On the Properties of GZ-Aggregates in Answer Set Programming*

Mario Alviano and Nicola Leone
Department of Mathematics and Computer Science
University of Calabria, Italy
{alviano.leone}@mat.unical.it

Abstract
Gelfond and Zhang recently proposed a new stable model semantics based on Vicious Circle Principle in order to improve the interpretation of logic programs with aggregates. A detailed complexity analysis of coherence testing and cautious reasoning under the new semantics highlighted similarities and differences versus mainstream stable model semantics for aggregates, which eventually led to the design of compilation techniques for implementing the new semantics on top of existing ASP solvers.

1 Introduction
Answer set programming (ASP) is a declarative language for knowledge representation and reasoning [Brewka et al., 2011] powered by many efficient systems [Calimeri et al., 2016]. ASP specifications are sets of logic rules, possibly using disjunction and default negation, interpreted according to the stable model semantics [Gelfond and Lifschitz, 1988; 1991]. Default negation in particular eases the representation of decisions to be taken based on assumptions on unknown knowledge. In fact, a stable model $I$ of an ASP program $P$ has to satisfy all logic rules in $P$, and in addition has to satisfy a stability condition: everything in $I$ is necessary in order to satisfy all logic rules in $P$ under the assumptions provided by $I$ itself for default negated literals. Hence, the stability condition guarantees correctness of the assumptions on the unknown knowledge used in the reasoning process. Moreover, the stability condition enforces the vicious circle principle, which essentially asserts that the truth of an atom must be inferred by means of a definition not referring, directly or indirectly, to the truth of the atom itself.

The basic language is extended by several constructs to ease the representation of practical knowledge. Aggregate functions are among these extensions [Simons et al., 2002; Pelov et al., 2007; Son and Pontelli, 2007; Shen et al., 2014; Liu et al., 2010; Faber et al., 2011; Bartholomew et al., 2011], and allow to express properties on sets of atoms declaratively. For example, aggregate functions are often used to enforce functional dependencies; a rule of the following form:

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

constrains relation $R$ to satisfy the functional dependency $X \rightarrow Y$, where $X \cup Y \cup Z$ is the set of attributes of $R$, and $R'$ is the projection of $R$ on $X$. Aggregate functions are also commonly used in ASP to constrain a nondeterministic guess. For example, in the knapsack problem the total weight of the selected items must not exceed a given limit, which can be modeled by the following rule aggregating over a multiset:

\[ \bot \leftarrow \text{sum}[W, O : \text{object}(O, W, C), \text{in}(O)] > \text{limit}. \]

As a further example, aggregate functions ease the representation of logic circuits made of gates of unbounded fan-in [Gelfond and Zhang, 2014]; the following rule models that the output value of an XOR gate is 1 if an odd number of its inputs have value 1:

\[ v(O, 1) \leftarrow \text{xor}(G), \text{out}(G, O), \text{odd}[I : \text{in}(G, I), v(I, 1)]. \]

This last example is of particular interest, as it includes a recursive definition involving an aggregate. In fact, a strength of ASP is the possibility to represent and reason on recursive definitions, which are quite common in mathematics and computer science. However, when aggregates are involved in recursive definitions, the notion of stable model is nontrivial, and actually still under debate. Of the several semantics proposed for ASP programs with aggregates, two of them [Ferraris, 2011; Faber et al., 2011] are implemented in popular ASP solvers [Gebser et al., 2012; Faber et al., 2008], agree for programs without negated aggregates [Alviano et al., 2015], and are referred here as F-stable model semantics. They are essentially based on the stability condition reported at the beginning of this introduction: everything in a stable model $I$ of an ASP program $P$ is necessary for satisfying all logic rules in $P$ under the assumptions provided by $I$ itself for default negated literals.

Technically, F-stable models do not satisfy the vicious circle principle, as inference of atoms via definitions referring the truth values of the atoms themselves is not inhibited. Actually, in some cases violating the vicious circle principle is necessary for associating some stable models to a given ASP

---

*This work is based on a paper presented at the International Conference on Logic Programming (ICLP 2015) [Alviano and Leone, 2015], and was partially supported by the Italian Ministry of University and Research under PON project “Ba2Know (Business Analytics to Know) Service Innovation - LAB”, No. PON03PE_00001_1, and by Gruppo Nazionale per il Calcolo Scientifico (GNCS-INdAM).
G-stable model semantics. Simulation of integrity constraints and default negation are the semantics by and possibly default negation when interpreted according to property: the increase in complexity also adds expressive this is likely to be an unwanted artifact of G-stable models, programs with monotone aggregates is guaranteed. Indeed, stable model semantics, for which coherence of negation-free aggregates in the literature. This result is in contrast with F-hardness already for negation-free programs with a very lim-
proved in [Gelfond and Zhang, 2014]. Eventually, the simulation of integrity constraints and default negation are the gadgets that we used to prove some complexity results for G-stable model semantics.

On the other hand, there are also many cases in which G-stable models actually decrease the complexity of the reasoning tasks. In fact, while any non-convex aggregate [Alviano and Faber, 2013] is sufficient to show \( \Sigma^P_2 \)-hardness for F-coherence testing already for disjunction-free programs, G-coherence testing was proved to be NP-complete in general for disjunction-free programs. Finally, P-completeness was proved for programs with monotone aggregates if disjunction and negation are not used, a result compatible with F-stable model semantics. However, also in this case G-stable models allow to simulate integrity constraints, which is not possible with F-stable models.

As for the complexity of cautious reasoning, membership and hardness in the complementary complexity classes were proved for all the analyzed fragments of the language. These complexity results also implicitly characterize the computational complexity of another common reasoning task in ASP, known as brave reasoning, which consists in checking whether a given propositional atom is true in some stable model of an input program. In fact, brave reasoning has the same complexity of coherence testing when programs are interpreted under G-stable model semantics.

Further interesting results in [Alviano and Leone, 2015] are the two rewriting techniques for compiling programs interpreted according to G-stable model semantics into programs interpreted according to F-stable model semantics. While the first rewriting is simpler and introduces fewer auxiliary symbols, the second has the advantage of producing programs with non recursive aggregates only. These two rewritings, recalled in Section 3, are polynomial, faithful and modular translation functions [Janhunen, 2006], and are implemented in a system prototype. It is publicly available (http://alviano.net/software/g-stable-models/) and allows for experimenting with this newly proposed semantics.

### Table 1: Complexity of G-coherence testing and G-cautious reasoning.

<table>
<thead>
<tr>
<th>COHERENCE TESTING</th>
<th>CAUTIOUS REASONING</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>{}</td>
</tr>
<tr>
<td>~ K</td>
<td>NP</td>
</tr>
<tr>
<td>M</td>
<td>P</td>
</tr>
<tr>
<td>C</td>
<td>NP</td>
</tr>
<tr>
<td>N</td>
<td>NP</td>
</tr>
</tbody>
</table>

Programs with monotone aggregates are more expressive than programs with arbitrary aggregates. However, when possible, stable models obeying the vicious circle principle may be preferred, and this fact motivated an alternative semantics that was recently proposed by [Gelfond and Zhang, 2014], and is here referred to as GZ- or G-stable model semantics. In the new semantics, an assumption is also done on satisfied aggregates: they are satisfied because of the true atoms in their domains. The stability condition thus requires that everything in a stable model \( I \) of an ASP program \( I \) is necessary for satisfying all logic rules in \( I \) under the assumptions provided by \( I \) itself for both default negated literals and aggregates. Interestingly, G-stable models are F-stable models, but the converse is not always true [Alviano and Faber, 2015a].

Our previous paper [Alviano and Leone, 2015], honored with the Best Paper Award at the 31st International Conference on Logic Programming (ICLP 2015), explored this new semantics, reporting a detailed complexity analysis of coherence testing and cautious reasoning [Eiter and Gottlob, 1995], two of the main computational tasks in ASP. In a nutshell, coherence testing amounts to check the existence of a stable model of an input program, while cautious reasoning consists in checking whether a given atom is true in all stable models of the input program. A summary of the complexity results is reported in Table 1, where programs combining monotone (M), convex (C) and non-convex (N) aggregates with negation (\( \sim \)) and disjunction (\( \lor \)) are considered.

Concerning coherence testing, membership in \( \Sigma^P_2 \) was proved in [Gelfond and Zhang, 2014], and we proved \( \Sigma^P_2 \)-hardness already for negation-free programs with a very limited form of aggregate functions, referred to as monotone aggregates in the literature. This result is in contrast with F-stable model semantics, for which coherence of negation-free programs with monotone aggregates is guaranteed. Indeed, this is likely to be an unwanted artifact of G-stable models, which however also comes with an interesting and unique property: the increase in complexity also adds expressive power to the language, as aggregates referred to as monotone in the literature allow to simulate integrity constraints and possibly default negation when interpreted according to the semantics by [Gelfond and Zhang, 2014]. Eventually, the simulation of integrity constraints and default negation are the gadgets that we used to prove some complexity results for G-stable model semantics.

After defining the syntax of logic programs with aggregates, two semantics are introduced, referred to as F- [Ferraris, 2011; Faber et al., 2011] and G-stable models [Gelfond and Zhang, 2014]. (The original definitions are properly adapted to better fit the results in this paper.)

#### 2 Background

After defining the syntax of logic programs with aggregates, two semantics are introduced, referred to as F- [Ferraris, 2011; Faber et al., 2011] and G-stable models [Gelfond and Zhang, 2014]. (The original definitions are properly adapted to better fit the results in this paper.)

#### 2.1 Syntax

Let \( T,F \) denote the Boolean truth values true and false, respectively. Let \( U \) be a finite set of propositional atoms. An aggregate \( A \) is a Boolean function whose domain, denoted \( \text{dom}(A) \), is a subset of \( U \). A literal is a propositional atom or an aggregate possibly preceded by (one or more occurrences of) the negation as failure symbol \( \sim \). A rule \( r \) is an expression
of the following form:
\[ p_1 \lor \cdots \lor p_m \leftarrow l_1, \ldots, l_n \]  
(1)
where \( p_1, \ldots, p_m \) are propositional atoms, \( l_1, \ldots, l_n \) are literals, \( m \geq 1 \) and \( n \geq 0 \). Set \( \{p_1, \ldots, p_m\} \) is the head of \( r \), denoted \( H(r) \), and set \( \{l_1, \ldots, l_n\} \) is the body of \( r \), denoted \( B(r) \). A program \( \Pi \) is a finite set of rules of the form (1). The set of propositional atoms occurring in \( \Pi \) is denoted \( \text{At}(\Pi) \).

Example 1. Let \( A_1 \) be an aggregate such that \( \text{dom}(A_1) = \{a, b\} \) and \( A_1(C) = \{a, b\} \cap C \geq 1 \), for all \( C \subseteq \text{dom}(A_1) \). A program using \( A_1 \) is \( \Pi_1 = \{a \leftarrow \sim a; b \lor c \leftarrow A_1\} \).

2.2 Semantics

An interpretation \( I \) is a subset of \( \mathcal{U} \). Let \( S, S' \) be sets of interpretations, and \( C \) be a set of propositional atoms. Sets \( S \) and \( S' \) are equivalent in the context \( C \), denoted \( S \equiv_C S' \), if \( |S| = |S'| \) and \( \{C \cap I \in S\} = \{I \in S'\} \).

Aggregates are classified in three groups [Liu and Truszczynski, 2006]. An aggregate \( A \) is monotone if \( A(I) = T \) implies \( A(J) = T \), for all \( I \subseteq J \subseteq \mathcal{U} \). An aggregate \( A \) is convex if \( A(I) = A(K) = T \) implies \( A(J) = T \), for all \( I \subseteq J \subseteq K \subseteq \mathcal{U} \). The remaining aggregates are called non-convex. Note that monotone aggregates are convex, and the inclusion is strict.

Example 2. Let \( k \) be a natural number, and \( I \) be an interpretation. An aggregate \( A \) such that \( A(I) := |\text{dom}(A) \cap I| \geq k \) is monotone. An aggregate \( A \) such that \( A(I) := |\text{dom}(A) \cap I| = k \) is convex. An aggregate \( A \) such that \( A(I) := |\text{dom}(A) \cap I| \neq k \) is non-convex.

Relation \( \models \) is inductively defined as follows: for a propositional atom \( p \in \mathcal{U} \), \( I \models p \) if \( p \in I \); for an aggregate \( A \), \( I \models A \) if \( A(I) = T \); for a negated literal \( \sim I \), \( I \models \sim I \) if \( I \notin I \); for a set or conjunction \( C, I \models C \) if \( I \models p \) holds for each \( p \in C \); for a rule \( r, I \models r \) if \( H(r) \cap I \neq \emptyset \) whenever \( I \models B(r), I \models r \) is a model of a program \( \Pi \) if \( I \models \Pi, \text{i.e.}, if \( I \models r \) for all \( r \in \Pi \).

Example 3. Continuing Example 1, the models of \( \Pi_1 \), restricted to the atoms occurring in the program, are the following: \( \emptyset, \{a, b\}, \{a, c\}, \{a, b, c\}, \{b\}, \{c\}, \) and \( \{c\} \).

G-stable models. Let \( \Pi \) be a program and \( I \) an interpretation. The G-reduct of \( \Pi \) with respect to \( I \) is defined as follows: \( G(\Pi, I) = \{G(r, I) \mid r \in \Pi, I \models B(r)\} \), where \( G(r, I) = p_1 \lor \cdots \lor p_m \leftarrow l_1, \ldots, l_n \) for being of the form (1), \( G(l, I) = I \) if \( l \) is a propositional literal, \( G(l, I) = I \cap \text{dom}(A) \) if \( l \) is an aggregate \( A \), and \( G(I, I) = \emptyset \) if \( I \) is a negative literal. \( I \) is a G-stable model of \( \Pi \) if \( I \models \Pi \) and there is no \( J \subseteq I \) such that \( J \models G(\Pi, I) \). The set of G-stable models of \( \Pi \) is denoted \( \text{GSM}(\Pi) \).

Example 5. The G-stable models of \( \Pi_1 \) in Example 1 are the following: \( \emptyset \) and \( \{a, c\} \). Indeed, \( G(\Pi_1, \emptyset) = \emptyset, G(\Pi_1, \{a, b\}) = G(\Pi_1, \{a, c\}) = \{a \leftarrow b \lor c \leftarrow A_1\} \), and each model is minimal for its reduct. On the other hand, \( \{b\} \) is not an F-stable model because \( \emptyset \) is a model of \( F(\Pi_1, \{b\}) = \{b \lor c \leftarrow A_1\} \).

3 Compilation

G-stable models of a logic program can be computed via compilations into F-stable model semantics. Actually, two different rewritings are presented in this section. The first rewriting is more compact, in the sense that it introduces fewer auxiliary atoms. The second rewriting instead requires more auxiliary atoms, but has the advantage that the output program only comprise stratified aggregates (essentially, in these programs no recursive definition involves an aggregate; see [Faber et al., 2011] for a formal definition).

First Rewriting. Let \( \text{rew}(\Pi) \) be the program obtained from program \( \Pi \) by performing the following operations:

1. For each \( p \in \text{At}(\Pi) \) occurring in aggregates, a fresh atom \( p' \) and the following rules are introduced:
\[ p' \leftarrow \sim p \quad p' \leftarrow p \]  
(2)
2. For each rule \( r \in \Pi \) and each aggregate \( A \in B(r) \) such that \( \text{dom}(A) = \{p_1, \ldots, p_m\} \) (for some \( n \geq 0 \)), literals \( p'_1, \ldots, p'_n \) are added to the body of \( r \).

Example 6. Consider again program \( \Pi_1 \) from Example 1, whose G-stable models are \( \emptyset \) and \( \{a, c\} \), as shown in Example 5. Program \( \text{rew}(\Pi_1) \) is the following:
\[ a \leftarrow \sim a \quad b \lor c \leftarrow A_1, a', b' \]
\[ a' \leftarrow \sim a \quad a' \leftarrow b \quad a' \leftarrow b \]
\[ a' \leftarrow \sim a \quad b' \leftarrow b \]
Its F-stable models are the following: \( \emptyset \cup X \) and \( \{a, c\} \cup X \), where \( X = \{a', b'\} \). In fact, \( a', b' \) are necessarily true because of rules of the form (2). Moreover, note that if \( a \) is false in some model \( I \) then \( a' \) is necessarily true in any model of the reduct \( F(\Pi_1, I) \). On the other hand, if \( a \) is true in \( I \) then \( a' \) can be possibly assumed false in a model of \( F(\Pi_1, I) \). Similarly for \( b \) and \( b' \).
A drawback of this first compilation is that the evaluation of the resulting program may be in a higher complexity class than the evaluation of the original program. For example, G-coherence testing of disjunction-free programs is NP-complete in general, while a $\Sigma_2^P$ procedure will be used to test F-coherence of the rewritten program.

Example 7. Let $A_2$ be a non-convex aggregate such that $\text{dom}(A_2) = \{a, b\}$ and $A_2(C) = \{\{a, b\} \cap C \neq \emptyset$, for all $C \subseteq \text{dom}(A_2)$. Let $\Pi_2$ be the following program:

$$a \leftarrow A_2 \quad a \leftarrow b \quad b \leftarrow a$$

and $\text{rew}(\Pi_2)$ be its rewriting:

$$a \leftarrow A_2, a', b' \quad a \leftarrow b \quad b \leftarrow a$$

$$a' \leftarrow \sim_a \quad a' \leftarrow a \quad b' \leftarrow \sim_b \quad b' \leftarrow b$$

Note that $GSM(\Pi_2) = FSM(\text{rew}(\Pi_2)) = \emptyset$. However, G-coherence testing of $\Pi_2$ is in NP, while F-coherence testing of programs with non-convex aggregates such as $\text{rew}(\Pi_2)$ is $\Sigma_2^P$-complete in general. In fact, F-reducts may still contain (non-convex) aggregates; since a polyno-
time procedure for testing subset minimality of a model for such programs is unknown, a nondeterministic guess is re-
quired in general. Continuing with program $\text{rew}(\Pi_2)$, its F-reduct with respect to $\{a, b, a', b'\}$ is the following program $F(\text{rew}(\Pi_2), \{a, b, a', b'\})$:

$$a \leftarrow A_2, a', b' \quad a \leftarrow b \quad b \leftarrow a \quad a' \leftarrow a \quad b' \leftarrow b$$

which still contains the non-convex aggregate $A_2$. In this case, $\{a, b, a', b'\}$ is not an F-stable model because $\emptyset$ is a model of $F(\text{rew}(\Pi_2), \{a, b, a', b'\})$. ■

Such a drawback motivates the introduction of a second compilation. To ease the presentation, the syntax of the lan-
guage is extended with integrity constraints, that is, rules of the form (1) with empty heads. Note that the semantics pro-
vided in Section 2 can already cope with such an extension.

Second Rewriting. Let $\text{str}(\Pi)$ be the program obtained from program $\Pi$ by performing the following operations:

1. For each $p \in \text{At}(\Pi)$ occurring in aggregates, two fresh atoms $p', p''$ and the following rules are introduced:

$$p' \leftarrow \sim p \quad p' \leftarrow p \quad (3)$$

$$p'' \leftarrow \sim p'' \quad p'' \leftarrow p'' \quad (4)$$

2. For each rule $r \in \Pi$ and each aggregate $A \in B(r)$ such that $\text{dom}(A) = \{p_1, \ldots, p_n\}$, literals $p'_1, \ldots, p'_n$ are added to $B(r)$, and $A$ is replaced by a new aggregate $A''$ such that $\text{dom}(A'') = \{p'_1, \ldots, p'_n\}$ and $A''(I) = A(I[p \leftarrow p'], \{p \in \mathcal{U} \mid p' \in I\})$, for all $I \subseteq \mathcal{U}$.

Example 8. Resorting again program $\Pi_1$ from Example 1, $\text{str}(\Pi_1)$ is the following program:

$$a \leftarrow \sim a \quad b \cup c \leftarrow A'_1, a, b'$$

$$a' \leftarrow \sim a \quad a' \leftarrow a \quad b' \leftarrow \sim b \quad b' \leftarrow b$$

$$a'' \leftarrow \sim a'' \quad a'' \leftarrow a'' \quad a'' \leftarrow a'' \quad a'' \leftarrow a''$$

$$b'' \leftarrow \sim b'' \quad b'' \leftarrow b'' \quad b'' \leftarrow b'' \quad b'' \leftarrow b''$$

where $\text{dom}(A'_1) = \{a'', b''\}$ and $A''(I) = \{|a'', b''\} \cap I \geq 1$, for all $I \subseteq \mathcal{U}$. The F-stable models of $\text{str}(\Pi_1)$ are the following: $\emptyset \cup X$ and $\{a, c\} \cup X \cup \{a''\}$, where $X = \{a', b'\}$. In fact, for atoms $a', b'$, comments in Example 6 apply. Atom $a''$ instead is forced to have the same truth value of $a$ because of rules of the form (4). Similarly for $b''$ and $b'$.

Note that the additional auxiliary atoms of the form $p''$ in the second rewriting are used to fix the interpretation of ag-
ggregates in program reducts.

Example 9. Consider again program $\Pi_2$ from Example 7, and its rewriting $\text{str}(\Pi_2)$:

$$a \leftarrow A'_2, a', b' \quad a \leftarrow b \quad b \leftarrow a$$

$$a' \leftarrow \sim a \quad a' \leftarrow a \quad b' \leftarrow \sim b \quad b' \leftarrow b$$

$$a'' \leftarrow \sim a'' \quad a'' \leftarrow a'' \quad a'' \leftarrow a'' \quad a'' \leftarrow a''$$

$$b'' \leftarrow \sim b'' \quad b'' \leftarrow b'' \quad b'' \leftarrow b'' \quad b'' \leftarrow b''$$

where $\text{dom}(A'_2) = \{a'', b''\}$ and $A''(I) = \{|a'', b''\} \cap I \neq \emptyset$, for all $I \subseteq \mathcal{U}$. The rewritten program $\text{str}(\Pi_2)$ still contains a non-convex aggregate, namely $A'_2$, which however is such that its interpretation is fixed in program reducts. Hence, any F-reduct of $\text{str}(\Pi_2)$ can be considered as a program without negation and aggregates. For example, the F-reduct of $\text{str}(\Pi_2)$ with respect to $\{a, b, a', b', a'', b''\}$ is the following program $F(\text{str}(\Pi_2), \{a, b, a', b', a'', b''\})$:

$$a \leftarrow A''_2, a', b' \quad a \leftarrow b \quad b \leftarrow a$$

$$a' \leftarrow a \quad b' \leftarrow b \quad a'' \leftarrow b'' \quad b'' \leftarrow b''$$

Note that $a''$ and $b''$ are necessarily true in any model of the above program, which in turn implies truth of $A_2$. ■

Correctness of the two rewritings presented in this section was formally proved by [Alviano and Leone, 2015].

Theorem 1. If $\Pi$ is a program, then

$$GSM(\Pi) \equiv \text{At}(\Pi) \text{ FSM}(\text{rew}(\Pi)) \equiv \text{At}(\Pi) \text{ FSM}(\text{str}(\Pi)).$$

4 Conclusion

G-stable models are a recent proposal for interpreting logic programs with aggregates. A detailed complexity analysis of the main reasoning tasks for this new semantics was reported in [Alviano and Leone, 2015], and briefly recalled in the introduction of this paper, highlighting similarities and differences versus mainstream ASP semantics, here referred to as F-stable models. A practical link between G- and F-
stable models is provided by the rewritings in Section 3: G-stable models of an input program can be obtained by computing F-stable models of a rewritten program, where the size of the rewritten program is linear with respect to the size of the original program. It is interesting to ob-
serve that the second rewriting in Section 3 is such that all atoms occurring in aggregates are defined only by rules of the form $p'' \leftarrow \sim p''$. This fact is sufficient to guarantee that the rewritten programs contain stratified aggregates only, which are handled efficiently by modern ASP solvers. A prototype system for computing G-stable models is thus obtained by means of these rewritings and using existing ASP solvers as back-end. As a final remark, it is interesting to observe that the second rewriting in Section 3 can be combined with the rewritings by [Bomanson and Janhunen, 2013; Bomanson et al., 2014] in order to completely remove GZ-aggregates from ASP programs.
References


