Supportedly Stable Answer Sets for Logic Programs with Generalized Atoms

Mario Alviano[1(✉)] and Wolfgang Faber[2]

[1] University of Calabria, Rende, Italy
alviano@mat.unical.it
[2] University of Huddersfield, Huddersfield, UK
wf@wfaber.com

Abstract. Answer Set Programming (ASP) is logic programming under the stable model or answer set semantics. During the last decade, this paradigm has seen several extensions by generalizing the notion of atom used in these programs. Among these, there are dl-atoms, aggregate atoms, HEX atoms, generalized quantifiers, and abstract constraints. In this paper we refer to these constructs collectively as generalized atoms. The idea common to all of these constructs is that their satisfaction depends on the truth values of a set of (non-generalized) atoms, rather than the truth value of a single (non-generalized) atom. Motivated by several examples, we argue that for some of the more intricate generalized atoms, the previously suggested semantics provide unintuitive results and provide an alternative semantics, which we call supportedly stable or SFLP answer sets. We show that it is equivalent to the major previously proposed semantics for programs with convex generalized atoms, and that it in general admits more intended models than other semantics in the presence of non-convex generalized atoms. We show that the complexity of supportedly stable answer sets is on the second level of the polynomial hierarchy, similar to previous proposals and to answer sets of disjunctive logic programs.

1 Introduction

Answer Set Programming (ASP) is a widely used problem-solving framework based on logic programming under the stable model semantics. The basic language relies on Datalog with negation in rule bodies and possibly disjunction in rule heads. When actually using the language for representing practical knowledge, it became apparent that generalizations of the basic language are necessary for usability. Among the suggested extensions are aggregate atoms (similar to aggregations in database queries) [2–5] and atoms that rely on external truth

The main ideas of this paper were also presented in [1]. Mario Alviano was partly supported by MIUR within project "SI-LAB BA2KNOW – Business Analitycs to Know", by Regione Calabria, POR Calabria FESR 2007-2013, within project "ITravel PLUS" and project "KnowRex", by the National Group for Scientific Computation (GNCS-INDAM), and by Finanziamento Giovani Ricercatori UNICAL.

B. ten Cate and A. Mileo (Eds.): RR 2015, LNCS 9209, pp. 30–44, 2015.
DOI: 10.1007/978-3-319-22002-4_4

valuations [6–9]. These extensions are characterized by the fact that deciding the truth values of the new kinds of atoms depends on the truth values of a set of traditional atoms rather than a single traditional atom. We will refer to such atoms as *generalized atoms*, which cover also several other extensions such as abstract constraints, generalized quantifiers, and HEX atoms.

Concerning semantics for programs containing generalized atoms, there have been several different proposals. All of these appear to coincide for programs that do not contain generalized atoms in recursive definitions. The two main semantics that emerged as standards are the PSP semantics [10–12], and the FLP semantics [13,14] (the latter coinciding with Ferraris stable models [15] for the language considered in this paper). In a recent paper [16] the relationship between these two semantics was analyzed in detail; among other, more intricate results, it was shown that the semantics coincide up to convex generalized atoms. It was already established earlier that each PSP answer set is also an FLP answer set, but not vice versa. So for programs containing non-convex generalized atoms, some FLP answer sets are not PSP answer sets. In particular, there are programs that have FLP answer sets but no PSP answer sets.

In this paper, we argue that the FLP semantics is still too restrictive, and some programs that do not have any FLP answer set should instead have answer sets. In order to illustrate the point, consider a coordination game that is remotely inspired by the prisoners' dilemma. There are two players, each of which has the option to confess or defect. Let us also assume that both players have a fixed strategy already, which however still depends on the choice of the other player as well. In particular, each player will confess exactly if both players choose the same option, that is, if both players confess or both defect. This situation can be represented using two propositional atoms for "the first player confesses" and "the second player confesses," which must be derived true when "both players choose the same option," a composed proposition encoded by a generalized atom. As will be explained later, the FLP semantics does not assign any answer set to a program encoding this scenario, and therefore also the PSP semantics will not assign any answer sets to such a program. We observe that such a program is also incoherent according to a more recent refinement of the FLP semantics [17], called *well-justified FLP*.

We point out that this is peculiar, as the scenario in which both players confess seems like a reasonable one; indeed, even a simple inflationary operator would result in this solution: starting from the empty set, the generalized atom associated with "both players choose the same option" is true; therefore, the atoms associated with "the first player confesses" and "the second player confesses" are derived true on the first application of the operator, which is also its fixpoint.

Looking at the reason why this is not an FLP answer set, we observe that it has two countermodels that prevent it from being an answer set, one in which only the first player confesses, and another one in which only the second player confesses (see Fig. 1). Both of these countermodels are models in the classical sense, but they are weak in the sense that they are not supported, meaning that there is no rule justifying their truth. This is a situation that does not

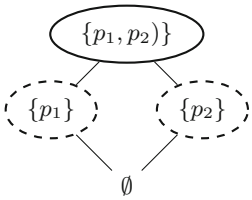

Fig. 1. Interpretations, supported (solid) and unsupported models (dashed) of the prisoners' dilemma example, where p_1 and p_2 are the propositions "the first player confesses" and "the first player confesses," respectively.

occur for programs without generalized atoms, which always have supported countermodels. We argue that one needs to look at supported countermodels, instead of looking at minimal countermodels. It turns out that doing this yields the same results not only for programs without generalized atoms, but also for programs containing convex generalized atoms, which we believe is the reason why this issue has not been noticed earlier.

This paper is first of all a position paper, in which we argue that the existing FLP and PSP semantics are too restrictive on the one hand, and that instead of defining restricting conditions, some conditions need be relaxed. We then proceed to define a new semantics along these lines and call it supportedly stable or SFLP (supportedly FLP) semantics. It provides answer sets for more programs than FLP and PSP, but is shown to be equal on convex programs. Analyzing the computational complexity of the new semantics, we show that it is in the same classes as the FLP and PSP semantics when considering polynomial-time computable generalized atoms. However, it should also be mentioned that the new semantics has its own peculiarities, for instance adding "tautological" rules like $p \leftarrow p$ can change the semantics of the program. These peculiarities suggest that a stronger notion of support is required for obtaining a solid semantics extending FLP.

The remainder of this paper is structured as follows. In Sect. 2, we present the notation and FLP semantics for programs with generalized atoms. After that, in Sect. 3 we analyze issues with the FLP semantics and define the SFLP semantics. In Sect. 4, we prove several useful properties of the new semantics. Finally, in Sect. 6, we discuss our results and provide outlines for future work.

2 Background

In this section we present the notation used in this paper and present the FLP semantics [13,14]. To ease the presentation, we will directly describe a propositional language here. This can be easily extended to the more usual ASP notations of programs involving variables, which stand for their ground versions (that are equivalent to a propositional program).

2.1 Notation

Let $\mathcal{B}$ be a countable set of *propositional atoms*. A *generalized atom* A on $\mathcal{B}$ is a pair (D_A, f_A), where $D_A \subseteq \mathcal{B}$ is the *domain* of A, and f_A is a mapping from 2^{D_A} to Boolean truth values $\{\mathbf{T}, \mathbf{F}\}$. To ease the presentation, we assume that the domain of each generalized atom is a finite set.

Example 1. Let p_1 represent the proposition "the first player confesses," and p_2 represent the proposition "the second player confesses." A generalized atom A representing the composed proposition "both players choose the same option" is such that $D_A = \{p_1, p_2\}$, $f_A(\{\}) = f_A(\{p_1, p_2\}) = \mathbf{T}$, and $f_A(\{p_1\}) = f_A(\{p_2\}) = \mathbf{F}$. ■

A general rule r is of the following form:

$$H(r) \leftarrow B(r) \tag{1}$$

where $H(r)$ is a disjunction $a_1 \vee \cdots \vee a_n$ $(n \geq 0)$ of propositional atoms in $\mathcal{B}$ referred to as the head of r, and $B(r)$ is a generalized atom on $\mathcal{B}$ called the body of r. For convenience, $H(r)$ is sometimes considered a set of propositional atoms. A general program P is a set of general rules. Let $At(P)$ denote the set of propositional atoms occurring in P.

It should be noted that this is a very abstract notation, aiming to be general enough to encompass many concrete languages. Languages adopted in practical systems will feature concrete syntax in place of generalized atoms, for example aggregate atoms or dl-atoms. In the sequel, we will at times also use more concrete notation in examples to ease reading.

2.2 FLP Semantics

An *interpretation* I is a subset of $\mathcal{B}$. I is a *model* for a generalized atom A, denoted $I \models A$, if $f_A(I \cap D_A) = \mathbf{T}$. Otherwise, if $f_A(I \cap D_A) = \mathbf{F}$, I is not a model of A, denoted $I \not\models A$. I is a model of a rule r of the form (1), denoted $I \models r$, if $H(r) \cap I \neq \emptyset$ whenever $I \models B(r)$. I is a model of a program P, denoted $I \models P$, if $I \models r$ for every rule $r \in P$.

Note that the fact that rule bodies are forced to be a single generalized atom is not really a limitation, and will ease the presentation of the results in the paper. In fact, a single generalized atom is sufficient for modeling conjunctions, default negation, aggregates and similar constructs.

Example 2. A conjunction $p_1 \wedge \cdots \wedge p_n$ of $n \geq 1$ propositional atoms is equivalently represented by a generalized atom A such that $D_A = \{p_1, \ldots, p_n\}$, and $f_A(B) = \mathbf{T}$ if and only if $B = \{p_1, \ldots, p_n\}$.

A conjunction $p_1, \ldots, p_m, \sim p_{m+1}, \ldots, \sim p_n$ of literals, where $n \geq m \geq 0$, $p_1, \ldots, p_n$ are propositional atoms and $\sim$ denotes *negation as failure*, is equivalently represented by a generalized atom A such that $D_A = \{p_1, \ldots, p_n\}$, and $f_A(B) = \mathbf{T}$ if and only if $\{p_1, \ldots, p_m\} \subseteq B$ and $B \cap \{p_{m+1}, \ldots, p_n\} = \emptyset$.

An aggregate $COUNT(\{p_1, \ldots, p_n\}) \neq k$, where $n \geq k \geq 0$, and $p_1, \ldots, p_n$ are propositional atoms, is equivalently represented by a generalized atom A such that $D_A = \{p_1, \ldots, p_n\}$, and $f_A(B) = \mathbf{T}$ if and only if $|B \cap D_A| \neq k$. ■

In the following, when convenient, we will represent generalized atoms as conjunctions of literals or aggregate atoms. Subsets of $\mathcal{B}$ mapped to true by such generalized atoms will be those satisfying the associated conjunction.

Example 3. Consider the following rules:

$$r_1 : \ a \leftarrow COUNT(\{a, b\}) \neq 1 \qquad r_2 : \ b \leftarrow COUNT(\{a, b\}) \neq 1$$

The following are general programs that will be used for illustrating the differences between the semantics considered in this paper:

$$\begin{array}{ll} P_1 := \{r_1; r_2\} & P_4 := \{r_1; r_2; a \vee b \leftarrow\} \\ P_2 := \{r_1; r_2; a \leftarrow b; b \leftarrow a\} & P_5 := \{r_1; r_2; a \leftarrow\sim b\} \\ P_3 := \{r_1; r_2; \leftarrow\sim a; \leftarrow\sim b\} & \end{array}$$

Note that if a and b are replaced by p_1 and p_2, the aggregate $COUNT(\{a, b\}) \neq 1$ is equivalent to the generalized atom A from Example 1, and therefore program P_1 encodes the coordination game depicted in the introduction. ■

Generalized atoms can be partitioned into two classes, referred to as *convex* and *non-convex*, according to the following definition: A generalized atom A is convex if for all triples I, J, K of interpretations such that $I \subset J \subset K$, $I \models A$ and $K \models A$ implies $J \models A$. A convex program is a general program whose rules have convex bodies. Note that convex generalized atoms are closed under conjunction, but not under disjunction or complementation. In more detail, the conjunction of two generalized atoms A, A', denoted $A \wedge A'$, is such that $D_{A \wedge A'} = D_A \cup D_{A'}$, and for all $I \subseteq D_{A \wedge A'}$, $f_{A \wedge A'}(I) = f_A(I \cap D_A) \wedge f_{A'}(I \cap D_{A'})$. The disjunction $D_{A \vee A'}$ is defined similarly, and the complementation $\overline{A}$ of A is such that $D_{\overline{A}} = D_A$, and for all $I \subseteq D_A$, $f_{\overline{A}}(I) = \neg f_A(I)$. To show that convex generalized atoms are not closed under disjunction and complementation, an example is sufficient. Let A, A' be such that $D_A = D_{A'} = \{a, b\}$, $f_A(I) = \mathbf{T}$ if and only if $I = \emptyset$, and $f_{A'}(I) = \mathbf{T}$ if and only if $I = \{a, b\}$. Hence, A, A' are convex, but $A \vee A'$ is not. However, its complement $\overline{A \vee A'}$ is convex because true only for $\{a\}$ and $\{b\}$. Closure with respect to conjunction is proved by the following claim.

Lemma 1. *The conjunction $A \wedge A'$ of two convex generalized atoms is a convex generalized atom.*

Proof. Let $I \subset J \subset K$ be such that $I \models A \wedge A'$ and $K \models A \wedge A'$. Hence, $I \models A$, $K \models A$, $I \models A'$, and $K \models A'$ by definition of $A \wedge A'$. Since A and A' are convex, we have $J \models A$ and $J \models A'$, which in turn imply $J \models A \wedge A'$. □

We now describe a reduct-based semantics, usually referred to as FLP, which has been introduced and analyzed in [13,14].

Definition 1 (FLP Reduct). *The FLP reduct P^I of a program P with respect to I is defined as the set $\{r \in P \mid I \models B(r)\}$.*

Definition 2 (FLP Answer Sets). *I is an FLP answer set of P if $I \models P$ and for each $J \subset I$ it holds that $J \not\models P^I$. Let $FLP(P)$ denote the set of FLP answer sets of P.*

Example 4. Consider the programs from Example 3:

- The models of P_1 are $\{a\}$, $\{b\}$ and $\{a, b\}$, none of which is an FLP answer set. Indeed, $P_1^{\{a\}} = P_1^{\{b\}} = \emptyset$, which have the trivial model $\emptyset$, which is of course a subset of $\{a\}$ and $\{b\}$. On the other hand $P_1^{\{a,b\}} = P_1$, and so $\{a\} \models P_1^{\{a,b\}}$, where $\{a\} \subset \{a, b\}$. We will discuss in the next section why this is a questionable situation.
- Concerning P_2, it has one model, namely $\{a, b\}$, which is also its unique FLP answer set. Indeed, $P_2^{\{a,b\}} = P_2$, and hence the only model of $P_2^{\{a,b\}}$ is $\{a, b\}$.
- Interpretation $\{a, b\}$ is also the unique model of program P_3, which however has no FLP answer sets. Here, $P_3^{\{a,b\}} = P_1$, hence similar to P_1, $\{a\} \models P_3^{\{a,b\}}$ and $\{a\} \subset \{a, b\}$.
- P_4 instead has two FLP answer sets, namely $\{a\}$ and $\{b\}$, and a further model $\{a, b\}$. In this case, $P_4^{\{a\}} = \{a \vee b \leftarrow\}$, and no proper subset of $\{a\}$ satisfies it. Also $P_4^{\{b\}} = \{a \vee b \leftarrow\}$, and no proper subset of $\{b\}$ satisfies it. Instead, for $\{a, b\}$, we have $P_4^{\{a,b\}} = P_4$, and hence $\{a\} \models P_4^{\{a,b\}}$ and $\{a\} \subset \{a, b\}$.
- Finally, P_5 has tree models, $\{a\}$, $\{b\}$ and $\{a, b\}$, but only one answer set, namely $\{a\}$. In fact, $P_5^{\{a\}} = \{a \leftarrow\sim b\}$ and $\emptyset$ is not a model of the reduct. On the other hand, $\emptyset$ is a model of $P_5^{\{b\}} = \emptyset$, and $\{a\}$ is a model of $P_5^{\{a,b\}} = P_1$.

Models and FLP answer sets of these programs are summarized in Table 1. ■

3 SFLP Semantics

As noted in the introduction, the fact that P_1 has no FLP answer sets is striking. If we first assume that both a and b are false (interpretation $\emptyset$), and then apply a generalization of the well-known one-step derivability operator, we obtain truth of both a and b (interpretation $\{a, b\}$). Applying this operator once more again yields the same interpretation, a fix-point. Interpretation $\{a, b\}$ is also a supported model, that is, for all true atoms there exists a rule in which this atom is the only true head atom, and in which the body is true.

It is instructive to examine why this seemingly robust model is not an FLP answer set. Its reduct is equal to the original program, $P_1^{\{a,b\}} = P_1$. There are therefore two models of P_1, $\{a\}$ and $\{b\}$, that are subsets of $\{a, b\}$ and therefore inhibit $\{a, b\}$ from being an FLP answer set. The problem is that, contrary to $\{a, b\}$, these two models are rather weak, in the sense that they are not supported. Indeed, when considering $\{a\}$, there is no rule in P_1 such that a is the only true

Table 1. (Supported) models and (S)FLP answer sets of programs in Example 3, where A is the generalized atom $COUNT(\{a, b\}) \neq 1$.

	Rules	Models	FLP	Supported Models	SFLP
P_1	$a \leftarrow A \quad b \leftarrow A$	$\{a\}, \{b\}, \{a, b\}$	—	$\{a, b\}$	$\{a, b\}$
P_2	$a \leftarrow A \quad b \leftarrow A$ $a \leftarrow b \quad b \leftarrow a$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$	$\{a, b\}$
P_3	$a \leftarrow A \quad b \leftarrow A$ $\leftarrow\sim a \quad \leftarrow\sim b$	$\{a, b\}$	—	$\{a, b\}$	$\{a, b\}$
P_4	$a \leftarrow A \quad b \leftarrow A$ $a \vee b \leftarrow$	$\{a\}, \{b\}, \{a, b\}$	$\{a\}, \{b\}$	$\{a\}, \{b\}, \{a, b\}$	$\{a\}, \{b\}$
P_5	$a \leftarrow A \quad b \leftarrow A$ $a \leftarrow\sim b$	$\{a\}, \{b\}, \{a, b\}$	$\{a\}$	$\{a\}, \{a, b\}$	$\{a\}, \{a, b\}$

atom in the rule head and the body is true in $\{a\}$: The only available rule with a in the head has a false body. The situation for $\{b\}$ is symmetric.

It is somewhat counter-intuitive that a model like $\{a, b\}$ should be inhibited by two weak models like $\{a\}$ and $\{b\}$. Indeed, this is a situation that normally does not occur in ASP. For programs that do not contain generalized atoms, whenever one finds a $J \subset I$ such that $J \models P^I$ there is for sure also a K such that $J \subseteq K \subset I$, $K \models P^I$ and K is supported. Indeed, we will show in Sect. 4 that this is the case also for programs containing only convex generalized atoms. Our feeling is that since such a situation does not happen for a very wide set of programs, it has been overlooked so far.

We will now attempt to repair this kind of anomaly by stipulating that one should only consider supported models for finding inhibitors of answer sets. In other words, one does not need to worry about unsupported models of the reduct, even if they are subsets of the candidate. Let us first define supported models explicitly.

Definition 3 (Supportedness). *A model I of a program P is supported if for each $a \in I$ there is a rule $r \in P$ such that $I \cap H(r) = \{a\}$ and $I \models B(r)$. In this case we will write $I \models_s P$.*

Example 5. Continuing Example 4, programs P_1, P_2, and P_3 have one supported model, namely $\{a, b\}$. The model $\{a\}$ of P_1 is not supported because the body of the the rule with a in the head has a false body with respect to $\{a\}$. For a symmetric argument, model $\{b\}$ of P_1 is not supported either. The supported models of P_4, instead, are $\{a\}$, $\{b\}$, and $\{a, b\}$, so all models of the program are supported. Note that both models $\{a\}$ and $\{b\}$ have the disjunctive rule as the only supporting rule for the respective single true atom, while for $\{a, b\}$, the two rules with generalized atoms serve as supporting rules for a and b. Finally, the supported models of P_5 are $\{a\}$ and $\{a, b\}$. Supported models of these programs are summarized in Table 1. ■

We are now ready to formally introduce the new semantics. In this paper we will normally refer to it as SFLP answer sets or SFLP semantics, but also call it *supportedly stable models* occasionally.

Definition 4 (SFLP Answer Sets). *I is an SFLP answer set of P if $I \models_s P$ and for each $J \subset I$ it holds that $J \not\models_s P^I$. Let $SFLP(P)$ denote the set of SFLP answer sets of P.*

Example 6. Consider again the programs from Example 3.

- Recall that P_1 has only one supported model, namely $\{a, b\}$, and $P_1^{\{a,b\}} = P_1$, but $\emptyset \not\models_s P_1^{\{a,b\}}$, $\{a\} \not\models_s P_1^{\{a,b\}}$, and $\{b\} \not\models_s P_1^{\{a,b\}}$, therefore no proper subset of $\{a, b\}$ is a supported model. Hence, it is an SFLP answer set.
- Concerning P_2, it has one model, namely $\{a, b\}$, which is supported and also its unique SFLP answer set. Indeed, recall that $P_2^{\{a,b\}} = P_2$, and hence no proper subset of $\{a, b\}$ can be a model (let alone a supported model) of $P_2^{\{a,b\}}$.
- Interpretation $\{a, b\}$ is the unique model of program P_3, which is supported and also its SFLP answer set. In fact, $P_3^{\{a,b\}} = P_1$.
- P_4 has two SFLP answer sets, namely $\{a\}$ and $\{b\}$. In this case, recall $P_4^{\{a\}} = \{a \vee b \leftarrow\}$, and no proper subset of $\{a\}$ satisfies it. Also $P_4^{\{b\}} = \{a \vee b \leftarrow\}$, and no proper subset of $\{b\}$ satisfies it. Instead, for $\{a, b\}$, we have $P_4^{\{a,b\}} = P_4$, hence since $\{a\} \models_s P_4^{\{a,b\}}$, and $\{b\} \models_s P_4^{\{a,b\}}$, we obtain that $\{a, b\}$ is not an SFLP answer set.
- Finally, P_5 has two SFLP answer sets, namely $\{a\}$ and $\{a, b\}$. In fact, $P_5^{\{a\}} = \{a \leftarrow\sim b\}$ and $P_5^{\{a,b\}} = P_1$.

The programs, models, FLP answer sets, supported models, and SFLP answer sets are summarized in Table 1. ■

An alternative, useful characterization of SFLP answer sets can be given in terms of Clark's completion [18]. In fact, it is well-known that supported models of a program are precisely the models of its completion. We define this notion in a somewhat non-standard way, making use of the concept of generalized atom. We first define the completion of a propositional atom p with respect to a general program P as a generalized atom encoding the supportedness condition for p.

Definition 5. *The completion of a propositional atom $p \in \mathcal{B}$ with respect to a general program P is a generalized atom A such that $D_A = At(P)$, and for all $I \subseteq D_A$, $f_A(I) = \mathbf{T}$ if and only if $p \in I$ and there is no rule $r \in P$ for which $I \models B(r)$ and $I \cap H(r) = \{a\}$. Let $comp(p, P)$ denote the completion of p with respect to P.*

These generalized atoms are then used to effectively define a program whose models are the supported model of P.

Definition 6. *The completion of a general program P is a general program $comp(P)$ extending P with a rule $\leftarrow comp(p, P)$ for each propositional atom $p \in At(P)$.*

Example 7. Consider again the programs from Example 3.

- Program $comp(P_1)$ extends P_1 with the following rules:

$$\leftarrow a \wedge COUNT(\{a,b\}) = 1 \leftarrow b \wedge COUNT(\{a,b\}) = 1$$

- Program $comp(P_2)$ extends P_2 with the following rules:

$$\leftarrow a \wedge COUNT(\{a,b\}) = 1 \wedge \sim b \leftarrow b \wedge COUNT(\{a,b\}) = 1 \wedge \sim a$$

- Program $comp(P_3)$ is equal to $comp(P_1)$, and program $comp(P_4)$ extends P_4 with the following rules:

$$\leftarrow a \wedge COUNT(\{a,b\}) = 1 \wedge b \leftarrow b \wedge COUNT(\{a,b\}) = 1 \wedge a$$

- Program $comp(P_5)$ instead extends P_5 with the following rules:

$$\leftarrow a \wedge COUNT(\{a,b\}) = 1 \wedge b \leftarrow b \wedge COUNT(\{a,b\}) = 1$$

Note that the only model of $comp(P_1)$, $comp(P_2)$, and $comp(P_3)$ is $\{a,b\}$. As for $comp(P_4)$, and $comp(P_5)$, their models are $\{a\}$, $\{b\}$, and $\{a,b\}$. ■

Proposition 1. *Let P be a general program, and I be an interpretation. Then, $I \models_s P$ if and only if $I \models comp(P)$.*

This characterization, which follows directly from [18], provides us with a means for implementation that relies only on model checks, rather than supportedness checks.

Proposition 2. *Let P be a general program, and I be an interpretation. Then, I is an SFLP answer set of P if $I \models comp(P)$ and for each $J \subset I$ it holds that $J \not\models comp(P^I)$.*

4 Properties

The new semantics has a number of interesting properties that we report in this section. First of all, it is an extension of the FLP semantics, in the sense that each FLP answer set is also an SFLP answer set.

Theorem 1. *Let P be a general program. Then, $FLP(P) \subseteq SFLP(P)$.*

Proof. Let I be an FLP answer set of P. Hence, each $J \subset I$ is such that $J \not\models P^I$. Thus, we can conclude that $J \not\models_s P^I$ for any $J \subset I$. Therefore, I is a SFLP answer set of P. □

The inclusion is strict in general. In fact, P_1 is a simple program for which the two semantics disagree (see Examples 3–6 and Table 1). On the other hand, the two semantics are equivalent for a large class of programs, as shown below.

Theorem 2. *If P is a convex program then $FLP(P) = SFLP(P)$.*

Proof. $FLP(P) \subseteq SFLP(P)$ holds by Theorem 1. For the other direction, consider an interpretation I not being an FLP answer set of P. Hence, there is $J \subset I$ such that $J \models P^I$. We also assume that J is a subset-minimal model of P^I, that is, there is no $K \subset J$ such that $K \models P^I$. We shall show that $J \models_s P^I$. To this end, suppose by contradiction that there is $p \in J$ such that for each $r \in P^I$ either $J \not\models B(r)$ or $J \cap H(r) \neq \{p\}$. Consider $J \setminus \{p\}$ and a rule $r \in P^I$ such that $J \setminus \{p\} \models B(r)$. Since $r \in P^I$, $I \models B(r)$, and thus $J \models B(r)$ because $B(r)$ is convex. Therefore, $J \cap H(r) \neq \{p\}$. Moreover, $J \cap H(r) \neq \emptyset$ because $J \models P^I$ by assumption. Hence, $(J \setminus \{p\}) \cap H(r) \neq \emptyset$, and therefore $J \setminus \{p\} \models P^I$. This contradicts the assumption that J is a subset-minimal model of P^I. □

We will now focus on computational complexity. We consider here the problem of determining whether an SFLP answer set exists. We note that the only difference to the FLP semantics is in the stability check. For FLP, subsets need to be checked for being a model. For SFLP, instead, subsets need to be checked for being a supported model. Intuitively, one would not expect that this difference can account for a complexity jump, which is confirmed by the next result.

Theorem 3. *Let P be a general program whose generalized atoms are polynomial-time computable functions. Checking whether $SFLP(P) \neq \emptyset$ is in Σ_2^P in general; it is Σ_2^P-hard already in the disjunction-free case if at least one form of non-convex generalized atom is permitted. The problem is NP-complete if P is disjunction-free and convex.*

Proof. For the membership in Σ_2^P one can guess an interpretation I and check that there is no $J \subset I$ such that $J \models_s P$. The check can be performed by a coNP oracle. To prove Σ_2^P-hardness we note that extending a general program P by rules $p \leftarrow p$ for every $p \in At(P)$ is enough to guarantee that all models of any reduct of P are supported. We thus refer to the construction and proof by [16]. If P is disjunction-free and convex then $SFLP(P) = FLP(P)$ by Theorem 2. Hence, NP-completeness follows from results in [19]. □

We would like to point out that the above proof also illustrates a peculiar feature of SFLP answer sets, which it shares with the supported model semantics: the semantics is sensitive to tautological rules like $p \leftarrow p$, as their addition can turn non-SFLP answer sets into SFLP answer sets.

5 Discussion

Existing semantics for logic programs with generalized atoms do not yield intended models under particular conditions. Specifically, the lack of intended models seems to be ascribable to unsupported countermodels. In the FLP semantics, the anomaly arises in connection with non-convex generalized atoms. As the example in the introduction shows, the anomaly affects other semantics as well. For example, it affects also description logic programs interpreted according to the recent well-justified FLP semantics [17], which selects among FLP answer

sets. Former semantics for description logic programs, referred to as strong and weak stable models [7,8], behave differently. Without going into details, we use the prisoners' dilemma example from the introduction. Recall that there are two prisoners who will confess if (and only if) both players choose the same option. This situation can be represented using the following description logic program:

$$a(0) \leftarrow DL[A \uplus a, B \uplus b, A \uplus a, B \uplus b; Q](0)$$
$$b(0) \leftarrow DL[A \uplus a, B \uplus b, A \uplus a, B \uplus b; Q](0)$$

with the following ontology:

$$Q \equiv (A \sqcap B) \sqcup (\neg A \sqcap \neg B).$$

Here, $a(0)$ means that the first player confesses and $b(0)$ means that the second player confesses. The DL atoms transfer the extensions of a and b to A and B of the ontology, making sure that constants that are not in the extension of a will populate $\neg A$ (likewise for b and $\neg B$); this is achieved by means of the $\uplus$ and $\uplus$ operators. The ontology defines Q, which represents "good" situations (both confess or both defect). The rule thus states that $a(0)$ and $b(0)$ should hold in a "good" situation. An alternative encoding using $\uplus$ instead of $\uplus$ is given by the following program:

$$a(0) \leftarrow DL[A \uplus a, B \uplus b, C \uplus a, D \uplus b; Q](0)$$
$$b(0) \leftarrow DL[A \uplus a, B \uplus b, C \uplus a, D \uplus b; Q](0)$$

with the following ontology:

$$Q \equiv (\neg A \sqcap \neg B) \sqcup (\neg C \sqcap \neg D).$$

where operator $\uplus$ increases $\neg C$ and $\neg D$ with the extension of a and b, respectively. These programs correspond to P_1 in Example 3, which just uses different notation. Interpretation $\{a(0), b(0)\}$ is both the only strong and weak answer set of the above programs. However, these semantics suffer from other problems. For example, if a fact $a(0)$ is added to the above programs, $\{a(0), b(0)\}$ would still be both a strong and weak answer set, along with $\{a(0)\}$, which we consider to be quite unintuitive.

Another semantics that selects among FLP answer sets, and is therefore affected by the anomaly on unsupported countermodels, is PSP [10–12]. Analyzing Fig. 2, we can better understand a common issue for PSP and well-justified FLP: The adopted operator for fixpoint computation cannot "jump" over gaps in the lattice of interpretations when establishing the truth value of a generalized atom. Examining the figure, the fact that $\{a(0)\}$ and $\{b(0)\}$ both do not satisfy the generalized atom prevents these operators from deriving $\{a(0), b(0)\}$. Yet these two interpretations are not relevant for the example in the introduction.

We observe that other interesting semantics, such as the one by [20], are also affected by the anomaly on unsupported models. In particular, the semantics by [20] is presented for programs consisting of arbitrary set of propositional

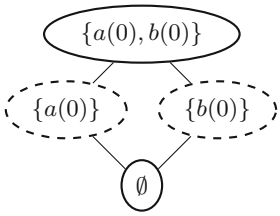

Fig. 2. Satisfying (solid) and unsatisfying (dashed) interpretations of the description logic programs in Sect. 5 encoding the prisoners' dilemma.

formulas, and it is based on a reduct in which false subformulas are replaced by $\bot$. Answer sets are then defined as interpretations being subset-minimal models of their reducts. For the notation used in this paper, when rewriting generalized atoms to an equivalent formula, the semantics by [20] coincides with FLP, which immediately shows the anomaly. In [20] there is also a method for rewriting aggregates, however $COUNT(\{a, b\}) \neq 1$ is not explicitly supported, but should be rewritten to $\neg(COUNT(\{a, b\}) = 1)$. Doing this, one can observe that for P_1, P_2, P_3, and P_5 the semantics of [20] behaves like SFLP (cf. Table 1), while for P_4 the semantics of [20] additionally has the answer set $\{a, b\}$, which is not a supported minimal model of the FLP reduct. P_4 therefore shows that the two semantics do not coincide, even if generalized atoms are interpreted as their negated complements, and the precise relationship is left for further study. However, we also believe that rewriting a generalized atom into its negated complement is not always natural, and we are also convinced that there should be a semantic difference between a generalized atom and its negated complement.

6 Conclusion

In this paper, we have first studied conditions under which existing semantics for logic programs with generalized atoms do not yield intended models, while they arguably should. Analyzing the reasons, we argue that this seems to be connected to a lack of support in countermodels. Motivated by this situation, we have then defined a new semantics for programs with generalized atoms, called supportedly stable models, supportedly FLP, or SFLP semantics. The new definition overcomes the anomaly on unsupported countermodels, and provides a robust semantics for programs with generalized atoms. We show several properties of this new semantics. For example, it coincides with the FLP semantics (and thus also the PSP semantics) on convex programs, and therefore also on standard programs. Furthermore, the complexity of common reasoning tasks is equal to the respective tasks using the FLP semantics. We also provide a characterization of the new semantics by a Clark-inspired completion.

Concerning future work, implementing a reasoner supporting the new semantics would be of interest, for example by compiling the new semantics in FLP, so to use current ASP solvers such as DLV [21], CMODELS [22], CLASP [23], and

WASP [24,25]. An application area would be systems that loosely couple OWL ontologies with rule bases, for instance by means of HEX programs. As we have shown earlier, HEX atoms interfacing to ontologies will in general not be convex, and therefore using them in recursive definitions falls into our framework, where the FLP and SFLP semantics differ. In fact, the solver `dlvhex`[1] does not produce any answer for the program and ontology provided in Sect. 5. We also believe that it would be important to collect example programs that contain non-convex generalized atoms in recursive definitions. We have experimented with a few simple domains stemming from game theory (as outlined in the introduction), but we are not aware of many other attempts. Our intuition is that such programs would be written in several domains that describe features with feedback loops, which applies to many so-called complex systems. Also computing or checking properties of neural networks might be a possible application in this area. Another area of future work arises from the fact that rules like $p \leftarrow p$ are not irrelevant for the SFLP semantics. To us, it is not completely clear how much of a drawback this really is. However, we intend to study this along two avenues. The first is trying to characterize programs for which this phenomenon occurs, the second is to propose variants of the SFLP semantics that do not exhibit it. Concerning the latter, one idea would be to require models or countermodels to have level mappings as defined in [17].

References

1. Alviano, M., Faber, W.: Semantics and compilation of answer set programming with generalized atoms. In: Konieczny, S., Tompits, H. (eds.) 15th International Workshop on Nonmonotonic Reasoning (NMR 2014), Number 1843–14-01 in INFSYS Research Reports, Institut für Informationssysteme, pp. 214–222, July 2014
2. Niemelä, I., Simons, P., Soininen, T.: Stable model semantics of weight constraint rules. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) LPNMR 1999. LNCS, vol. 1730, pp. 317–331. Springer, Heidelberg (1999)
3. Niemelä, I., Simons, P.: Extending the smodels system with cardinality and weight constraints. In: Minker, J. (ed.) Logic-Based Artificial Intelligence, pp. 491–521. Kluwer Academic Publishers, Dordrecht (2000)
4. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate functions in disjunctive logic programming: semantics, complexity, and implementation in DLV. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI) 2003. Morgan Kaufmann Publishers, Acapulco, Mexico, pp. 847–852, August 2003
5. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and implementation of aggregate functions in the DLV system. Theory Pract. Logic Program. **8**(5–6), 545–580 (2008)
6. Calimeri, F., Cozza, S., Ianni, G.: External sources of knowledge and value invention in logic programming. Ann. Math. Artif. Intell. **50**(3–4), 333–361 (2007)

[1] http://www.kr.tuwien.ac.at/research/systems/dlvhex/.

7. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, pp. 141–151 (2004). Extended Report RR-1843-03-13, Institut für Informationssysteme, TU Wien, 2003
8. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. **172**(12–13), 1495–1539 (2008)
9. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer set programming. In: International Joint Conference on Artificial Intelligence (IJCAI) 2005, pp. 90–96, Edinburgh, UK, August 2005
10. Pelov, N.: Semantics of logic programs with aggregates. Ph.D. thesis, Katholieke Universiteit Leuven, Leuven, Belgium, April 2004
11. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and Stable semantics of logic programs with aggregates. Theory Pract. Logic Program. **7**(3), 301–353 (2007)
12. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in ASP. Theory Pract. Logic Program. **7**, 355–375 (2007)
13. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: semantics and complexity. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 200–212. Springer, Heidelberg (2004)
14. Faber, W., Leone, N., Pfeifer, G.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. **175**(1), 278–298 (2011). Special Issue: John McCarthy's Legacy
15. Ferraris, P.: Logic programs with propositional connectives and aggregates. ACM Trans. Comput. Log. **12**(4), 25 (2011)
16. Alviano, M., Faber, W.: The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS, vol. 8148, pp. 67–72. Springer, Heidelberg (2013)
17. Shen, Y., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., Deng, J.: FLP answer set semantics without circular justifications for general logic programs. Artif. Intell. **213**, 1–41 (2014)
18. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press, New York (1978)
19. Liu, L., Truszczyński, M.: Properties and applications of programs with monotone and convex constraints. J. Art. Intell. Res. **27**, 299–334 (2006)
20. Ferraris, P.: Answer sets for propositional theories. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 119–131. Springer, Heidelberg (2005)
21. Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., Terracina, G.: The disjunctive datalog system DLV. In: de Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.) Datalog 2010. LNCS, vol. 6702, pp. 282–301. Springer, Heidelberg (2011)
22. Lierler, Y., Maratea, M.: Cmodels-2: sat-based answer set solver enhanced to non-tight programs. In: Lifschitz, V., Niemelä, I. (eds.) LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 346–350. Springer, Heidelberg (2003)
23. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187**, 52–89 (2012)

24. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: a native ASP solver based on constraint learning. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS, vol. 8148, pp. 54–66. Springer, Heidelberg (2013)
25. Alviano, M., Dodaro, C., Ricca, F.: Anytime computation of cautious consequences in answer set programming. TPLP **14**(4–5), 755–770 (2014)