, X)\}$, and
- for $\text{AGG} = \text{MAX}$, $\alpha(A, X) = \max\{w \mid w \in \omega(A, X)\}$.

Note that the average over a (non-empty) multiset of weights can be a rational number, for example, $(1 + 2)/|[1, 2]| = 3/2$. The ε outcome of taking the average over the empty multiset stands for undefined, while other aggregation functions map the empty multiset to their neutral elements: $\sum_{w \in []} w = 0$, $\prod_{w \in []} w = 1$, $\min\{w \mid w \in []\} = \infty$, and $\max\{w \mid w \in []\} = -\infty$.

For $\odot \in \{<, \leq, \geq, >, =, \neq\}$, by $X \models (\alpha(A, X) \odot w_0)$ we denote that $\alpha(A, X) \odot w_0$ holds, and write $X \not\models (\alpha(A, X) \odot w_0)$ otherwise. Note that $X \models (\infty \geq w_0)$, $X \models (\infty > w_0)$, $X \models (\infty \neq w_0)$, $X \models (-\infty < w_0)$, $X \models (-\infty \leq w_0)$, and $X \models (-\infty \neq w_0)$ apply independently of the (integer) bound w_0, while $X \not\models (\infty < w_0)$, $X \not\models (\infty \leq w_0)$, $X \not\models (\infty = w_0)$, $X \not\models (-\infty \geq w_0)$, $X \not\models (-\infty > w_0)$, $X \not\models (-\infty = w_0)$, and $X \not\models (\varepsilon \odot w_0)$, for $\odot \in \{<, \leq, \geq, >, =, \neq\}$, are the cases in which $\alpha(A, X) \odot w_0$ cannot hold.

An aggregate expression A of the form (1) is *satisfied* by an interpretation X, also written $X \models A$, if $X \models (\alpha(A, X) \odot w_0)$, and otherwise $X \not\models A$ denotes that A is *unsatisfied* by X. The body of a rule r as in (2) is satisfied by X, written $X \models B(r)$, if $X \models A$ for each $A \in B(r)$, and unsatisfied by X, indicated by $X \not\models B(r)$, otherwise. Likewise, the head of r is satisfied by X, $X \models H(r)$, if $H(r) \cap X \neq \emptyset$, and otherwise $X \not\models H(r)$ expresses that $H(r)$ is unsatisfied by X. The rule r is satisfied by X, that is, $X \models r$, if $X \models B(r)$ implies $X \models H(r)$, while r is unsatisfied by X, $X \not\models r$, otherwise. Note that we have $X \not\models H(r)$ when r is a constraint, so that $X \not\models B(r)$ is necessary to satisfy a constraint r by X. For a fact r, $X \models B(r)$ is instantaneous, and r is satisfied by X only if $X \models H(r)$. Finally, X is a *model* of a program P, denoted by $X \models P$, if $X \models r$ for every $r \in P$, and we write $X \not\models P$ otherwise.

Example 1

Consider the program P_1 consisting of three rules as follows:

$$p \leftarrow \text{SUM}[1 : p, -1 : q] \geq 0 \qquad (r_1)$$

$$p \leftarrow \text{SUM}[1 : q] > 0 \qquad (r_2)$$

$$q \leftarrow \text{SUM}[1 : p] > 0 \qquad (r_3)$$

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Intuitively, rule r_1 requires p to be true if p is true or q is false, rule r_2 requires p true if q is true, and rule r_3 requires q true if p is true. The four interpretations over the set $\mathcal{P} = \{p, q\}$ of propositional atoms are $X_1 = \emptyset$, $X_2 = \{p\}$, $X_3 = \{q\}$, and $X_4 = \{p, q\}$. We have that $\alpha(\text{SUM}[1 : p, -1 : q] \geq 0, X_1) = 0$, so that $X_1 \models B(r_1)$ but $X_1 \not\models H(r_1)$. For X_2 and X_3, $\alpha(\text{SUM}[1 : p] > 0, X_2) = 1$ and $\alpha(\text{SUM}[1 : q] > 0, X_3) = 1$ yield $X_2 \models B(r_3)$ and $X_3 \models B(r_2)$, while $X_2 \not\models H(r_3)$ and $X_3 \not\models H(r_2)$. That is, neither X_1, X_2, nor X_3 is a model of P_1. The remaining interpretation X_4 is such that $X_4 \models B(r)$ and $X_4 \models H(r)$ for every $r \in P_1$, so that X_4 is a model of P_1. ■

2.2 Satisfiability of aggregates

With the syntax and satisfaction of aggregate expressions at hand, let us turn to the complexity of deciding whether an aggregate expression A is *satisfiable*, that is, checking whether there is some interpretation $X \subseteq \mathcal{P}$ such that $X \models A$. To this end, we first note that determining the smallest and greatest feasible outcome, denoted by $\text{LB}(A)$ and $\text{UB}(A)$, of the aggregation function in A of the form (1) can be accomplished as follows (where ε denotes an *undefined outcome*):

- for $\text{AGG} = \text{SUM}$, $\text{LB}(A) = \sum_{p \in \text{AT}(A), \alpha(A,\{p\}) < 0} \alpha(A, \{p\})$,
  $\text{UB}(A) = \sum_{p \in \text{AT}(A), \alpha(A,\{p\}) > 0} \alpha(A, \{p\})$,
- for $\text{AGG} = \text{TIMES}$, let $\pi_{\pm} = \prod_{p \in \text{AT}(A), \alpha(A,\{p\}) \neq 0} \alpha(A, \{p\})$,
  $\pi_0 = \prod_{p \in \text{AT}(A), \alpha(A,\{p\}) = 0} \alpha(A, \{p\})$, and
  $w = \max\{\alpha(A, \{p\}) \mid p \in \text{AT}(A), \alpha(A, \{p\}) < 0\}$ in
  $$\text{LB}(A) = \begin{cases} \pi_{\pm} & \text{, if } \pi_{\pm} < 0, \\ \pi_{\pm}/w & \text{, if } \pi_{\pm} > 0 \text{ and } w \neq -\infty, \\ \pi_0 & \text{, if } w = -\infty, \end{cases}$$
  $$\text{UB}(A) = \begin{cases} \pi_{\pm} & \text{, if } \pi_{\pm} > 0, \\ \pi_{\pm}/w & \text{, if } \pi_{\pm} < 0, \end{cases}$$
- for $\text{AGG} = \text{AVG}$, $\text{LB}(A) = \begin{cases} \min\{\alpha(A, \{p\}) \mid p \in \text{AT}(A)\} & \text{, if } \text{AT}(A) \neq \emptyset, \\ \varepsilon & \text{, if } \text{AT}(A) = \emptyset, \end{cases}$
  $\text{UB}(A) = \begin{cases} \max\{\alpha(A, \{p\}) \mid p \in \text{AT}(A)\} & \text{, if } \text{AT}(A) \neq \emptyset, \\ \varepsilon & \text{, if } \text{AT}(A) = \emptyset, \end{cases}$
- for $\text{AGG} = \text{MIN}$, $\text{LB}(A) = \min\{\alpha(A, \{p\}) \mid p \in \text{AT}(A)\}$,
  $\text{UB}(A) = \infty$,
- for $\text{AGG} = \text{MAX}$, $\text{LB}(A) = -\infty$,
  $\text{UB}(A) = \max\{\alpha(A, \{p\}) \mid p \in \text{AT}(A)\}$.

For the SUM aggregation function, the lower and upper bound are obtained by summing over all atoms associated with a negative or positive weight, respectively. In case of AVG, we just pick an atom with the smallest or greatest weight, provided that some atom occurs in A. For MIN and MAX, either the upper or the lower bound is trivial, and the respective

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

counterpart is obtained by applying the aggregation function to all weights. The TIMES aggregation function is the most sophisticated, as negative weights may swap the sign of its outcome. If the product $\pi_{\pm}$ over all atoms with a non-zero weight is negative, $\pi_{\pm}$ yields the lower bound, while it is otherwise divided by the greatest negative weight w (i.e., smallest absolute value), provided that it exists, to achieve a negative outcome. In the remaining case that no negative outcome is feasible, the lower bound π_0 is zero when there is some atom with the weight zero, or it defaults to one. Finally, the upper bound is given by $\pi_{\pm}$ if it is positive, and otherwise we divide $\pi_{\pm}$ by the greatest negative weight w to obtain the greatest positive outcome.

Given $\text{LB}(A)$ and $\text{UB}(A)$ for an aggregate expression A of the form (1), $\text{LB}(A) = \text{UB}(A) = \varepsilon$ indicates that A is unsatisfiable regardless of the comparison operator $\odot$, which applies only when A matches $\text{AVG}[] \odot w_0$. Otherwise, if the comparison operator $\odot$ is $<$ or $\leq$, we have that A is satisfiable if and only if $\text{LB}(A) < w_0$ or $\text{LB}(A) \leq w_0$ holds, respectively. The comparison operators $>$ and $\geq$ are dual, and an aggregate expression A including one of them is satisfiable if and only if either $\text{UB}(A) > w_0$ or $\text{UB}(A) \geq w_0$ holds. In case $\odot$ is $\neq$, we have that A is satisfiable if and only if $\text{LB}(A) \neq w_0$ or $\text{UB}(A) \neq w_0$ holds. These considerations show that checking the satisfiability of an aggregate expression with a comparison operator other than $=$ is tractable, that is, in the complexity class P. The situation gets more involved when $\odot$ is $=$, where the aggregation functions MIN and MAX still allow for tractable decisions: an aggregate expression A with AGG either MIN or MAX and $=$ for $\odot$ is satisfiable if and only if $\alpha(A, \{p\}) = w_0$ for some $p \in \text{AT}(A)$. Unlike that, for $\text{AGG} = \text{SUM}$ and $\text{AGG} = \text{TIMES}$, deciding whether there is some (non-empty) subset X of $\text{AT}(A)$ such that $\alpha(A, X) = w_0$ amounts to the NP-complete *subset sum* or *subset product* problem (Garey and Johnson 1979). Concerning $\text{AGG} = \text{AVG}$, subset sum can be reduced to checking whether an aggregate expression A with bound 0 is satisfiable: $\text{SUM}[w_1 : p_1, \ldots, w_n : p_n] = w_0$ with positive integers $w_0, \ldots, w_n$ is satisfiable if and only if $\text{AVG}[w_1 : p_1, \ldots, w_n : p_n, -w_0 : p] = 0$ is satisfiable, where $p \in \mathcal{P}$ is some new atom. As constants like $w_0 + 1$ can be added to each weight and the bound of an aggregate expression with AVG, we have NP-completeness of checking whether A with $\text{AGG} \in \{\text{SUM}, \text{TIMES}, \text{AVG}\}$ and $=$ for $\odot$ is satisfiable, which applies regardless of whether the included weights and bound are restricted to be positive or negative integers only.

Example 2
Consider aggregate expressions constructed as follows:

$$\text{SUM}[1 : p_1, 3 : p_2, 3 : p_3, -4 : p_4] \odot w_0 \tag{A_1}$$

$$\text{TIMES}[0 : p_1, 3 : p_2, -2 : p_3, -4 : p_4] \odot w_0 \tag{A_2}$$

$$\text{AVG}[1 : p_1, 2 : p_2, 3 : p_3, 6 : p_4] \odot w_0 \tag{A_3}$$

$$\text{MIN}[0 : p_1, 3 : p_2, -2 : p_3, -4 : p_4] \odot w_0 \tag{A_4}$$

$$\text{MAX}[1 : p_1, 3 : p_2, 3 : p_3, -4 : p_4] \odot w_0. \tag{A_5}$$

By summing up negative or positive weights, respectively, we get $\text{LB}(A_1) = -4$ and $\text{UB}(A_1) = 7$. Hence, A_1 is satisfiable for $\odot = <$ and any bound w_0 greater than -4, for $\odot = \leq$ and w_0 not smaller than -4, for $\odot = \geq$ and w_0 not greater than 7, for $\odot = >$ and w_0 smaller than 7, and for $\odot = \neq$ with any bound w_0. When $\odot$ is $=$, satisfiability

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

is more intricate, and one can check that A_1 is satisfiable for bounds w_0 other than -2 and 5 in the range from $\text{LB}(A_1) = -4$ to $\text{UB}(A_1) = 7$.

The product of non-zero weights, two of which are negative, for A_2 is 24. To obtain the lower bound for A_2, we have thus to divide by the greatest negative weight -2, leading to $\text{LB}(A_2) = -12$, while $\text{UB}(A_2) = 24$. For $\odot \in \{<, \leq, \geq, >, \neq\}$, satisfiability is determined similar to A_1, yet w.r.t. the lower and upper bound for A_2. When $\odot$ is $=$, we have that A_2 is satisfiable for $w_0 \in \{-12, -6, -4, -2, 0, 1, 3, 8, 24\}$, and unsatisfiable otherwise.

Concerning A_3, we get $\text{LB}(A_3) = 1$ and $\text{UB}(A_3) = 6$, and satisfiability for $\odot \in \{<, \leq, \geq, >, \neq\}$ w.r.t. these bounds is analogous. As 1, 2, 3, 4, and 6 are the feasible integer outcomes, A_3 with $=$ for $\odot$ is satisfiable for such outcomes as w_0, and unsatisfiable for any other bound w_0.

Turning to A_4 and A_5, the lower and upper bounds are $\text{LB}(A_4) = -4$, $\text{UB}(A_4) = \infty$, $\text{LB}(A_5) = -\infty$, and $\text{UB}(A_5) = 3$. For $\odot \in \{<, \leq, \geq, >, \neq\}$, satisfiability is determined similar to A_1, where A_4 is satisfiable for $\odot = \geq$, $\odot = >$, and $\odot = \neq$ regardless of the bound w_0, and likewise A_5 for $\odot = <$, $\odot = \leq$, and $\odot = \neq$. When $\odot$ is $=$, we have that A_4 and A_5 are satisfiable for bounds w_0 matching their contained weights, that is, $w_0 \in \{-4, -2, 0, 3\}$ or $w_0 \in \{-4, 1, 3\}$, respectively. ■

2.3 Monotonicity of aggregates

A relevant property related to satisfiability concerns the (non)monotonicity of aggregate expressions (cf. Faber *et al.* (2011); Liu *et al.* (2010)) and the notion of convex aggregates introduced by Liu and Truszczyński (2006). An aggregate expression A is *convex* if $X \subseteq Y$, $X \models A$ and $Y \models A$ imply $Z \models A$ for any interpretation Z between X and Y, that is, $X \models A$ and $Y \models A$ yield $Z \models A$ for all $X \subseteq Z \subseteq Y$. Otherwise, the aggregate expression is *non-convex*. Two frequent special cases of convexity are given by *monotone* and *anti-monotone* aggregate expressions, for which $X \models A$ implies that $\mathcal{P} \models A$ or $\emptyset \models A$, respectively. That is, a monotone aggregate expression A remains satisfied when adding (true) propositional atoms to an interpretation X satisfying A, while the satisfaction of an anti-monotone A is preserved when propositional atoms from X are falsified.

Specific monotonicity properties that follow from the aggregation functions and comparison operators of aggregate expressions are summarized in Figure 3 in the introduction. For the SUM and TIMES aggregation functions, restrictions to positive or negative weights and bounds affect monotonicity, and the subscripts $_+$ and $_-$ indicate such particular conditions. Unlike that, the signs of weights and bounds are immaterial for AVG, MIN, and MAX, and we also disregard special cases like the unsatisfiability of $\text{AVG}[] \odot w_0$, which makes aggregate expressions with AVG over the empty multiset monotone, anti-monotone, and convex.

While aggregate expressions with SUM are in general non-convex regardless of the comparison operator, restrictions to positive or negative weights and bounds make them convex for comparison operators other than $\neq$, given that the sum of weights can merely increase or decrease when propositional atoms are added to an interpretation. More specifically, positive weights and bounds lead to an anti-monotone aggregate expression for $<$ or $\leq$ as comparison operator, and to a monotone aggregate expression for $\geq$ or $>$, where dual properties apply in case of negative weights and bounds only. For either

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

restriction, the comparison operator $\neq$ does not lead to more regular monotonicity though because a respective aggregate expression may be satisfied by the empty interpretation $\emptyset$, become unsatisfied when atoms whose weights sum up to the bound w_0 are made true, and then get satisfied again by adding further propositional atoms to the interpretation.

Aggregate expressions with TIMES are generally non-convex, even if weights and bounds are restricted to be negative only. This is due to the condition that negative weights swap the sign of the outcome, so that satisfaction may switch back and forth for any comparison operator. With positive weights and bounds (excluding zero), the outcome increases when more propositional atoms are included in an interpretation. Hence, such aggregate expressions yield similar monotonicity as SUM with positive weights and bounds, and entries for SUM_+ and TIMES_+ in Figure 3 match.

For the remaining aggregation functions, AVG, MIN, and MAX, the signs of weights do not make a difference regarding monotonicity. In fact, the outcome of AVG may increase or decrease when propositional atoms are added to an interpretation, so that corresponding aggregate expressions are generally non-convex regardless of the comparison operator. The outcomes of MIN and MAX monotonically decrease or increase, respectively, with growing interpretations, leading to (anti-)monotonicity for the comparison operators $<$ or $\leq$ and $\geq$ or $>$, as well as convexity for $=$. Similar to the other aggregation functions, no matter whether weights and bounds are positive, negative, or unrestricted, $\neq$ as comparison operator makes aggregate expressions with MIN and MAX non-convex in general, as they are satisfied by the empty interpretation $\emptyset$, may become unsatisfied and then satisfied again when the interpretation is extended by propositional atoms.

Example 3
The aggregate expression

$$\text{SUM}[1 : p_1, 2 : p_2, 2 : p_3, 3 : p_4] \odot 5 \tag{A_6}$$

is monotone for $\odot = \geq$ or $\odot = >$, anti-monotone for $\odot = <$ or $\odot = \leq$, convex when $\odot$ is $=$, and non-convex with $\neq$ for $\odot$. The three mutually incomparable interpretations $\{p_1, p_2, p_3\}$, $\{p_2, p_4\}$, and $\{p_3, p_4\}$ satisfying A_6 with $=$ for $\odot$ meet the condition for convexity, but neither the additional requirements for monotonicity nor anti-monotonicity. Moreover, they witness the non-convexity of A_6 when $\odot$ is $\neq$, as there are interpretations with both fewer and more propositional atoms such that their (positive) weights do not sum up to the bound 5 and satisfy A_6 for $\odot = \neq$. ■

3 Answer set semantics of aggregates

This section characterizes the main answer set semantics proposed for logic programs with aggregate expressions (Denecker *et al*. 2001; Faber *et al*. 2011; Ferraris 2011; Gelfond and Zhang 2019; Liu *et al*. 2010; Marek and Remmel 2004; Pelov *et al*. 2007). Their common essence is to extend the notion of provability of the true propositional atoms in a model to programs that include aggregate expressions. In Section 3.1, we review aggregate semantics based on $\subseteq$-minimal models of a program reduct. Section 3.2 then gathers several constructive approaches to capture the provability of true atoms. Correspondences between different aggregate semantics w.r.t. the monotonicity of aggregate expressions are analyzed in Section 3.3.

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

3.1 Model-based semantics

In line with Gelfond and Lifschitz (1988), Gelfond and Lifschitz (1991), we let the *reduct* of a program P relative to an interpretation $X \subseteq \mathcal{P}$ be $P^X = \{r \in P \mid X \models B(r)\}$. In other words, the reduct is obtained from P by dropping rules whose bodies are unsatisfied by X. (Note that the original definition of reduct provided by Gelfond and Lifschitz (1988) also removes negative literals, which are not part of the language considered in this paper.) Hence, we have that X is a model of P if and only if X is a model of P^X.

Different answer set semantics inspect the reduct in specific ways to check the provability of true atoms. We note that our reduct notion matches the definitions given by Faber *et al.* (2011) and Ferraris (2011), as it does not eliminate false atoms or falsified expressions, respectively. The latter would happen for negated atoms with the seminal Gelfond–Lifschitz reduct (Gelfond and Lifschitz 1988; 1991), and entirely with the reduct by Ferraris (2011). However, the syntax of aggregate expressions and rules in (1)–(2) is chosen such that specific reducts do not make a difference regarding answer sets.

The model-theoretic approaches by Faber *et al.* 2011 and Ferraris (2011) agree in focusing on $\subseteq$-minimal models of the reduct P^X of a program P relative to an interpretation $X \subseteq \mathcal{P}$. We use the convention to indicate particular semantics by their proposers, and call X an FFLP-*answer set* of P if X is a $\subseteq$-minimal model of P^X. That is, an FFLP-answer set of P fulfills two conditions:

- X is a model of P, that is, $X \models P$, and
- for any interpretation $Y \subset X$, we have that $Y \not\models P^X$.

While the first condition is equivalent to $X \models P^X$, exchanging $Y \not\models P^X$ for $Y \not\models P$ in the second condition would yield a different semantics.

Example 4
Consider the program P_2 containing the following two rules:

$$p \leftarrow \text{SUM}[1 : p] > 0 \qquad (r_4)$$
$$p \leftarrow \text{SUM}[1 : p] < 1 \qquad (r_5)$$

Intuitively, rule r_4 requires p true if p is true, and rule r_5 requires p true if p is false. The interpretation $X_1 = \emptyset$ is not a model of P_2 because $\alpha(\text{SUM}[1 : p] < 1, X_1) = 0$ yields that $X_1 \models B(r_5)$ but $X_1 \not\models H(r_5)$. For $X_2 = \{p\}$, we have that $\alpha(\text{SUM}[1 : p] < 1, X_2) = 1$, so that $X_2 \not\models B(r_5)$. Hence, $P_2^{X_2}$ consists of the rule r_4 only. Since $X_1 \subset X_2$ is such that $\alpha(\text{SUM}[1 : p] > 0, X_1) = 0$, $X_1 \not\models B(r_4)$, and $X_1 \models P_2^{X_2}$, the interpretation X_1 shows that X_2 is not a $\subseteq$-minimal model of $P_2^{X_2}$. As a consequence, the program P_2 does not have any FFLP-answer set. However, note that X_2 is the $\subseteq$-minimal model of P_2 in view of $X_2 \models P_2$ and $X_1 \not\models P_2$. ■

Example 5
Reconsider the program P_1 from Example 1:

$$p \leftarrow \text{SUM}[1 : p, -1 : q] \geq 0 \qquad (r_1)$$
$$p \leftarrow \text{SUM}[1 : q] > 0 \qquad (r_2)$$
$$q \leftarrow \text{SUM}[1 : p] > 0 \qquad (r_3)$$

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

As checked in Example 1, we have that $X_4 = \{p, q\}$ is the unique model of P_1 (over the set $\mathcal{P} = \{p, q\}$ of propositional atoms). Given that $X_4 \models B(r)$ for every $r \in P_1$, we obtain $P_1^{X_4} = P_1$, which shows that X_4 is an FFLP-answer set of P_1. ■

Example 6
Consider the program P_3 consisting of the following rules:

$$x_1 \leftarrow \text{SUM}[1 : y_1] < 1 \tag{r_6}$$
$$y_1 \leftarrow \text{SUM}[1 : x_1] < 1 \tag{r_7}$$
$$x_2 \leftarrow \text{SUM}[1 : y_2] < 1 \tag{r_8}$$
$$y_2 \leftarrow \text{SUM}[1 : x_2] < 1 \tag{r_9}$$
$$z_1 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{10}}$$
$$z_2 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{11}}$$
$$p \leftarrow \text{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5 \tag{r_{12}}$$
$$\bot \leftarrow \text{SUM}[1 : p] < 1 \tag{r_{13}}$$

As will be clarified later in Section 4 and Example 14, P_3 (under FFLP-, LPST-, or DPB-answer set semantics) encodes an instance of *generalized subset sum* (Berman *et al.* 2002), specifically

$$\exists y_1 y_2 \forall z_1 z_2 (1 \cdot y_1 + 2 \cdot y_2 + 2 \cdot z_1 + 3 \cdot z_2 \neq 5).$$

Intuitively, rules r_6 and r_7 are intended to select exactly one of x_1 and y_1, and similarly rules r_8 and r_9 are intended to select exactly one of x_2 and y_2; these rules represent the existential variables of the generalized subset sum instance. Rules r_{10} and r_{11} require z_1 and z_2 to be true if p is true; these rules represent the universal variables of the generalized subset sum instance. Rule r_{13} enforces p true, but does not provide a justification for it, which instead must be provided by rule r_{12}, encoding the inequality of the generalized subset sum instance. Note that P_3 follows the *saturation technique* by Eiter and Gottlob (1995) to express that z_1 and z_2 are universally quantified: z_1 and z_2 must be true because of p (they are saturated), while p (and z_1, z_2) are provable if and only if the universal quantification holds w.r.t. the chosen (true) subset of $\{y_1, y_2\}$.

Let us focus on interpretations that are potential candidates for FFLP-answer sets of P_3. In view of the constraint r_{13}, each model of P_3 must include the atom p, and then the atoms z_1 and z_2 because of the rules r_{10} and r_{11}. Regarding x_i and y_i for $1 \leq i \leq 2$, having both atoms false yields unsatisfied rules r_6 and r_7 or r_8 and r_9, while both atoms true lead to a reduct excluding r_6 and r_7 or r_8 and r_9. In the latter case, falsifying x_i and y_i gives a smaller model of the reduct, so that the original interpretation cannot be an FFLP-answer set of P_3.

The previous considerations leave the models $X_1 = \{x_1, x_2, z_1, z_2, p\}$, $X_2 = \{x_1, y_2, z_1, z_2, p\}$, $X_3 = \{y_1, x_2, z_1, z_2, p\}$, and $X_4 = \{y_1, y_2, z_1, z_2, p\}$ as potential FFLP-answer sets of P_3. For each of these interpretations, the reduct includes either of the rules r_6 or r_7 as well as r_8 or r_9, so that falsifying an atom x_i or y_i leads to unsatisfaction of the rule with the atom in the head. A smaller model of the reduct must thus falsify p

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

along with an appropriate combination of the atoms z_1 and z_2, in order to establish the unsatisfaction of $B(r_{12})$. Note that the reduct does not include the constraint r_{13}, which enables the falsification of p, provided that $B(r_{12})$ is unsatisfied.

For X_1, the observation that $\alpha(\text{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5, X_1) = 5$ yields $X_1 \not\models B(r_{12})$ and $r_{12} \notin P_3^{X_1}$, so that $Y_1 \models P_3^{X_1}$, for each $\{x_1, x_2\} \subseteq Y_1 \subseteq X_1 \setminus \{p\}$, disproves X_1 to be an FFLP-answer set of P_3. Regarding X_2 and X_4, we have that $Y_2 = X_2 \setminus \{z_1, p\} = \{x_1, y_2, z_2\}$ and $Y_4 = X_4 \setminus \{z_2, p\} = \{y_1, y_2, z_1\}$ furnish counterexamples satisfying the respective reduct in view of $Y_2 \not\models B(r_{12})$ and $Y_4 \not\models B(r_{12})$. It thus remains to investigate X_3, where $P_3^{X_3} = \{r_7, r_8, r_{10}, r_{11}, r_{12}\}$ and $\alpha(\text{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5, X_3) = 6$. That is, an interpretation $Y_3 \subset X_3$ with $\alpha(\text{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5, Y_3) = 5$ can only be obtained by falsifying y_1, which leads to $Y_3 \not\models r_7$. In turn, $Y \models B(r_{12})$ is the case for every model $Y \subseteq X_3$ of $P_3^{X_3}$, leaving $Y = X_3$ as the unique such interpretation. Hence, we conclude that X_3 is a $\subseteq$-minimal model of $P_3^{X_3}$ and thus an FFLP-answer set of P_3. ■

3.2 Construction-based semantics

While the aggregate semantics by Faber *et al.* (2011) and Ferraris (2011) build on $\subseteq$-minimal models of a reduct, without considering the calculation of such models in the first place, the semantics given by Gelfond and Zhang (2019), Liu *et al.* 2010, Marek and Remmel (2004), and Denecker *et al.* 2001 focus on constructive characterizations for checking the provability of true atoms. In particular, the semantics by Gelfond and Zhang (2019) and Liu *et al.* (2010), referred to as GZ- and LPST-answer set semantics indicated by their proposers, are based on a monotone fixpoint construction to check the stability of an answer set candidate X, which amounts to testing whether all atoms of X (and only those) can be constructed/derived from the rules, assuming that atoms not contained in X are fixed to false. The family of semantics by Pelov *et al.* (2007), defined in terms of approximation fixpoint theory, provides a strongly related approach: for each suitable choice of a three-valued truth assignment function mapping each aggregate expression to true, false, or unknown w.r.t. a partial interpretation, the framework induces an answer set, a well-founded, and a Kripke–Kleene semantics. As shown by Vanbesien *et al.* (2021), the particular truth assignment functions for *trivial* and *ultimate* approximating aggregates (Pelov *et al.* 2007) lead exactly to the GZ- or LPST-answer set semantics, respectively. In addition, Pelov *et al.* (2007) proposed a truth assignment function for *bound* approximating aggregates, based on evaluating lower and upper bounds like $\text{LB}(A)$ and $\text{UB}(A)$ in Section 2.2, which achieves provability for more answer set candidates than the GZ-answer set semantics without increasing the computational complexity. Given such close relationships, we do not separately address trivial, ultimate, and bound approximating aggregates, but focus on the so-called *ultimate semantics*, originally introduced in earlier work (Denecker *et al.* 2001), as a complementary instance of the framework by Pelov *et al.* (2007).

In the following, we describe the constructions characterizing the aforementioned semantics by dedicated extensions of the well-known immediate consequence operator $\mathcal{T}_P$ (van Emden and Kowalski 1976; Lloyd 1987) to programs with aggregate expressions. We start by specifying particular notions of satisfaction, again indicated by the proposers of

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

a semantics, of an aggregate expression A relative to two interpretations $Y \subseteq \mathcal{P}, X \subseteq \mathcal{P}$, where X is an answer set candidate and Y is the reconstructed model:

- $(Y, X) \models_{\mathrm{GZ}} A$ if $X \models A$ and $\mathrm{AT}(A) \cap Y = \mathrm{AT}(A) \cap X$,
- $(Y, X) \models_{\mathrm{LPST}} A$ if $Z \models A$ for every interpretation $Y \subseteq Z \subseteq X$, and
- $(Y, X) \models_{\mathrm{MR}} A$ if $X \models A$ and there is some interpretation $Z \subseteq Y$ such that $Z \models A$.

When $Y \subseteq X$, the satisfaction relations $\models_{\mathrm{GZ}}$, $\models_{\mathrm{LPST}}$, and $\models_{\mathrm{MR}}$ can apply only if $X \models A$, which parallels the idea of a reduct relative to X. Their second purpose is to determine partial interpretations (Y, X), where the truth of atoms in $X \setminus Y$ is considered unknown, for which an aggregate expression A should be regarded as provably true: $(Y, X) \models_{\mathrm{GZ}} A$ expresses that no unknown atoms from $X \setminus Y$ occur in A, $(Y, X) \models_{\mathrm{LPST}} A$ means that A is satisfied no matter which unknown atoms are taken to be true or false, respectively, and $(Y, X) \models_{\mathrm{MR}} A$ merely requires the satisfaction of A by some subset of the true atoms in Y. Unlike that, the ultimate semantics relies on the conventional satisfaction of aggregate expressions, as introduced in Section 2.2. Similar to $X \models B(r)$, we extend $\models_\Delta$, for $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}\}$, to the body of a rule r by writing $(Y, X) \models_\Delta B(r)$ if $(Y, X) \models_\Delta A$ for each $A \in B(r)$, and indicate that $(Y, X) \not\models_\Delta A$, for some $A \in B(r)$, by $(Y, X) \not\models_\Delta B(r)$.

Example 7
Let us investigate the aggregate expression

$$\mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5 \tag{A_7}$$

along with the interpretation $X = \{y_1, z_1, z_2\}$, where $\alpha(A_7, X) = 6$ and thus $X \models A_7$. The condition for $(Y, X) \models_{\mathrm{GZ}} A_7$ additionally requires $\mathrm{AT}(A_7) \cap Y = \mathrm{AT}(A_7) \cap X = \{y_1, z_1, z_2\}$, so that $Y = X = \{y_1, z_1, z_2\}$ is the only subset of X for which $\models_{\mathrm{GZ}}$ applies. For identifying interpretations $Y \subseteq X$ such that $(Y, X) \models_{\mathrm{LPST}} A_7$, observe that $\alpha(A_7, \{z_1, z_2\}) = 5$ and $\{z_1, z_2\} \not\models A_7$. That is, any subset of X that excludes y_1 can be filled up to the interpretation $Z = \{z_1, z_2\}$ for which A_7 is unsatisfied. In turn, such a Z does not exist for interpretations $\{y_1\} \subseteq Y \subseteq X$, and we have $(Y, X) \models_{\mathrm{LPST}} A_7$ for each subset Y of X that includes y_1. Regarding $(Y, X) \models_{\mathrm{MR}} A_7$, it is sufficient to observe that $\alpha(A_7, \emptyset) = 0$ and $\emptyset \models A_7$, so that $\models_{\mathrm{MR}}$ applies for every subset Y of X. ■

Provided that $Y \subseteq X \subseteq \mathcal{P}$, we have that $(Y, X) \models_{\mathrm{GZ}} A$ implies $(Y, X) \models_{\mathrm{LPST}} A$, and also that $(Y, X) \models_{\mathrm{LPST}} A$ implies $(Y, X) \models_{\mathrm{MR}} A$. Moreover, $(Y, X) \models_\Delta A$ yields $(Z, X) \models_\Delta A$ for interpretations $Y \subseteq Z \subseteq X$ and $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}\}$. These properties carry forward to the body $B(r)$ of a rule r.

Given the specific satisfaction relations, we now formulate *immediate consequence operators* characterizing semantics based on constructions for a program P and interpretations $Y \subseteq \mathcal{P}, X \subseteq \mathcal{P}$:

$$\mathcal{T}^{\Delta}_{P,X}(Y) = \bigcup\nolimits_{r \in P, (Y,X) \models_\Delta B(r)} H(r), \text{ for } \Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}\},$$

$$\mathcal{T}^{\mathrm{DPB}}_{P,X}(Y) = \bigcap\nolimits_{Y \subseteq Z \subseteq X} \left(\bigcup\nolimits_{r \in P, Z \models B(r)} H(r) \right).$$

Each operator $\mathcal{T}^{\Delta}_{P,X}$, for $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}, \mathrm{DPB}\}$, joins the heads of rules such that the underlying satisfaction relation applies to their bodies. This would yield unintended

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

results for (proper) disjunctive rules like $p \vee q \leftarrow \top$, having $\{p\}$ and $\{q\}$ as its standard answer sets (Gelfond and Lifschitz 1991), while any of the operators $\mathcal{T}^{\Delta}_{P,X}$ would produce the non-minimal model $\{p, q\}$. In fact, such immediate consequence operators have been devised with non-disjunctive rules in mind, and in the following we assume that $|H(r)| \leq 1$ for rules r under consideration. For the sake of completeness, we point out that the semantics by Gelfond and Zhang (2019), Liu *et al.* (2010), and Marek and Remmel (2004) additionally allow for aggregate expressions in heads of rules, understood as choice constructs (Simons *et al.* 2002), and Gelfond and Zhang (2019) also handle disjunctive rules, as their original concept builds on a specific reduct rather than an operational characterization.

The operators $\mathcal{T}^{\Delta}_{P,X}$ have in common that $\mathcal{T}^{\Delta}_{P,X}(Y) \subseteq \mathcal{T}^{\Delta}_{P,X}(Z)$ when $Y \subseteq Z \subseteq X$. Considering $\mathcal{T}^{\mathrm{DPB}}_{P,X}$, this is because the intersection over interpretations between Z and X involves fewer elements than with Y, while $(Y, X) \models_{\Delta} B(r)$ implies $(Z, X) \models_{\Delta} B(r)$ for the remaining operators $\mathcal{T}^{\Delta}_{P,X}$ and any rule $r \in P$. Provided that X is a model of P, in which case $\mathcal{T}^{\mathrm{GZ}}_{P,X}(X) = \mathcal{T}^{\mathrm{LPST}}_{P,X}(X) = \mathcal{T}^{\mathrm{MR}}_{P,X}(X) = \mathcal{T}^{\mathrm{DPB}}_{P,X}(X) \subseteq X$, the least fixpoint of $\mathcal{T}^{\Delta}_{P,X}$ is thus guaranteed to exist and can be constructed as follows:

$$\mathcal{T}^{\Delta}_{P,X} \uparrow 0 = \emptyset,$$
$$\mathcal{T}^{\Delta}_{P,X} \uparrow i + 1 = \mathcal{T}^{\Delta}_{P,X}(\mathcal{T}^{\Delta}_{P,X} \uparrow i).$$

That is, starting from the empty interpretation $\emptyset$ for $\mathcal{T}^{\Delta}_{P,X} \uparrow 0$, the iterated application of $\mathcal{T}^{\Delta}_{P,X}$ leads to $\mathcal{T}^{\Delta}_{P,X} \uparrow i + 1 = \mathcal{T}^{\Delta}_{P,X}(\mathcal{T}^{\Delta}_{P,X} \uparrow i) = \mathcal{T}^{\Delta}_{P,X} \uparrow i$ for some $i \geq 0$, and we denote this *least fixpoint* by $\mathcal{T}^{\Delta}_{P,X} \uparrow \infty$. For $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}, \mathrm{DPB}\}$, we call X a Δ-*answer set* of P if X is a model of P such that $\mathcal{T}^{\Delta}_{P,X} \uparrow \infty = X$.

Example 8
Reconsider the program P_2 from Example 4:

$$p \leftarrow \mathrm{SUM}[1 : p] > 0 \qquad (r_4)$$
$$p \leftarrow \mathrm{SUM}[1 : p] < 1. \qquad (r_5)$$

We have that $X = \{p\}$ is the unique model of P_2 (over $\mathcal{P} = \{p\}$). Then, $\alpha(\mathrm{SUM}[1 : p] < 1, X) = 1$ yields $X \not\models B(r_5)$ and $(\emptyset, X) \not\models_{\Delta} B(r_5)$ for $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}\}$. Moreover, $(\emptyset, X) \not\models_{\Delta} B(r_4)$ follows from $\alpha(\mathrm{SUM}[1 : p] > 0, \emptyset) = 0$ and $\emptyset \not\models B(r_4)$. As a consequence, for $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}\}$, we obtain $\mathcal{T}^{\Delta}_{P_2,X} \uparrow \infty = \mathcal{T}^{\Delta}_{P_2,X} \uparrow 0 = \emptyset$, so that X is not a Δ-answer set of P_2.

Unlike that, $\emptyset \models B(r_5)$ and $X \models B(r_4)$ with $H(r_5) = H(r_4) = \{p\}$ lead to $\mathcal{T}^{\mathrm{DPB}}_{P_2,X} \uparrow \infty = \mathcal{T}^{\mathrm{DPB}}_{P_2,X} \uparrow 1 = \mathcal{T}^{\mathrm{DPB}}_{P_2,X}(\emptyset) = \{p\} = X$. Hence, we conclude that X is a DPB-answer set of P_2. ■

Example 9
Recall from Example 5 that $X = \{p, q\}$ is the unique model and FFLP-answer set of P_1:

$$p \leftarrow \mathrm{SUM}[1 : p, -1 : q] \geq 0 \qquad (r_1)$$
$$p \leftarrow \mathrm{SUM}[1 : q] > 0 \qquad (r_2)$$
$$q \leftarrow \mathrm{SUM}[1 : p] > 0. \qquad (r_3)$$

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Considering the empty interpretation $\emptyset$, we have that $\text{AT}(A) \cap X \neq \emptyset$ for each aggregate expression $A \in B(r_1) \cup B(r_2) \cup B(r_3)$. Hence, $\mathcal{T}^{\text{GZ}}_{P_1,X} \uparrow \infty = \mathcal{T}^{\text{GZ}}_{P_1,X} \uparrow 0 = \emptyset$ yields that X is not a GZ-answer set of P_1. Regarding $\mathcal{T}^{\text{LPST}}_{P_1,X}$, observe that $\emptyset \not\models B(r_2)$, $\emptyset \not\models B(r_3)$, and $\{q\} \not\models B(r_1)$ in view of $\alpha(\text{SUM}[1:p, -1:q] \geq 0, \{q\}) = -1$. This means that $(\emptyset, X) \not\models_{\text{LPST}} B(r)$ for all $r \in P_1$, so that $\mathcal{T}^{\text{LPST}}_{P_1,X} \uparrow \infty = \mathcal{T}^{\text{LPST}}_{P_1,X} \uparrow 0 = \emptyset$ disproves X to be an LPST-answer set of P_1.

When we turn to $\mathcal{T}^{\text{MR}}_{P_1,X}$, then $\alpha(\text{SUM}[1:p, -1:q] \geq 0, \emptyset) = \alpha(\text{SUM}[1:p, -1:q] \geq 0, X) = 0$ shows that $(\emptyset, X) \models_{\text{MR}} B(r_1)$ and $\mathcal{T}^{\text{MR}}_{P_1,X} \uparrow 1 = \mathcal{T}^{\text{MR}}_{P_1,X}(\emptyset) = \{p\}$. Given that $(\{p\}, X) \models_{\text{MR}} B(r_3)$, this leads on to $\mathcal{T}^{\text{MR}}_{P_1,X} \uparrow \infty = \mathcal{T}^{\text{MR}}_{P_1,X} \uparrow 2 = \mathcal{T}^{\text{MR}}_{P_1,X}(\{p\}) = \{p, q\} = X$, from which we conclude that X is an MR-answer set of P_1.

For obtaining the least fixpoint of $\mathcal{T}^{\text{DPB}}_{P_1,X}$, consider the rules r_1 and r_2, and observe that $\emptyset \models B(r_1)$, $\{p\} \models B(r_1)$, $\{q\} \models B(r_2)$, and $X \models B(r_1)$ as well as $X \models B(r_2)$. That is, for each interpretation $Z \subseteq X$, the body of some rule with p in the head is satisfied by Z. Hence, we get $\mathcal{T}^{\text{DPB}}_{P_1,X} \uparrow 1 = \mathcal{T}^{\text{DPB}}_{P_1,X}(\emptyset) = \{p\}$, which in view of $\{p\} \models B(r_3)$ and $X \models B(r_3)$ brings us further to $\mathcal{T}^{\text{DPB}}_{P_1,X} \uparrow \infty = \mathcal{T}^{\text{DPB}}_{P_1,X} \uparrow 2 = \mathcal{T}^{\text{DPB}}_{P_1,X}(\{p\}) = \{p, q\} = X$. This means that X is also a DPB-answer set of P_1.

Let us modify the program P_1 to P_1' over $\mathcal{P} = \{p, q, s\}$, in which r_1 and r_3 are replaced by:

$$s \leftarrow \text{SUM}[1:p, -1:q] \geq 0 \tag{r_1'}$$

$$q \leftarrow \text{SUM}[1:s] > 0. \tag{r_3'}$$

Intuitively, rule r_1' requires s true if p is true or q is false, and rule r_3' requires q true if s is true (recall that r_2 requires p true if q is true). One can check that $X' = \{p, q, s\}$ is the unique model as well as an FFLP- and MR-answer set of P_1'. Turning again to $\mathcal{T}^{\text{DPB}}_{P_1',X'}$, we have that $\emptyset \not\models B(r_2)$, $\emptyset \not\models B(r_3')$, $\{q\} \not\models B(r_1')$, $\{q\} \not\models B(r_3')$, and the heads of r_1' and r_2 are disjoint for the modified program P_1'. (Each interpretation $Z' \subseteq X'$ is such that $Z' \models B(r_1')$ or $Z' \models B(r_2)$, yet $H(r_1') \cap H(r_2) = \{s\} \cap \{p\} = \emptyset$.) Given these considerations, we conclude that $\mathcal{T}^{\text{DPB}}_{P_1',X'} \uparrow \infty = \mathcal{T}^{\text{DPB}}_{P_1',X'} \uparrow 0 = \emptyset$, so that P_1' does not have any DPB-answer set. This shows that FFLP-answer sets are not necessarily DPB-answer sets as well. ■

Example 10

In Example 6, we have checked that $X = \{y_1, x_2, z_1, z_2, p\}$ is the unique FFLP-answer set of P_3:

$$x_1 \leftarrow \text{SUM}[1:y_1] < 1 \tag{r_6}$$

$$y_1 \leftarrow \text{SUM}[1:x_1] < 1 \tag{r_7}$$

$$x_2 \leftarrow \text{SUM}[1:y_2] < 1 \tag{r_8}$$

$$y_2 \leftarrow \text{SUM}[1:x_2] < 1 \tag{r_9}$$

$$z_1 \leftarrow \text{SUM}[1:p] > 0 \tag{r_{10}}$$

$$z_2 \leftarrow \text{SUM}[1:p] > 0 \tag{r_{11}}$$

$$p \leftarrow \text{SUM}[1:y_1, 2:y_2, 2:z_1, 3:z_2] \neq 5 \tag{r_{12}}$$

$$\bot \leftarrow \text{SUM}[1:p] < 1. \tag{r_{13}}$$

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Inspecting $\mathcal{T}^{\mathrm{GZ}}_{P_3,X}$, we get $\mathcal{T}^{\mathrm{GZ}}_{P_3,X} \uparrow 1 = \mathcal{T}^{\mathrm{GZ}}_{P_3,X}(\emptyset) = \{y_1, x_2\}$ because $X \models B(r_7)$, $X \models B(r_8)$, and $\mathrm{AT}(\mathrm{SUM}[1 : x_1] < 1) \cap X = \mathrm{AT}(\mathrm{SUM}[1 : y_2] < 1) \cap X = \emptyset$. Given that $X \not\models B(r_6)$, $X \not\models B(r_9)$, $\mathrm{AT}(\mathrm{SUM}[1 : p] > 0) \cap \{y_1, x_2\} = \emptyset \neq \{p\} = \mathrm{AT}(\mathrm{SUM}[1 : p] > 0) \cap X$, and $\mathrm{AT}(\mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5) \cap \{y_1, x_2\} = \{y_1\} \neq \{y_1, z_1, z_2\} = \mathrm{AT}(\mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5) \cap X$, we obtain $\mathcal{T}^{\mathrm{GZ}}_{P_3,X} \uparrow \infty = \mathcal{T}^{\mathrm{GZ}}_{P_3,X} \uparrow 1 = \{y_1, x_2\}$, so that X is not a GZ-answer set of P_3.

For $\Delta \in \{\mathrm{LPST}, \mathrm{DPB}\}$, we also have that $\mathcal{T}^{\Delta}_{P_3,X} \uparrow 1 = \mathcal{T}^{\Delta}_{P_3,X}(\emptyset) = \{y_1, x_2\}$, as $Z \models B(r_7)$ and $Z \models B(r_8)$ for each $Z \subseteq X$, while the interpretation $\{z_1, z_2\} \subseteq X$ is such that $\{z_1, z_2\} \not\models \mathrm{SUM}[1 : p] > 0$ and $\{z_1, z_2\} \not\models \mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5$. However, $Z \models \mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5$ is the case for every $\{y_1, x_2\} \subseteq Z \subseteq X$, which leads to $\mathcal{T}^{\Delta}_{P_3,X} \uparrow 2 = \mathcal{T}^{\Delta}_{P_3,X}(\{y_1, x_2\}) = \{y_1, x_2, p\}$. Then, $Z \models \mathrm{SUM}[1 : p] > 0$, for any interpretation $\{y_1, x_2, p\} \subseteq Z$, yields $\mathcal{T}^{\Delta}_{P_3,X} \uparrow \infty = \mathcal{T}^{\Delta}_{P_3,X} \uparrow 3 = \mathcal{T}^{\Delta}_{P_3,X}(\{y_1, x_2, p\}) = \{y_1, x_2, z_1, z_2, p\} = X$, which shows that X is an LPST- and a DPB-answer set of P_3.

Regarding $\mathcal{T}^{\mathrm{MR}}_{P_3,X}$, in view of $X \models \mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5$ and $\alpha(\mathrm{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5, \emptyset) = 0$, we get $\mathcal{T}^{\mathrm{MR}}_{P_3,X} \uparrow 1 = \mathcal{T}^{\mathrm{MR}}_{P_3,X}(\emptyset) = \{y_1, x_2, p\}$ in the first step, and then $\mathcal{T}^{\mathrm{MR}}_{P_3,X} \uparrow \infty = \mathcal{T}^{\mathrm{MR}}_{P_3,X} \uparrow 2 = \mathcal{T}^{\mathrm{MR}}_{P_3,X}(\{y_1, x_2, p\}) = \{y_1, x_2, z_1, z_2, p\} = X$. On the one hand, this means that X is an MR-answer set of P_3. On the other hand, the interpretations $X_2 = \{x_1, y_2, z_1, z_2, p\}$ and $X_4 = \{y_1, y_2, z_1, z_2, p\}$ (according to the naming scheme of Example 6) are obtained as additional MR-answer sets of P_3 that are not backed up by any of the other semantics. In fact, counting on the existence of some interpretation $Z \subseteq \mathcal{T}^{\mathrm{MR}}_{P,X} \uparrow i$ such that $Z \models A$, for an aggregate expression A belonging to the body of a rule $r \in P$, to proceed with (potentially) adding $H(r)$ to $\mathcal{T}^{\mathrm{MR}}_{P,X} \uparrow i$ disregards the satisfaction of A by $\mathcal{T}^{\mathrm{MR}}_{P,X} \uparrow i$ itself as well as interpretations extending it. For this reason, Vanbesien *et al.* (2021) classify $\mathcal{T}^{\mathrm{MR}}_{P,X}$ as not well-behaved, while $\mathcal{T}^{\Delta}_{P,X}$ is well-behaved for $\Delta \in \{\mathrm{GZ}, \mathrm{LPST}, \mathrm{DPB}\}$. ■

The programs considered in Example 8–10 did not have GZ-answer sets because of the strong condition $\mathrm{AT}(A) \cap Y = \mathrm{AT}(A) \cap X$ for $(Y, X) \models_{\mathrm{GZ}} A$. Its intention is to circumvent so-called vicious circles (Gelfond and Zhang 2019), where the satisfaction of an aggregate expression A has some influence on the outcome of the aggregation function in A, which happens in Example 8–10, for example, with the rule $p \leftarrow \mathrm{SUM}[1 : p, -1 : q] \geq 0$ to conclude p from an aggregate expression mentioning p.

To make circularity more formal, for a program P, let $\mathcal{G}_P = (\mathcal{P}, \{(p, q) \mid r \in P, p \in H(r), A \in B(r), q \in \mathrm{AT}(A)\})$ be the (directed) *atom dependency graph* of P. If $\mathcal{G}_P$ is acyclic (and the rules in P are non-disjunctive), one can show that there is at most one Δ-answer set of P, for $\Delta \in \{\mathrm{FFLP}, \mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}, \mathrm{DPB}\}$, and that all aggregate semantics under consideration coincide. Provided that P does not include constraints, acyclicity of $\mathcal{G}_P$ is sufficient for the existence of a GZ-answer set of P. However, such acyclicity is not a necessary condition, for example, GZ-answer sets match standard answer sets (Gelfond and Lifschitz 1988) when aggregate expressions of the form $\mathrm{SUM}[1 : p] > 0$ or $\mathrm{SUM}[1 : p] < 1$ replace (negated) propositional atoms p, no matter whether $\mathcal{G}_P$ is acyclic or not, and this correspondence applies to all but the DPB-answer set semantics (cf. Example 8). In fact, vicious circles are not identified syntactically, but rather the construction of a GZ-answer set X of P by $\mathcal{T}^{\mathrm{GZ}}_{P,X}$ witnesses the non-circular provability of all atoms in X.

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

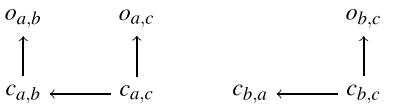


Fig. 4. Dependency graph of program P_4 from Example 11.

Example 11

Consider the program P_4, which constitutes a propositional version of the company controls problem encoding introduced in Section 1 for the instance given in Figure 2:

$$o_{a,b} \leftarrow \top \tag{r_{14}}$$

$$o_{a,c} \leftarrow \top \tag{r_{15}}$$

$$o_{b,c} \leftarrow \top \tag{r_{16}}$$

$$c_{a,b} \leftarrow \text{SUM}[80 : o_{a,b}] > 50 \tag{r_{17}}$$

$$c_{a,c} \leftarrow \text{SUM}[30 : o_{a,c}, 30 : c_{a,b}] > 50 \tag{r_{18}}$$

$$c_{b,c} \leftarrow \text{SUM}[30 : o_{b,c}, 30 : c_{b,a}] > 50. \tag{r_{19}}$$

The atom dependency graph $\mathcal{G}_{P_4} = (\{o_{a,b}, o_{a,c}, o_{b,c}, c_{a,b}, c_{a,c}, c_{b,a}, c_{b,c}\}, \{(c_{a,b}, o_{a,b}), (c_{a,c}, o_{a,c}), (c_{a,c}, c_{a,b}), (c_{b,c}, o_{b,c}), (c_{b,c}, c_{b,a})\})$, shown in Figure 4, is acyclic, and the interpretation $X = \{o_{a,b}, o_{a,c}, o_{b,c}, c_{a,b}, c_{a,c}\}$ is a model of P_4. Let us construct $\mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow \infty$ to check that X is a GZ-answer set of P_4. In the first step, $(\emptyset, X) \models_{\text{GZ}} B(r)$ applies for the facts $r \in \{r_{14}, r_{15}, r_{16}\}$, so that $\mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow 1 = \mathcal{T}^{\text{GZ}}_{P_4,X}(\emptyset) = \{o_{a,b}, o_{a,c}, o_{b,c}\}$. Then, we have that $\text{AT}(\text{SUM}[80 : o_{a,b}] > 50) \cap \mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow 1 = \text{AT}(\text{SUM}[80 : o_{a,b}] > 50) \cap X = \{o_{a,b}\}$, and $X \models B(r_{17})$ in view of $\alpha(\text{SUM}[80 : o_{a,b}] > 50, X) = 80$. We thus get $\mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow 2 = \mathcal{T}^{\text{GZ}}_{P_4,X}(\{o_{a,b}, o_{a,c}, o_{b,c}\}) = \{o_{a,b}, o_{a,c}, o_{b,c}, c_{a,b}\}$, and $\text{AT}(\text{SUM}[30 : o_{a,c}, 30 : c_{a,b}] > 50) \cap \mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow 2 = \text{AT}(\text{SUM}[30 : o_{a,c}, 30 : c_{a,b}] > 50) \cap X = \{o_{a,c}, c_{a,b}\}$ along with $\alpha(\text{SUM}[30 : o_{a,c}, 30 : c_{a,b}] > 50, X) = 60$ and $X \models B(r_{18})$ lead to $\mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow 3 = \mathcal{T}^{\text{GZ}}_{P_4,X}(\{o_{a,b}, o_{a,c}, o_{b,c}, c_{a,b}\}) = \{o_{a,b}, o_{a,c}, o_{b,c}, c_{a,b}, c_{a,c}\} = X$. Given that $X \not\models B(r_{19})$, the least fixpoint $\mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow \infty = \mathcal{T}^{\text{GZ}}_{P_4,X} \uparrow 3 = X$ yields that X is a GZ-answer set of P_4.

However, note that X is no longer a GZ-answer set of P'_4, obtained by replacing r_{17} with

$$c_{a,b} \leftarrow \text{SUM}[80 : o_{a,b}, w : c_{a,c}] > 50 \tag{r'_{17}}$$

for some weight $w \geq 0$. While we still obtain $\mathcal{T}^{\text{GZ}}_{P'_4,X} \uparrow 1 = \mathcal{T}^{\text{GZ}}_{P'_4,X}(\emptyset) = \{o_{a,b}, o_{a,c}, o_{b,c}\}$, we end up with $\mathcal{T}^{\text{GZ}}_{P'_4,X} \uparrow \infty = \mathcal{T}^{\text{GZ}}_{P'_4,X} \uparrow 1$ because $\text{AT}(\text{SUM}[80 : o_{a,b}, w : c_{a,c}] > 50) \cap \mathcal{T}^{\text{GZ}}_{P'_4,X} \uparrow 1 = \{o_{a,b}\} \neq \{o_{a,b}, c_{a,c}\} = \text{AT}(\text{SUM}[80 : o_{a,b}, w : c_{a,c}] > 50) \cap X$ and thus $(\mathcal{T}^{\text{GZ}}_{P'_4,X} \uparrow 1, X) \not\models_{\text{GZ}} B(r'_{17})$. That is, X is rejected as a GZ-answer set of P'_4 in view of the circularity between $c_{a,b}$ and $c_{a,c}$, which are involved in aggregate expressions in the bodies of r'_{17} and r_{18}. When considering standard answer sets of programs without (sophisticated) aggregates (Gelfond and Lifschitz 1988; 1991), circular rules alone do not yield provability of atoms in their heads. Unlike that, not all propositional atoms subject to an aggregation function may be needed to satisfy a respective aggregate expression, so that rejecting any circularity is a very strong condition for programs with aggregates. In fact, without going into the details, we have that X is the unique Δ-answer set of both

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Table 1. *Relationships between semantics for non-disjunctive programs with (monotone, anti-monotone, convex, or arbitrary) aggregate expressions, showing when Monotonicity conditions guaranteeing answer sets according to the semantics in rows to be answer sets according to the semantics in columns*

	GZ	LPST	DPB	FFLP	MR
GZ	–	arbitrary	arbitrary	arbitrary	arbitrary
LPST	none	–	arbitrary	arbitrary	arbitrary
DPB	none	(anti-)monotone	–	(anti-)monotone	(anti-)monotone
FFLP	none	convex	convex	–	arbitrary
MR	none	convex	convex	convex	–

P_4 and P_4' for $\Delta \in \{\mathrm{FFLP}, \mathrm{LPST}, \mathrm{MR}, \mathrm{DPB}\}$, that is, all aggregate semantics other than GZ. ■

3.3 Semantic relationships

As mentioned above, the satisfaction relation $\models_{\mathrm{GZ}}$ is a subset of $\models_{\mathrm{LPST}}$, which is in turn a subset of $\models_{\mathrm{MR}}$. That is, given a model X of a (non-disjunctive) program P, we immediately get $\mathcal{T}^{\mathrm{LPST}}_{P,X} \uparrow \infty = X$ from $\mathcal{T}^{\mathrm{GZ}}_{P,X} \uparrow \infty = X$, and $\mathcal{T}^{\mathrm{MR}}_{P,X} \uparrow \infty = X$ from $\mathcal{T}^{\mathrm{LPST}}_{P,X} \uparrow \infty = X$, which means that a GZ-answer set X of P is an LPST-answer set as well, and an LPST-answer set X of P also an MR-answer set. The converse relationships do not apply in general, yet may come into effect for programs restricted to monotone, anti-monotone, or convex aggregate expressions only. Moreover, the question arises how DPB-answer sets determined by the operator $\mathcal{T}^{\mathrm{DPB}}_{P,X}$ and model-based FFLP-answer sets relate to the other aggregate semantics. A part of these relationships have already been studied by Ferraris (2011) and Liu *et al.* (2010), and in the following we give a complete account for the aggregate semantics under consideration, assuming programs in question to be non-disjunctive.

Figure 5 and Table 1 visualize the relationships between different aggregate semantics, where arcs without label express that answer sets under one semantics are preserved by another semantics for programs with arbitrary, that is, possibly non-convex, aggregate expressions. The restriction of such a correspondence to programs with convex, monotone, or anti-monotone aggregate expressions only is indicated by the arc label $\pm$, $+$, or $-$, respectively. Transitive relationships, for example, the fact that each GZ-answer set is an MR-answer set as well, are not stated by explicit arcs for better readability. A summary of the logic programs used to illustrate the similarities and differences between the aggregate semantics in Sections 3.1 and 3.2 is given in Table 2.

The most restrictive semantics is obtained with GZ-answer sets, in the sense that the fewest models X of a program P pass the construction by means of $\mathcal{T}^{\mathrm{GZ}}_{P,X}$. As $\mathcal{T}^{\mathrm{LPST}}_{P,X}$ relaxes the conditions for concluding head atoms, LPST-answer sets are guaranteed to include GZ-answer sets, as indicated by an arc without label in Figure 5. The observation that $(Y, X) \models_{\mathrm{LPST}} B(r)$, for some rule $r \in P$ and $Y \subseteq X$, implies $Z \models B(r)$ for every interpretation $Y \subseteq Z \subseteq X$ yields that $H(r) \subseteq \mathcal{T}^{\mathrm{DPB}}_{P,X}(Y)$, so that each LPST-answer set X of P is a DPB-answer set as well. Moreover, when $Y \subset X$ for an LPST-answer set

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

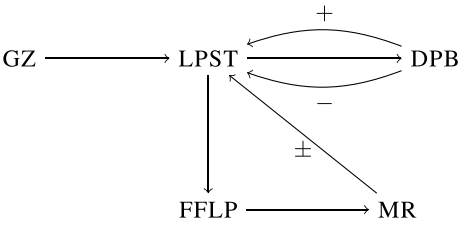


Fig. 5. Relationships between aggregate semantics, where an arc indicates that answer sets carry forward from the origin to the target semantics for non-disjunctive programs with monotone (arc label +), anti-monotone (arc label −), convex (arc label ±), or arbitrary (no arc label) aggregate expressions, respectively.

Table 2. *Summary of examples showing differences between aggregate semantics, where* $X = \{o_{a,b}, o_{a,c}, o_{b,c}, c_{a,b}, c_{a,c}\}$

Program	FFLP	GZ	LPST	MR	DPB
P_1 Example 1	$\{p, q\}$ Example 5	– Example 9	– Example 9	$\{p, q\}$ Example 9	$\{p, q\}$ Example 9
P_1' Example 9	$\{p, q, s\}$ Example 9	– Example 9	– Example 9	$\{p, q, s\}$ Example 9	– Example 9
P_2 Example 4	– Example 4	– Example 8	– Example 8	– Example 8	$\{p\}$ Example 8
P_3 Example 6	$\{y_1, x_2, z_1, z_2, p\}$ Example 6	– Example 10	$\{y_1, x_2, z_1, z_2, p\}$ Example 10	$\{x_1, y_2, z_1, z_2, p\}$, $\{y_1, y_2, z_1, z_2, p\}$, $\{y_1, x_2, z_1, z_2, p\}$ Example 10	$\{y_1, x_2, z_1, z_2, p\}$ Example 10
P_4 Example 11	X Example 11	X Example 11	X Example 11	X Example 11	X Example 11
P_4' Example 11	X Example 11	– Example 11	X Example 11	X Example 11	X Example 11

X of P, the fact that $\mathcal{T}_{P,X}^{\mathrm{LPST}}(Y) \not\subseteq Y$ shows that Y is not a model of the reduct P^X, which in turn witnesses that X is a $\subseteq$-minimal model of P^X and thus an FFLP-answer set of P. The last general relationship that applies for programs with arbitrary aggregate expressions expresses that each FFLP-answer set X of P is also an MR-answer set, given that $Y \not\models P^X$, for any interpretation $Y \subset X$, implies that $\mathcal{T}_{P,X}^{\mathrm{MR}}(Y) \not\subseteq Y$. As discussed in Examples 8 and 9, FFLP- and DPB-answer sets can be mutually distinct, so that no (transitive) general relationship is obtained between these two aggregate semantics.

Considering relationships for convex, monotone, or anti-monotone aggregate expressions, an MR-answer set X of P is also an LPST-answer set when the program P includes convex aggregate expressions only. This follows from the observation that, for a convex aggregate expression A and each interpretation $Y \subseteq X$, the conditions $X \models A$

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

and $Z \models A$, for some $Z \subseteq Y$, of $(Y, X) \models_{\text{MR}} A$ yield the condition of $(Y, X) \models_{\text{LPST}} A$ that $Z \models A$ for every $Y \subseteq Z \subseteq X$. The general relationship between LPST- and FFLP-answer sets then further implies that X is an FFLP-answer set of P as well, so that FFLP-, LPST-, and MR-answer sets coincide for programs with convex aggregate expressions only. For establishing similar correspondences to DPB-answer sets, as Example 8 shows, programs with convex or both monotone and anti-monotone aggregate expressions, respectively, are still too general. However, when the aggregate expressions in a program P are monotone or anti-monotone only, for interpretations $Y \subseteq X$, the intersection over all $Y \subseteq Z \subseteq X$ taken by $\mathcal{T}^{\text{DPB}}_{P,X}(Y)$ collapses to checking whether rule bodies are satisfied by Y or X, respectively, while satisfaction by other interpretations Z is a consequence of (anti-)monotonicity. Hence, we have that $\mathcal{T}^{\text{LPST}}_{P,X}(Y) = \mathcal{T}^{\text{DPB}}_{P,X}(Y)$, which means that each DPB-answer set X of P is also an LPST-answer set in case of monotone or anti-monotone aggregate expressions only. By (transitive) general relationships, this correspondence carries forward to FFLP- and MR-answer sets.

Interestingly, GZ-answer sets do not correspond to the other semantics even for programs whose aggregate expressions are restricted to be monotone. For instance, the program P consisting of the rule $p \leftarrow \text{SUM}[1 : p] \geq 0$, which includes the monotone as well as anti-monotone aggregate expression $\text{SUM}[1 : p] \geq 0$, has $\{p\}$ as an answer set under all aggregate semantics but GZ, where the latter is due to $\mathcal{T}^{\text{GZ}}_{P,\{p\}} \uparrow \infty = \mathcal{T}^{\text{GZ}}_{P,\{p\}} \uparrow 0 = \emptyset$ in view of $\text{AT}(\text{SUM}[1 : p] \geq 0) \cap \{p\} \neq \emptyset$. As $\{p\}$ is the unique model of P, this means that P does not have any GZ-answer set, even though P includes only one aggregate expression that is both monotone and anti-monotone. Such a behavior is justified by Gelfond and Zhang (2019) by a more uniform treatment of recursive symbolic rules like the following:

```
p :- #sum{1 : p} >= 0.
p :- #sum{1 : p} = S, S >= 0.
```

In fact, a program consisting of the second rule above, would be instantiated as

$$p \leftarrow \text{SUM}[1 : p] = 0$$
$$p \leftarrow \text{SUM}[1 : p] = 1$$

which has no answer sets according to all semantics discussed in this paper.

4 Computational complexity of aggregates

The expressiveness of programs with aggregate expressions can be assessed in terms of the computational complexity (Garey and Johnson 1979) of specific reasoning tasks. To this end, we investigate the following two decision problems for $\Delta \in \{\text{FFLP}, \text{GZ}, \text{LPST}, \text{MR}, \text{DPB}\}$:

Check: For a program P and an interpretation $X \subseteq \mathcal{P}$, decide whether X is a Δ-answer set of P.

Exist: For a program P, decide whether there is some Δ-answer set of P.

The complexity of both tasks is well-understood (Dantsin *et al.* 2001) for programs P in which rule bodies consist of (negated) propositional atoms p, corresponding to aggregate expressions of the form $\text{SUM}[1 : p] > 0$ or $\text{SUM}[1 : p] < 1$, under standard answer set semantics (Gelfond and Lifschitz 1988; 1991). Here the complexity of reasoning tasks

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Table 3. *Computational complexity of the Exist and Check reasoning tasks for different aggregate semantics and the monotonicity of aggregate expressions included in non-disjunctive programs, where cases in which elevated complexity relies on the* NP*-hardness of deciding whether aggregate expressions under consideration are satisfiable are indicated by lower complexity classes obtained with tractable satisfiability checks in parentheses*

	Monotonicity	GZ	MR	LPST	FFLP	DPB
Check	Monotone	P	P	P	P	P
	Anti-monotone	P	P	P	P	P
	Convex	P	P	P	P	coNP
	Arbitrary	P	NP (P)	coNP (P)	coNP	coNP
Exist	Monotone	P	P	P	P	P
	Anti-monotone	NP	NP	NP	NP	NP
	Convex	NP	NP	NP	NP	Σ_2^{P}
	Arbitrary	NP	NP	Σ_2^{P} (NP)	Σ_2^{P}	Σ_2^{P}

depends on whether P is non-disjunctive or not, Check is P-complete (i.e., tractable) in the non-disjunctive case, and coNP-complete otherwise. The Exist task, where no interpretation $X \subseteq \mathcal{P}$ is fixed, is NP-complete when P is non-disjunctive, and in general Σ_2^{P}-complete in the presence of (proper) disjunctive rules. The latter means that two orthogonal combinatorial problems are interleaved, one about determining candidate models X among an exponential number of interpretations, and the other concerning the check that no $Y \subset X$ is a model of the reduct relative to X (again, the search space for Y is exponential).

Given that (proper) disjunctive rules lead to elevated computational complexity already for the simple forms of aggregate expressions resembling (negated) propositional atoms, we again assume programs to be non-disjunctive in the following. This allows us to study the complexity of the Check and Exist tasks for Δ-answer sets relative to $\Delta \in \{\mathrm{FFLP}, \mathrm{GZ}, \mathrm{LPST}, \mathrm{MR}, \mathrm{DPB}\}$ and the monotonicity of aggregate expressions, where the obtained complexity classes are summarized in Table 3. Except for the complexity class P, for which we indicate membership but not necessarily hardness, our considerations address completeness, that is, membership along with hardness for some complexity class beyond P. In Section 4.1, we investigate the computational complexity of the Check task in detail, and Section 4.2 provides a closer study of the Exist task.

4.1 Answer set checking

Starting with Check for (non-disjunctive) programs P including monotone or anti-monotone aggregate expressions only, we have that the task stays in P for all aggregate semantics. An intuitive explanation is that all but the GZ-answer set semantics coincide for such programs and that a construction, for example, by means of $\mathcal{T}_{P,X}^{\mathrm{DPB}}(Y)$, for some $Y \subseteq X$, can focus on concluding head atoms of rules $r \in P$ such that $Y \models B(r)$ or $X \models B(r)$, respectively. For $\mathcal{T}_{P,X}^{\mathrm{GZ}}(Y)$, also the additional condition $\mathrm{AT}(A) \cap Y = \mathrm{AT}(A) \cap X$ needs to be checked for each aggregate expression $A \in B(r)$. Regardless of the particular semantics, all relevant checks can be accomplished in polynomial time: $X \models P$ to

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

make sure that X is a model of P, $Y \models B(r)$ or $X \models B(r)$ for rules $r \in P$, and possibly $\text{AT}(A) \cap Y = \text{AT}(A) \cap X$ for aggregate expressions $A \in B(r)$. The strong condition $\text{AT}(A) \cap Y = \text{AT}(A) \cap X$ for $(Y, X) \models_{\text{GZ}} A$ also circumvents elevated complexity of Check under GZ-answer set semantics when turning to programs with convex or arbitrary, that is, possibly non-convex, aggregate expressions.

In case of convex aggregate expressions, which include monotone as well as anti-monotone aggregate expressions, the Check task remains tractable for $\Delta \in \{\text{FFLP}, \text{LPST}, \text{MR}\}$, given that the three semantics coincide, and for an aggregate expression A, checking $Y \models A$ and $X \models A$ is sufficient to conclude that $(Y, X) \models_{\text{LPST}} A$ as well as $(Y, X) \models_{\text{MR}} A$. Unlike that, Check becomes more complex, that is, coNP-complete, for DPB-answer sets of programs with convex aggregate expressions, where Denecker *et al.* (2004) show coNP-hardness by reduction from deciding whether a propositional formula in disjunctive normal form is a tautology. Notably, the logic program given in this reduction does not refer to (sophisticated) aggregates, and the simultaneous availability of monotone and anti-monotone aggregate expressions of the form $\text{SUM}[1 : p] > 0$ or $\text{SUM}[1 : p] < 1$ to express (negated) propositional atoms p is already sufficient. This complexity does also not increase any further in the presence of non-convex aggregate expressions, as for disproving X to be a DPB-answer set of P, it is sufficient to check that $X \not\models P$, or otherwise to determine a collection of (not necessarily distinct) interpretations $Y_1 \subseteq X, \ldots, Y_{|X|} \subseteq X$ such that $Z \subseteq Y$ and $Z \subset X$ hold for $Y = Y_1 \cap \cdots \cap Y_{|X|}$ and

$$Z = \left(\bigcup_{r \in P, Y_1 \models B(r)} H(r)\right) \cap \cdots \cap \left(\bigcup_{r \in P, Y_{|X|} \models B(r)} H(r)\right) \cap X.$$

In the latter case, we have that $\mathcal{T}^{\text{DPB}}_{P,X}(Y) \subseteq Y$ and $\mathcal{T}^{\text{DPB}}_{P,X}(Y) \subset X$ for $Y = Y_1 \cap \cdots \cap Y_{|X|} \subseteq X$, which in turn implies that $\mathcal{T}^{\text{DPB}}_{P,X} \uparrow \infty \subseteq \mathcal{T}^{\text{DPB}}_{P,X}(Y) \subset X$. Moreover, when $\mathcal{T}^{\text{DPB}}_{P,X} \uparrow \infty \subset X$, note that at most $|X \setminus \mathcal{T}^{\text{DPB}}_{P,X} \uparrow \infty| \leq |X|$ many interpretations $\mathcal{T}^{\text{DPB}}_{P,X} \uparrow \infty \subseteq Y \subseteq X$ are needed to show that none of the atoms in $X \setminus \mathcal{T}^{\text{DPB}}_{P,X} \uparrow \infty$ belongs to $\mathcal{T}^{\text{DPB}}_{P,X}(\mathcal{T}^{\text{DPB}}_{P,X} \uparrow \infty)$, so that Check under DPB-answer set semantics stays in coNP for programs with arbitrary aggregate expressions.

For $\Delta \in \{\text{FFLP}, \text{LPST}, \text{MR}\}$, arbitrary aggregate expressions make a difference regarding the complexity of the Check task. Concerning FFLP-answer sets, membership in coNP is immediate as checking that $X \not\models P$ or $Y \models P^X$, for some $Y \subset X$, is tractable. The coNP-hardness can be established by a reduction from disjunctive programs to programs with nested implications due to Ferraris (2011), which replaces a disjunctive rule like $p \vee q \leftarrow \top$ by two rules $p \leftarrow \underline{(p \leftarrow q)}$ and $q \leftarrow \underline{(q \leftarrow p)}$. (Note that rule bodies with nested implications are underlined to avoid any ambiguity with rules.) These nested implications are satisfied by the empty interpretation $\emptyset$, become unsatisfied by $\{q\}$ or $\{p\}$, respectively, and are satisfied by the remaining interpretations, in particular, the extended interpretation $\{p, q\}$. Such satisfaction can in turn be captured in terms of non-convex aggregate expressions like $\text{SUM}[1 : p, -1 : q] \geq 0$ and $\text{SUM}[-1 : p, 1 : q] \geq 0$ for $\underline{(p \leftarrow q)}$ or $\underline{(q \leftarrow p)}$, respectively, and various other representations, for example, $\text{MAX}[1 : p, -1 : q] \neq -1$ or $\text{AVG}[0 : x, -1 : p, 1 : q] \geq 0$, where $x \leftarrow \top$ is used as an auxiliary fact, yield the same effect. This shows that Check under FFLP-answer set semantics is coNP-complete for programs including non-convex aggregate expressions (Alviano and Faber 2013).

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Turning to LPST-answer sets, an interpretation $Y \subset X$ disproves a model X of P to be an LPST-answer set of P, if for each rule $r \in P$ with $H(r) \not\subseteq Y$, we find an aggregate expression $A \in B(r)$ along with some $Y \subseteq Z \subseteq X$ such that $Z \not\models A$. As the problem of checking the existence of such a counterexample Y belongs to NP, the complementary Check task for LPST-answer sets is a member of coNP. The coNP-hardness though depends on the complexity of deciding whether $(Y, X) \models_{\text{LPST}} A$ for an aggregate expression A, which is intractable if and only if deciding the satisfiability of the complement of A, obtainable by inverting the comparison operator of A, for example, $\text{AGG}[w_1 : p_1, \ldots, w_n : p_n] = w_0$ when A is of the form $\text{AGG}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$, is NP-complete. As discussed in Section 2.2, such elevated complexity applies to $\text{AGG} \in \{\text{SUM}, \text{TIMES}, \text{AVG}\}$ in combination with the (complementary) comparison operator $=$. Hence, Check for LPST-answer sets is only coNP-complete when P includes aggregate expressions $\text{AGG}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$ with $\text{AGG} \in \{\text{SUM}, \text{TIMES}, \text{AVG}\}$, while it drops to P otherwise, even in the presence of other forms of non-convex aggregate expressions like $\text{AVG}[w_1 : p_1, \ldots, w_n : p_n] \geq w_0$ or $\text{MIN}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$.

Example 12
Consider the program P_5 as follows:

$$x_1 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{20}}$$

$$x_2 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{21}}$$

$$x_3 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{22}}$$

$$p \leftarrow \text{SUM}[2 : x_1, 2 : x_2, 3 : x_3] \neq 5 \tag{r_{23}}$$

For deciding whether the interpretation $X = \{x_1, x_2, x_3, p\}$ is an LPST-answer set of P_5, starting from $\mathcal{T}^{\text{LPST}}_{P_5,X} \uparrow 0 = \emptyset$, we need to check the condition of $(\emptyset, X) \models_{\text{LPST}} B(r_{23})$ that $Z \models \text{SUM}[2 : x_1, 2 : x_2, 3 : x_3] \neq 5$ for every $Z \subseteq X$. This condition fails if and only if the complementary aggregate expression $A = \text{SUM}[2 : x_1, 2 : x_2, 3 : x_3] = 5$ is satisfiable, that is, when $\alpha(A, Z) = 5$ for some $Z \subseteq X$. As $Z_1 = \{x_1, x_3\}$ and $Z_2 = \{x_2, x_3\}$ are such that $\alpha(A, Z_1) = \alpha(A, Z_2) = 5$, the complementary aggregate expression A is satisfiable, so that $(\emptyset, X) \not\models_{\text{LPST}} B(r_{23})$ and $\mathcal{T}^{\text{LPST}}_{P_5,X} \uparrow \infty = \mathcal{T}^{\text{LPST}}_{P_5,X} \uparrow 0 = \emptyset$ disproves X to be an LPST-answer set of P_5.

When we replace the rule r_{23} by

$$p \leftarrow \text{SUM}[2 : x_1, 2 : x_2, 3 : x_3] \neq 6 \tag{r'_{23}}$$

to obtain P'_5, there is no interpretation $Z \subseteq X$ such that $\alpha(\text{SUM}[2 : x_1, 2 : x_2, 3 : x_3] = 6, Z) = 6$. In turn, we have that $Z \models \text{SUM}[2 : x_1, 2 : x_2, 3 : x_3] \neq 6$ for every $Z \subseteq X$, which means that the condition of $(\emptyset, X) \models_{\text{LPST}} B(r'_{23})$ applies and $\mathcal{T}^{\text{LPST}}_{P'_5,X} \uparrow 1 = \mathcal{T}^{\text{LPST}}_{P'_5,X}(\emptyset) = \{p\}$. Given that $\alpha(\text{SUM}[1 : p] > 0, Z) = 1$ for all interpretations $\{p\} \subseteq Z$, we get $(\{p\}, X) \models_{\text{LPST}} B(r)$ for every $r \in P'_5$, which leads to $\mathcal{T}^{\text{LPST}}_{P'_5,X} \uparrow \infty = \mathcal{T}^{\text{LPST}}_{P'_5,X} \uparrow 2 = \mathcal{T}^{\text{LPST}}_{P'_5,X}(\{p\}) = \{x_1, x_2, x_3, p\} = X$. This shows that X is an LPST-answer set of P'_5.

The programs P_5 and P'_5 illustrate that the Check task for LPST-answer sets can be used to decide whether an instance of the NP-complete *subset sum* problem is (un)satisfiable, as programs following the same scheme along with an interpretation consisting of all atoms capture the complement of *subset sum* for arbitrary multisets of weights and bounds. Moreover, Section 2.2 discusses reductions of *subset sum* and the

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

likewise NP-complete *subset product* problem to the satisfiability of aggregate expressions with AVG or TIMES, respectively, so that their complementary aggregate expressions with $\neq$ as comparison operator also make the Check task coNP-complete. ■

For a program P with arbitrary aggregate expressions and a given model X of P, the Check task of deciding whether X is an MR-answer set of P can be accomplished by guessing, for each aggregate expression $A \in \bigcup_{r \in P^X} B(r)$, an interpretation $Z_A \subseteq X$ such that $Z_A \models A$. Given that $X \models A$, such an interpretation Z_A exists for every A under consideration and can be used to witness that $(Y, X) \models_{\mathrm{MR}} A$ for interpretations $Z_A \subseteq Y \subseteq X$. Checking whether $Z_A \subseteq Y$ instead of taking the condition of $(Y, X) \models_{\mathrm{MR}} A$ as such, in order to approximate $\mathcal{T}^{\mathrm{MR}}_{P,X}(Y)$ for some $Y \subseteq X$, yields a tractable immediate consequence operator, whose least fixpoint matches $\mathcal{T}^{\mathrm{MR}}_{P,X} \uparrow \infty$ for well-chosen interpretations Z_A. This shows that the Check task for MR-answer sets stays in NP also in the presence of non-convex aggregate expressions.

For establishing NP-hardness, the complexity of checking the existence of some $Z \subset \mathrm{AT}(A) \cap X$ such that $Z \models A$ for an aggregate expression A with $X \models A$ matters, as the construction of $\mathcal{T}^{\mathrm{MR}}_{P,X} \uparrow \infty$ gets tractable and the complexity of Check drops to P when the satisfiability of aggregate expressions (by some smaller interpretation) can be determined in polynomial time. As the discussion in Section 2.2 shows, deciding satisfiability is NP-hard only for aggregate expressions with SUM, TIMES, or AVG along with the comparison operator $=$, so that any other (non-convex) aggregate expressions do not make the Check task NP-complete. However, for an aggregate expression A of the form $\mathrm{TIMES}[w_1 : p_1, \ldots, w_n : p_n] = w_0$ such that $w_0 = 0$, only singletons $Z = \{p\}$ with $\alpha(A, Z) = 0$ are relevant as subsets $Z \subset \mathrm{AT}(A) \cap X$ such that $Z \models A$. Otherwise, if $w_0 \neq 0$, we have to guarantee that $\alpha(A, X \setminus Z) = 1$, so that a linear collection of ($\subseteq$-minimal) subsets $Z \subset \mathrm{AT}(A) \cap X$ with $Z \models A$ is obtained by excluding atoms $p \in X$ such that $\alpha(A, \{p\}) = 1$ together with a maximum even number of atoms $p \in X$ such that $\alpha(A, \{p\}) = -1$. As there are at most $|\mathrm{AT}(A) \cap X|$ relevant $Z \subset \mathrm{AT}(A) \cap X$ in either case, aggregate expressions with TIMES do not yield NP-completeness of the Check task for MR-answer sets. The next example illustrates that this is different for aggregate expressions with SUM or AVG along with the comparison operator $=$.

Example 13

Let us investigate the model $X = \{x_1, x_2, x_3, p\}$ of the following program P_6:

$$x_1 \leftarrow \top \qquad (r_{24})$$

$$x_2 \leftarrow \top \qquad (r_{25})$$

$$x_3 \leftarrow \top \qquad (r_{26})$$

$$p \leftarrow \mathrm{SUM}[2 : x_1, 2 : x_2, 3 : x_3, -2 : p] = 5 \qquad (r_{27})$$

In order to decide whether X is an MR-answer set of P_6, we need to construct $\mathcal{T}^{\mathrm{MR}}_{P_6,X} \uparrow \infty$, where $\mathcal{T}^{\mathrm{MR}}_{P_6,X} \uparrow 1 = \mathcal{T}^{\mathrm{MR}}_{P_6,X}(\emptyset) = \{x_1, x_2, x_3\}$ in view of the facts r_{24}, r_{25}, and r_{26}. Then, the condition of $(\{x_1, x_2, x_3\}, X) \models_{\mathrm{MR}} B(r_{27})$ is fulfilled if there is some interpretation $Z \subseteq \{x_1, x_2, x_3\}$ such that $\alpha(\mathrm{SUM}[2 : x_1, 2 : x_2, 3 : x_3, -2 : p] = 5, Z) = 5$. This is the case for $Z = \{x_1, x_3\}$ (and $Z = \{x_2, x_3\}$), so that

$$\mathcal{T}^{\mathrm{MR}}_{P_6,X} \uparrow \infty = \mathcal{T}^{\mathrm{MR}}_{P_6,X} \uparrow 2 = \mathcal{T}^{\mathrm{MR}}_{P_6,X}(\{x_1, x_2, x_3\}) = \{x_1, x_2, x_3, p\} = X.$$

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

When we replace the rule r_{27} by

$$p \leftarrow \text{SUM}[2 : x_1, 2 : x_2, 3 : x_3, -1 : p] = 6 \qquad (r'_{27})$$

in the modified program P'_6, there is no $Z \subseteq \{x_1, x_2, x_3\}$ such that $\alpha(\text{SUM}[2 : x_1, 2 : x_2, 3 : x_3, -1 : p] = 6, Z) = 6$, so that $\mathcal{T}^{\text{MR}}_{P'_6,X} \uparrow \infty = \mathcal{T}^{\text{MR}}_{P'_6,X} \uparrow 1 = \{x_1, x_2, x_3\}$ disproves X to be an MR-answer set of P'_6. Both programs P_6 and P'_6 are devised such that the construction of $\mathcal{T}^{\text{MR}}_{P_6,X} \uparrow \infty$ or $\mathcal{T}^{\text{MR}}_{P'_6,X} \uparrow \infty$, respectively, involves checking whether an instance of the *subset sum* problem is satisfiable. If so, the weight associated with the atom p to be concluded is set such that the desired bound is obtained by summing up all weights. As programs following the same scheme along with an interpretation consisting of all atoms can be used to decide arbitrary instances of *subset sum*, the Check task for MR-answer sets is NP-complete in the presence of non-convex aggregate expressions of the form $\text{SUM}[w_1 : p_1, \ldots, w_n : p_n] = w_0$.

Rules like r_{27} and r'_{27} can also be adjusted to use the aggregation function AVG instead of SUM by introducing an additional fact $x \leftarrow \top$ and taking the inverse of the original bound as weight for x:

$$p \leftarrow \text{AVG}[2 : x_1, 2 : x_2, 3 : x_3, -5 : x, -2 : p] = 0 \qquad (r_{28})$$

$$p \leftarrow \text{AVG}[2 : x_1, 2 : x_2, 3 : x_3, -6 : x, -1 : p] = 0 \qquad (r'_{28})$$

The condition of $(\{x_1, x_2, x_3, x\}, X \cup \{x\}) \models_{\text{MR}} B(r_{28})$ or $(\{x_1, x_2, x_3, x\}, X \cup \{x\}) \models_{\text{MR}} B(r'_{28})$, respectively, then again amounts to deciding the satisfiability of an underlying instance of *subset sum*. Hence, we have that the Check task for MR-answer sets is likewise NP-complete for programs including non-convex aggregate expressions of the form $\text{AVG}[w_1 : p_1, \ldots, w_n : p_n] = w_0$. ■

4.2 Answer set existence

The Exist reasoning task addresses the decision problem of whether a program P has some Δ-answer set for $\Delta \in \{\text{FFLP}, \text{GZ}, \text{LPST}, \text{MR}, \text{DPB}\}$, that is, a decision problem similar to Check but having no fixed interpretation $X \subseteq \mathcal{P}$. The summary of completeness properties for particular complexity classes is provided in Table 3, where completeness now also applies for P in view of the well-known P-completeness of deciding whether the $\subseteq$-minimal model of a positive program includes some atom of interest (Dantsin *et al.* 2001). In fact, when using (monotone) aggregate expressions of the form $\text{SUM}[1 : p] > 0$ to represent propositional atoms p in the bodies of rules, all Δ-answer set semantics reproduce the $\subseteq$-minimal model of a positive program as the unique Δ-answer set. Given that our notion of non-disjunctive programs syntactically allows for constraints, a program does not necessarily have a ($\subseteq$-minimal) model though, even when all aggregate expressions are monotone, which makes the Exist task non-trivial and thus P-complete. While going beyond the simple form $\text{SUM}[1 : p] > 0$ of monotone aggregate expressions lets the GZ-answer set semantics diverge from the other aggregate semantics, as observed on Example 11, such additional syntactic freedom does not increase the computational complexity any further because the Δ-answer set semantics, for $\Delta \in \{\text{FFLP}, \text{LPST}, \text{MR}, \text{DPB}\}$, still coincide and the constructions by means of different immediate consequence operators remain tractable.

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Regarding anti-monotone and convex aggregate expressions, the rows for the Exist task in Table 3 follow the pattern of Check, where P-membership or coNP-completeness, respectively, of Check leads to NP- or Σ_2^{P}-completeness of Exist. For NP-hardness, anti-monotone aggregate expressions of the form $\text{SUM}[1 : p] < 1$ are sufficient, as they allow for representing negated propositional atoms p in rule bodies, which directly lead to elevated complexity of deciding whether some (standard) answer set exists (Marek and Truszczyński 1991). Likewise, the Σ_2^{P}-hardness of Exist for DPB-answer sets follows from the simultaneous availability of monotone and anti-monotone aggregate expressions of the form $\text{SUM}[1 : p] > 0$ or $\text{SUM}[1 : p] < 1$, as the reduction from quantified Boolean formulas (with one quantifier alternation) by Denecker *et al.* (2004) shows.

The additional availability of non-convex aggregate expressions in programs with arbitrary aggregate expressions does not increase the complexity of Exist beyond NP for GZ- as well as MR-answer sets. Regarding GZ-answer sets, the strong condition $\text{AT}(A) \cap Y = \text{AT}(A) \cap X$ for $(Y, X) \models_{\text{GZ}} A$ is unaffected by the monotonicity of an aggregate expression A (Alviano and Leone 2015), as already observed on the tractability of the Check task. The latter reasoning task happens to be in NP for MR-answer sets, so that deciding whether a program including non-convex aggregate expressions has some MR-answer set does not involve orthogonal combinatorial problems. Unlike that, the Σ_2^{P}-hardness of Exist for DPB-answer sets is established already for programs with convex aggregate expressions, and non-convex aggregate expressions do not make a difference here because neither determining candidate models X of a program P nor checking whether $\mathcal{T}_{P,X}^{\text{DPB}} \uparrow \infty = X$ becomes more complex in their presence.

The complexity of Exist for FFLP-answer sets correlates to the monotonicity of aggregate expressions, given the same consideration as for Check that (proper) disjunctive rules can be captured in terms of non-disjunctive programs including non-convex aggregate expressions (Ferraris 2011). This yields Σ_2^{P}-completeness of the Exist task for FFLP-answer sets of programs with non-convex aggregate expressions regardless of their particular aggregation functions, comparison operators, weights and bounds (Alviano and Faber 2013). While the complexity of Exist is in general the same for LPST-answer sets, the Σ_2^{P}-hardness again depends on the complexity of deciding the satisfiability of the complement of a (non-convex) aggregate expression A. That is, the complexity drops to NP without aggregate expressions $\text{AGG}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$ such that $\text{AGG} \in \{\text{SUM}, \text{TIMES}, \text{AVG}\}$ because the underlying Check task for LPST-answer sets gets tractable when programs do not involve aggregate expressions of this form. As further detailed in the next example, aggregate expressions with SUM or AVG along with the comparison operator $\neq$ allow for expressing the Σ_2^{P}-complete *generalized subset sum* problem (Berman *et al.* 2002), so that the same complexity is obtained for LPST-answer sets of programs including such non-convex aggregate expressions. Although we are unaware of literature addressing the complexity of a corresponding generalized version of the *subset product* problem, the coNP-hardness of the Check task suggests that Exist remains Σ_2^{P}-hard with TIMES as well.

Example 14

As checked in Examples 6 and 10, the interpretation $X = \{y_1, x_2, z_1, z_2, p\}$ is the unique Δ-answer set, for $\Delta \in \{\text{FFLP}, \text{LPST}, \text{DPB}\}$, of the program P_3:

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

$$x_1 \leftarrow \text{SUM}[1 : y_1] < 1 \tag{r_6}$$
$$y_1 \leftarrow \text{SUM}[1 : x_1] < 1 \tag{r_7}$$
$$x_2 \leftarrow \text{SUM}[1 : y_2] < 1 \tag{r_8}$$
$$y_2 \leftarrow \text{SUM}[1 : x_2] < 1 \tag{r_9}$$
$$z_1 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{10}}$$
$$z_2 \leftarrow \text{SUM}[1 : p] > 0 \tag{r_{11}}$$
$$p \leftarrow \text{SUM}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2] \neq 5 \tag{r_{12}}$$
$$\bot \leftarrow \text{SUM}[1 : p] < 1. \tag{r_{13}}$$

In fact, $\{y_1, y_2\} \cap X = \{y_1\}$ represents the single solution to the instance

$$\exists y_1 y_2 \forall z_1 z_2 (1 \cdot y_1 + 2 \cdot y_2 + 2 \cdot z_1 + 3 \cdot z_2 \neq 5)$$

of the *generalized subset sum* problem. The rules r_6, r_7, r_8, and r_9 make use of the auxiliary atoms x_1 and x_2 to express exclusive choices between x_1 and y_1 as well as x_2 and y_2, where the chosen atoms y_i stand for a solution candidate. If the candidate is indeed a solution, any truth assignment of the atoms z_i leads to a weighted sum that is different from the bound, which is 5 for our instance. In this case, we have that $Z \models B(r_{12})$ for every interpretation $\{y_1, y_2\} \cap X \subseteq Z \subseteq X$, so that the head atom p must necessarily be true and is also concluded by the operators $\mathcal{T}^{\text{LPST}}_{P_3,X}$ as well as $\mathcal{T}^{\text{DPB}}_{P_3,X}$. The remaining rules r_{10} and r_{11} establish that the atoms z_i follow from p, which makes sure that a potential counterexample, that is, some truth assignment of the atoms z_i such that the weighted sum matches the given bound, corresponds to a smaller interpretation $Y \subset X$ disproving X to be a Δ-answer set of P_3. Finally, the constraint r_{13} expresses that p must be true, while its provability relies on r_{12}, and thus eliminates any solution candidate that would directly give the bound as weighted sum when all of the atoms z_i are assigned to false.

Introducing an additional fact $x \leftarrow \top$ and replacing r_{12} by

$$p \leftarrow \text{AVG}[1 : y_1, 2 : y_2, 2 : z_1, 3 : z_2, -5 : x] \neq 0 \tag{r'_{12}}$$

leads to a modified program P'_3 including a non-convex aggregate expression with AVG instead of SUM. This aggregate expression takes the inverse of the original bound as weight for a necessarily true atom x, so that the average happens to be 0 if and only if the weighted sum over the remaining (true) atoms matches the bound. Hence, we have that Δ-answer sets $X \cup \{x\}$ of P'_3 correspond to Δ-answer sets X of P_3 for $\Delta \in \{\text{FFLP}, \text{LPST}, \text{DPB}\}$.

Given that programs following the scheme of P_3 or P'_3, respectively, allow for expressing arbitrary instances of *generalized subset sum*, we conclude that the Exist task for the three aggregate semantics of interest, in particular, LPST-answer sets of programs with non-convex aggregate expressions as used in P_3 and P'_3, is Σ^{P}_2-hard. Such elevated complexity is not obtained for GZ- and MR-answer sets, and as discussed in Example 10, the program P_3 does not have any GZ-answer set although the corresponding instance of *generalized subset sum* is satisfiable, while X and two more interpretations that do not represent solutions are MR-answer sets of P_3. ■

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Finally, note that the computational complexity of the Check and Exist reasoning tasks is a common subject of investigation for specific logic programs. In this regard, the comparably low complexity for GZ-answer sets does not come as a surprise, as avoiding so-called vicious circles (Gelfond and Zhang 2019) circumvents elevated complexity due to (sophisticated) aggregate expressions. Hardness properties concerning DPB-answer sets are immediate consequences of the simultaneous availability of monotone and anti-monotone aggregate expressions, given the computational complexity of the so-called ultimate semantics (Denecker *et al.* 2004), and the complexity of reasoning tasks for FFLP-answer sets has been thoroughly studied in the corresponding literature (Alviano and Faber 2013; Faber *et al.* 2011; Ferraris 2011). For LPST-answer sets, the elevated complexity due to aggregate expressions of the form $\text{SUM}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$ or $\text{AVG}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$ has been put forward by Son and Pontelli (2007), and we here add $\text{TIMES}[w_1 : p_1, \ldots, w_n : p_n] \neq w_0$ to these considerations. To our knowledge, the computational complexity of reasoning tasks for MR-answer sets has not been investigated in depth, and the NP-hardness of Check for programs with aggregate expressions $\text{SUM}[w_1 : p_1, \ldots, w_n : p_n] = w_0$ or $\text{AVG}[w_1 : p_1, \ldots, w_n : p_n] = w_0$ was unrecognized before.

5 Discussion

As the variety of proposals investigated in Section 3 shows, the design of a general semantics of aggregates and efficient implementations thereof have been long-standing challenges. The intricacy results from two particularities: unlike for conjunctions of literals, aggregate expressions can yield a non-convex satisfiability pattern, and the handling of (explicit) negation by the NOT connective requires additional care. In the following, we discuss such phenomena, related rewriting methods and first-order semantics used to implement or generalize aggregates, respectively, and further extensions of logic programs to custom aggregate expressions.

The pioneering SMODELS system (Simons *et al.* 2002) handles *weight constraints* $\text{SUM}[w_1 : \ell_1, \ldots, w_n : \ell_n] \geq w_0$ such that each ℓ_i, for $1 \leq i \leq n$, is a literal of the form $\ell_i = p$ or $\ell_i = \text{NOT } p$ over some atom $p \in \mathcal{P}$. While negative weights w_i can be supplied in the input, weighted literals $w_i : \ell_i$ are transformed according to

$$\tau(w_i : \ell_i) = \begin{cases} w_i : \ell_i & , \text{if } w_i \geq 0, \\ -w_i : \overline{\ell_i} & , \text{if } w_i < 0, \end{cases}$$

where $\overline{\ell_i} = \text{NOT } p$, if $\ell_i = p$, or $\overline{\ell_i} = p$, if $\ell_i = \text{NOT } p$, denotes the complement of a literal ℓ_i, in the mapping to $\text{SUM}[\tau(w_i : \ell_i) \mid 1 \leq i \leq n] \geq w_0 - \sum_{1 \leq i \leq n, w_i < 0} w_i$, based on the idea that sanctioning some literal by a negative weight is in terms of satisfiability the same as associating its complement with the corresponding positive amount. Given that the reduct adopted by SMODELS evaluates negative literals, the transformed weight constraints without negative weights become monotone in the context of checking the provability of true atoms. Hence, deciding whether a given interpretation is an answer set gets tractable, which diverges from the coNP-completeness of FFLP-answer set checking for programs with non-convex aggregate expressions (cf. Table 3), as already established in the absence of negative literals built by means of the NOT connective. This

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

complexity gap brings about peculiarities, considering that $\{p\}$ is an answer set produced by SMODELS as well as a Δ-answer set, for $\Delta \in \{\mathrm{FFLP}, \mathrm{LPST}, \mathrm{MR}, \mathrm{DPB}\}$, of a program with just the rule $p \leftarrow \mathrm{SUM}[0 : p] \geq 0$. When the rule is changed to $p \leftarrow \mathrm{SUM}[1 : p, -1 : p] \geq 0$, we have that $\{p\}$ remains a Δ-answer set, while SMODELS transforms the rule to $p \leftarrow \mathrm{SUM}[1 : p, 1 : \mathrm{NOT}\ p] \geq 1$ and takes $p \leftarrow \mathrm{SUM}[1 : p] \geq 1$ as reduct relative to the unique model $\{p\}$ – In practice, negative weights are eliminated by the front-end LPARSE (Syrjänen 2001) during grounding, which then passes the transformed weight constraints on to SMODELS or other ASP solvers. This yields the empty interpretation $\emptyset$ as a smaller model disproving $\{p\}$, so that the transformation by SMODELS leads to different semantics of the rules $p \leftarrow \mathrm{SUM}[0 : p] \geq 0$ and $p \leftarrow \mathrm{SUM}[1 : p, -1 : p] \geq 0$. A common issue about transformations of logic programs, in general, and of aggregate expressions, in particular, is that satisfiability preservation does not necessarily yield equitable provability of atoms. For guaranteeing replaceability regardless of the specific program context, strong equivalence (Lifschitz *et al.* 2001) needs to be established, and Bomanson *et al.* (2020), Ferraris (2011), and Gebser *et al.* (2015a) investigate this concept for aggregate expressions.

According to Ferraris and Lifschitz (2005), a weight constraint $\mathrm{SUM}[w_1 : \ell_1, \ldots, w_n : \ell_n] \geq w_0$ with positive weights w_i, for $1 \leq i \leq n$, is equivalent to

$$\bigvee_{I \subseteq \{1,\ldots,n\}, \sum_{i \in I} w_i \geq w_0} \bigwedge_{i \in I} \ell_i,$$

where the disjunction can be understood as a shorthand for several alternative rule bodies. Similar unfoldings of aggregate expressions have been investigated by Pelov *et al.* (2003) and Son *et al.* (2006), and yield semantic correspondences to LPST-answer sets (Son and Pontelli 2007). More compact rewritings of weight constraints to aggregate-free rules, devised for increasing the range of applicable solving systems, are based on sequential weight counters (Ferraris and Lifschitz 2005) or merge-sorting (Bomanson *et al.* 2014), inspired by corresponding encodings of pseudo-Boolean constraints in propositional logic (Bailleux *et al.* 2009; Hölldobler *et al.* 2012; Roussel and Manquinho 2009). In general, when the joint use of positive and negative weights makes a weight constraint non-convex, the more sophisticated formula

$$\bigwedge_{I \subseteq \{1,\ldots,n\}, \sum_{i \in I} w_i < w_0} \left(\bigwedge_{i \in I} \ell_i \rightarrow \bigvee_{i \in \{1,\ldots,n\} \setminus I} \ell_i \right)$$

has been shown to be semantic-preserving (Ferraris 2011). This scheme introduces nested implications, which lead to elevated complexity of reasoning tasks, as discussed in Section 4.1. Such complex constructs are not directly supported by the ASP systems CLINGO and DLV, whose propagation procedures (Faber *et al.* 2008; Gebser *et al.* 2009) are designed for (anti-)monotone aggregate expressions – A convex aggregate expression can be decomposed into the conjunction of a monotone and an anti-monotone aggregate expression (Liu and Truszczyński 2006), for example, by syntactically taking the comparison operators $\leq$ and $\geq$ to represent $=$. Reasoning on non-convex aggregate expressions can still be accomplished by means of a rewriting to disjunctive rules with monotone aggregate expressions (Alviano *et al.* 2015), where (proper) disjunctive rules are needed only if a non-convex aggregate expression occurs in positively recursive rules, which are necessary to express a Σ_2^{P}-hard problem like *generalized subset sum* in Example 14. Since this rewriting is implemented in the grounding procedure of CLINGO (Gebser *et al.* 2015b), any disjunctive ASP solver that handles recursive monotone aggregate expressions, for

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

example, used to encode *company controls*, can be applied to reason on non-convex aggregate expressions as well. The input language of CLINGO (Gebser *et al.* 2015a), however, excludes the aggregation functions TIMES and AVG, which are also not part of the *ASP-Core-2* specification of a common first-order language for ASP systems (Calimeri *et al.* 2019). Moreover, CLINGO rewrites aggregate expressions with MIN or MAX to aggregate-free rules, so that only COUNT and SUM effectively lead to monotone aggregate expressions that use the SUM aggregation function at the ground level.

Current ASP solvers deal with weight constraints $\text{SUM}[w_1 : \ell_1, \ldots, w_n : \ell_n] \geq w_0$ with positive weights w_i, and grounding (components of) systems like CLINGO (Gebser *et al.* 2015b), DLV (Calimeri *et al.* 2020), and LPARSE (Syrjänen 2001) establish such a format, using transformations like those given by Alviano *et al.* (2015). In the context of non-disjunctive programs, translational approaches to propositional logic (Janhunen and Niemelä 2011) rely on the aforementioned rewritings to aggregate-free rules (Ferraris and Lifschitz 2005; Bomanson *et al.* 2014), and even more compact, linear representations in terms of pseudo-Boolean constraints or integer programming (Gebser *et al.* 2014) enable the use of corresponding back-end solvers to compute answer sets. The ASP systems CLINGO, DLV, and SMODELS incorporate propagation procedures (Faber *et al.* 2008; Gebser *et al.* 2009; Simons *et al.* 2002) that extend Boolean constraint propagation (Roussel and Manquinho 2009) from pseudo-Boolean constraints to logic programs. Their restrictions are that SMODELS handles non-disjunctive programs only, aggregate expressions must be non-recursive for DLV, and CLINGO does not use compact data structures for aggregate expressions in positively recursive rules such that their head atoms occur together in some (proper) disjunctive rule head (i.e., the program part under consideration is not head-cycle-free Ben-Eliyahu and Dechter 1994), while more space-consuming rewriting to aggregate-free rules is performed otherwise. Moreover, the recent ASP solver WASP (Alviano *et al.* 2019) implements a collective propagation procedure (Alviano *et al.* 2018) for aggregate expressions $\text{SUM}[w_1 : \ell_1, \ldots, w_n : \ell_n] \geq w_0$ with the same weighted literals $w_1 : \ell_1, \ldots, w_n : \ell_n$ and different bounds w_0, and the lazy-grounding ASP system ALPHA (Weinzierl *et al.* 2020) features an incremental on-demand rewriting (Bomanson *et al.* 2019) to aggregate-free rules.

While the semantics by Faber *et al.* (2011) and Ferraris (2011) agree for the syntax of aggregate expressions considered in this survey, that is, aggregates over propositional atoms, the different reduct notions, which either eliminate falsified expressions and thus negative literals or not, lead to distinct outcomes in the presence of NOT. For instance, the reduct of $p \leftarrow \text{SUM}[1 : p, 1 : \text{NOT } p] \geq 1$ relative to the interpretation $\{p\}$ is the rule itself according to Faber *et al.* (2011), and $p \leftarrow \text{SUM}[1 : p] \geq 1$ by Ferraris' 2011 definition. Hence, either $\{p\}$ or $\emptyset$ is obtained as $\subseteq$-minimal model of the reduct, so that the semantics disagree about whether $\{p\}$ is an answer set. Similarly, $p \leftarrow \text{NOT SUM}[1 : p] < 1$ yields the rule as such or $p \leftarrow \top$ as the reduct relative to $\{p\}$, resulting in either $\emptyset$ or $\{p\}$ as $\subseteq$-minimal model. That is, answer sets according to Faber *et al.* (2011) and Ferraris (2011) can be mutually distinct when the NOT connective is used in front of aggregate expressions or propositional atoms subject to aggregation functions. To circumvent such discrepancies, the *ASP-Core-2* language specification (Calimeri *et al.* 2019) requires aggregate-stratification (Faber *et al.* 2008), which restricts occurrences of aggregate expressions in a program to be non-recursive. Under this condition, apart from DPB-answer sets that differ already for aggregate-free programs (Denecker *et al.* 2001;

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

2004), the aggregate semantics investigated in Section 3 agree, and aggregate expressions may not increase the computational complexity of reasoning tasks. Harrison and Lifschitz (2019) relax the aggregate-stratification condition and show that answer sets according to Faber *et al.* (2011) and Ferraris (2011) coincide when recursive aggregate expressions do not involve the NOT connective, that is, they are of the form (1), while no syntactic restrictions are imposed otherwise, for example, for aggregate expressions occurring in constraints. Notably, the correspondence in Harrison and Lifschitz (2019) is established at the first-order level, thus connecting prior first-order generalizations of the two semantics (Bartholomew *et al.* 2011; Gebser *et al.* 2015a) based on second-order logic or infinitary formulas, respectively. Moreover, answer sets according to Faber *et al.* (2011) and Ferraris (2011) can be characterized in terms of each other (Lee and Meng 2009; Truszczyński 2010), so that their expressiveness is the same regardless of semantic differences arising on the NOT connective.

Beyond their use for a compact representation of properties on sets of atoms, semantics for aggregate expressions have been taken as basis for defining the answer sets of logic programs with further extensions, such as description logic and higher-order logic programs with external atoms (Shen *et al.* 2014). Similar to aggregate expressions, monotonicity properties of external atoms affect reasoning about them, and dedicated solving techniques take advantage of so-called assignment-monotonicity (Eiter *et al.* 2018). Extensions of ASP systems by theory propagators (Cuteri *et al.* 2020; Elkabani *et al.* 2004; Janhunen *et al.* 2017; Lierler and Susman 2017) likewise interpret specific atoms as custom aggregate expressions and incorporate respective procedures for propagating their truth values. In this broad sense, the notion of an aggregate includes any method of evaluating atoms as a compound, where some frequently used aggregation functions, in particular, the SUM aggregation function, are accommodated off-the-shelf in the modeling language of ASP.

While there is already a considerable body of work on aggregates, there clearly are numerous open issues to be addressed. We would like to outline some of them, of course without any claim of completeness. First of all, identifying sublanguages on which different semantics coincide is still a relevant topic. As mentioned earlier, most semantics agree on aggregate-stratified programs, and many also correspond for programs with convex aggregates. Another issue is that encodings with unstratified aggregate occurrences are fairly rare at the moment, especially when considering "real-world" applications. We believe, however, that providing a meaningful semantics for language constructs is important also when there are no frequent applications for them. Nevertheless, potential application fields might lie, for instance, in the area of analysis of dynamic systems, for example, addressing the location of fixpoints in biological systems. A related question is whether the proposed languages are actually fit for various practical purposes. We believe that postulating formal properties that semantics for programs with aggregates should satisfy is important and still lacking. For example, formalizations of the Closed World Assumption (Reiter 1977) for programs with aggregates would be of interest. There is also work arguing that certain programs are not handled well by the existing semantics, yet suggestions as, for example, by Alviano and Faber (2019) for "repairing" existing approaches to cover such cases do not seem satisfactory so far. Finally, we would like to mention that more implementations are needed. Most proposed semantics have never been supported by any system, making comparisons difficult and applications impossible,

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

and merely CLINGO implements non-convex aggregate expressions in positively recursive rules (under FFLP-answer set semantics). Hence, even implementations that are not at all geared towards efficiency would be useful. In summary, there is a substantial amount of room for future work on aggregates in ASP.

References

Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M. and Ricca, F. 2019. Evaluation of disjunctive programs in WASP. In *Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19)*, M. Balduccini, Y. Lierler and S. Woltran, Eds. Lecture Notes in Artificial Intelligence, vol. 11481. Springer-Verlag, 241–255.

Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P. and Zangari, J. 2017. The ASP system DLV2. In *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, M. Balduccini and T. Janhunen, Eds. Lecture Notes in Artificial Intelligence, vol. 10377. Springer-Verlag, 215–221.

Alviano, M., Dodaro, C. and Maratea, M. 2018. Shared aggregate sets in answer set programming. *Theory and Practice of Logic Programming 18,* 3-4, 301–318.

Alviano, M. and Faber, W. 2013. The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In *Proceedings of the Twelfth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'13)*, P. Cabalar and T. Son, Eds. Lecture Notes in Artificial Intelligence, vol. 8148. Springer-Verlag, 67–72.

Alviano, M. and Faber, W. 2019. Chain answer sets for logic programs with generalized atoms. In *Proceedings of the Sixteenth European Conference on Logics in Artificial Intelligence (JELIA'19)*, F. Calimeri, N. Leone and M. Manna, Eds. Lecture Notes in Computer Science, vol. 11468. Springer-Verlag, 462–478.

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

Alviano, M. and Leone, N. 2015. Complexity and compilation of GZ-aggregates in answer set programming. *Theory and Practice of Logic Programming 15,* 4-5, 574–587.

Apt, K., Blair, H. and Walker, A. 1987. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann Publishers, Chapter 2, 89–148.

Bailleux, O., Boufkhad, Y. and Roussel, O. 2009. New encodings of pseudo-Boolean constraints into CNF. In *Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, O. Kullmann, Ed. Lecture Notes in Computer Science, vol. 5584. Springer-Verlag, 181–194.

Bartholomew, M., Lee, J. and Meng, Y. 2011. First-order semantics of aggregates in answer set programming via modified circumscription. In *Proceedings of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, E. Davis, P. Doherty and E. Erdem, Eds. AAAI Press, 16–22.

Ben-Eliyahu, R. and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence 12,* 1-2, 53–87.

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

Bomanson, J., Gebser, M. and Janhunen, T. 2014. Improving the normalization of weight rules in answer set programs. In *Proceedings of the Fourteenth European Conference on Logics*

in Artificial Intelligence (JELIA'14), E. Fermé and J. Leite, Eds. Lecture Notes in Artificial Intelligence, vol. 8761. Springer-Verlag, 166–180.

Bomanson, J., Janhunen, T. and Niemelä, I. 2020. Applying visible strong equivalence in answer-set program transformations. *ACM Transactions on Computational Logic 4,* 21, 33:1–33:41.

Bomanson, J., Janhunen, T. and Weinzierl, A. 2019. Enhancing lazy grounding with lazy normalization in answer-set programming. In *Proceedings of the Thirty-third National Conference on Artificial Intelligence (AAAI'19)*, P. Van Hentenryck and Z. Zhou, Eds. AAAI Press, 2694–2702.

Brewka, G., Eiter, T. and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M. and Verwer, S. 2015. Predicate logic as a modeling language: Modeling and solving some machine learning and data mining problems with IDP3. *Theory and Practice of Logic Programming 15,* 6, 783–817.

Calimeri, F., Dodaro, C., Fuscà, D., Perri, S. and Zangari, J. 2020. Efficiently coupling the I-DLV grounder with ASP solvers. *Theory and Practice of Logic Programming 20,* 2, 205–224.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2019. ASP-Core-2 input language format. *Theory and Practice of Logic Programming 20,* 2, 294–309.

Codd, E. 1970. A relational model of data for large shared data banks. *Communications of the ACM 13,* 6, 377–387.

Codd, E. 1972. Relational completeness of data base sublanguages. *Research Report/RJ/IBM/San Jose, California RJ987.*

Cuteri, B., Dodaro, C., Ricca, F. and Schüller, P. 2020. Overcoming the grounding bottleneck due to constraints in ASP solving: Constraints become propagators. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI'20)*, C. Bessiere, Ed. ijcai.org, 1688–1694.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33,* 3, 374–425.

Denecker, M., Marek, V. and Truszczyński, M. 2004. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation 192,* 1, 84–121.

Denecker, M., Pelov, N. and Bruynooghe, M. 2001. Ultimate well-founded and stable semantics for logic programs with aggregates. In *Proceedings of the Seventeenth International Conference on Logic Programming (ICLP'01)*, P. Codognet, Ed. Lecture Notes in Computer Science, vol. 2237. Springer-Verlag, 212–226.

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

Eiter, T., Kaminski, T., Redl, C. and Weinzierl, A. 2018. Exploiting partial assignments for efficient evaluation of answer set programs with external source access. *Journal of Artificial Intelligence Research 62*, 665–727.

Elkabani, I., Pontelli, E. and Son, T. 2004. Smodels with CLP and its applications: A simple and effective approach to aggregates in ASP. In *Proceedings of the Twentieth International Conference on Logic Programming (ICLP'04)*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer-Verlag, 73–89.

Faber, W., Pfeifer, G. and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175,* 1, 278–298.

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

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Ferraris, P. 2011. Logic programs with propositional connectives and aggregates. *ACM Transactions on Computational Logic 12,* 4, 25:1–25:40.

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

Ganguly, S., Greco, S. and Zaniolo, C. 1995. Extrema predicates in deductive databases. *Journal of Computer and System Sciences 51,* 2, 244–259.

Garey, M. and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. Freeman and Co., New York.

Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V. and Schaub, T. 2015a. Abstract Gringo. *Theory and Practice of Logic Programming 15,* 4-5, 449–463.

Gebser, M., Janhunen, T. and Rintanen, J. 2014. Answer set programming as SAT modulo acyclicity. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence (ECAI'14)*, T. Schaub, G. Friedrich, and B. O'Sullivan, Eds. IOS Press, 351–356.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2009. On the implementation of weight constraint rules in conflict-driven ASP solvers. In *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. Lecture Notes in Computer Science, vol. 5649. Springer-Verlag, 250–264.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming 19,* 1, 27–82.

Gebser, M., Kaminski, R. and Schaub, T. 2015b. Grounding recursive aggregates: Preliminary report. In *Proceedings of the Third Workshop on Grounding, Transforming, and Modularizing Theories with Variables (GTTV'15)*, M. Denecker and T. Janhunen, Eds.

Gebser, M., Kaufmann, B. and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187-188*, 52–89.

Gelfond, M. and Leone, N. 2002. Logic programming and knowledge representation — the A-Prolog perspective. *Artificial Intelligence 138,* 1-2, 3–38.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Gelfond, M. and Zhang, Y. 2019. Vicious circle principle, aggregates, and formation of sets in ASP based languages. *Artificial Intelligence 275*, 28–77.

Harrison, A. and Lifschitz, V. 2019. Relating two dialects of answer set programming. *Theory and Practice of Logic Programming 19,* 5-6, 1006–1020.

Hölldobler, S., Manthey, N. and Steinke, P. 2012. A compact encoding of pseudo-Boolean constraints into SAT. In *Proceedings of the Thirty-fifth Annual German Conference on Artificial Intelligence (KI'12)*, B. Glimm and A. Krüger, Eds. Lecture Notes in Computer Science, vol. 7526. Springer-Verlag, 107–118.

Janhunen, T., Kaminski, R., Ostrowski, M., Schaub, T., Schellhorn, S. and Wanko, P. 2017. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming 17,* 5-6, 872–888.

Janhunen, T. and Niemelä, I. 2011. Compact translations of non-disjunctive answer set programs to propositional clauses. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of his 65th Birthday*, M. Balduccini and T. Son, Eds. Lecture Notes in Computer Science, vol. 6565. Springer-Verlag, 111–130.

Kemp, D. and Stuckey, P. 1991. Semantics of logic programs with aggregates. In *Proceedings of the 1991 International Symposium on Logic Programming (ISLP'91)*, V. Saraswat and K. Ueda, Eds. MIT Press, 387–401.

Klug, A. 1982. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM 29,* 3, 699–717.

Lee, J. and Meng, Y. 2009. On reductive semantics of aggregates in answer set programming. In *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, E. Erdem, F. Lin and T. Schaub, Eds. Lecture Notes in Artificial Intelligence, vol. 5753. Springer-Verlag, 182–195.

Lierler, Y. and Susman, B. 2017. On relation between constraint answer set programming and satisfiability modulo theories. *Theory and Practice of Logic Programming 17,* 4, 559–590.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138,* 1-2, 39–54.

Lifschitz, V., Pearce, D. and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2,* 4, 526–541.

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

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

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

Lloyd, J. 1987. *Foundations of Logic Programming.* Springer-Verlag.

Marek, V. and Remmel, J. 2004. Set constraints in logic programming. In *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Artificial Intelligence, vol. 2923. Springer-Verlag, 167–179.

Marek, V. and Truszczyński, M. 1991. Autoepistemic logic. *Journal of the ACM 38,* 3, 588–619.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, K. Apt, V. Marek, M. Truszczyński and D. Warren, Eds. Springer-Verlag, 375–398.

Mazuran, M., Serra, E. and Zaniolo, C. 2013. Extending the power of Datalog recursion. *Journal on Very Large Data Bases 22,* 4, 471–493.

Mumick, I., Pirahesh, H. and Ramakrishnan, R. 1990. The magic of duplicates and aggregates. In *Proceedings of the Sixteenth International Conference on Very Large Data Bases (VLDB'90)*, D. McLeod, R. Sacks-Davis and H. Schek, Eds. Morgan Kaufmann Publishers, 264–277.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

Özsoyoglu, G., Özsoyoglu, Z. and Matos, V. 1987. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems 12,* 4, 566–592.

Pelov, N., Denecker, M. and Bruynooghe, M. 2003. Translation of aggregate programs to normal logic programs. In *Proceedings of the Second International Workshop on Answer Set Programming (ASP'03)*, M. de Vos and A. Provetti, Eds. CEUR Workshop Proceedings (CEUR-WS.org), 29–42.

Pelov, N., Denecker, M. and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming 7,* 3, 301–353.

Reiter, R. 1977. On closed world data bases. In *Proceedings of Workshop on Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum Press, 119–140.

Ross, K. 1994. Modular stratification and magic sets for Datalog programs with negation. *Journal of the ACM 41,* 6, 1216–1266.

https://doi.org/10.1017/S1471068422000047 Published online by Cambridge University Press

Roussel, O. and Manquinho, V. 2009. Pseudo-Boolean and cardinality constraints. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren and T. Walsh, Eds. IOS Press, Chapter 22, 695–733.

Schlipf, J. 1995. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences 51*, 64–86.

Shen, Y., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T. and Deng, J. 2014. FLP answer set semantics without circular justifications for general logic programs. *Artificial Intelligence 213*, 1–41.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1-2, 181–234.

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

Son, T., Pontelli, E. and Elkabani, I. 2006. An unfolding-based semantics for logic programming with aggregates. CoRR abs/cs/0605038.

Sudarshan, S. and Ramakrishnan, R. 1991. Aggregation and relevance in deductive databases. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases (VLDB'91)*, G. Lohman, A. Sernadas and R. Camps, Eds. Morgan Kaufmann Publishers, 501–511.

Syrjänen, T. 2001. Lparse 1.0 user's manual. www.tcs.hut.fi/Software/smodels/.

Truszczyński, M. 2010. Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artificial Intelligence 174,* 16-17, 1285–1306.

Vanbesien, L., Bruynooghe, M. and Denecker, M. 2021. Analyzing semantics of aggregate answer set programming using approximation fixpoint theory. CoRR abs/2104.14789.

van Emden, M. and Kowalski, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM 23,* 4, 733–742.

Van Gelder, A. 1992. The well-founded semantics of aggregation. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'92)*, M. Vardi and P. Kanellakis, Eds. ACM Press, 127–138.

Weinzierl, A., Taupe, R. and Friedrich, G. 2020. Advancing lazy-grounding ASP solving techniques — restarts, phase saving, heuristics, and more. *Theory and Practice of Logic Programming 20,* 5, 609–624.

Zaniolo, C., Yang, M., Das, A., Shkapsky, A., Condie, T. and Interlandi, M. 2017. Fixpoint semantics and optimization of recursive Datalog programs with aggregates. *Theory and Practice of Logic Programming 17,* 5-6, 1048–1065.