Boolean Functions with Ordered Domains in Answer Set Programming

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

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

Hannes Strass
Leipzig University, Germany
strass@informatik.uni-leipzig.de

Abstract

Boolean functions in Answer Set Programming have proven a useful modelling tool. They are usually specified by means of aggregates or external atoms. A crucial step in computing answer sets for logic programs containing Boolean functions is verifying whether partial interpretations satisfy a Boolean function *for all possible values* of its undefined atoms. In this paper, we develop a new methodology for showing when such checks can be done in deterministic polynomial time. This provides a unifying view on all currently known polynomial-time decidability results, and furthermore identifies promising new classes that go well beyond the state of the art. Our main technique consists of using an ordering on the atoms to significantly reduce the necessary number of model checks. For many standard aggregates, we show how this ordering can be automatically obtained.

Introduction

Answer set programming (ASP) is a declarative language for knowledge representation and reasoning (Brewka, Eiter, and Truszczynski 2011). ASP programs are interpreted according to the stable model semantics (Gelfond and Lifschitz 1988; 1991), and several definitions were proposed for extensions of the basic language. A particularly useful construct of ASP are aggregate functions (Simons, Niemelä, and Soininen 2002; Liu et al. 2010; Bartholomew, Lee, and Meng 2011; Pelov, Denecker, and Bruynooghe 2007; Son and Pontelli 2007; Shen et al. 2014; Faber, Pfeifer, and Leone 2011; Ferraris 2011; Alviano et al. 2011; Gelfond and Zhang 2014), which allow for expressing properties on sets of atoms declaratively and in a space-efficient way. For example, aggregates are widely used to enforce *functional dependencies*, where a rule of the form

$$\bot \leftarrow \texttt{node}(X),\ \text{COUNT}(\{\, C \mid \texttt{hasColour}(X, C)\}) \neq 1$$

in a graph-colouring problem asserts that the colour of a node is a functional property. On the other hand, aggregates often make the evaluation of programs harder. In fact, the three-valued evaluation of an aggregate, that is, its evaluation with respect to a partial interpretation I, depends in general on the evaluation of the aggregate with respect to exponentially many totalisations of I.

It is important to observe that many of the semantics proposed for interpreting ASP programs with aggregates are not limited to common aggregation functions such as COUNT, SUM, and AVG, but are instead defined for Boolean functions in general (Liu and Truszczynski 2006; Alviano and Faber 2015). In fact, from a semantic viewpoint an aggregate is seen as a black box whose relevant property is the induced (partial) Boolean function mapping (partial) interpretations to Boolean truth values. For example, $\text{SUM}(\{1 : p, -1 : q\}) \geq 0$ maps to true any (partial) interpretation assigning true to p or false to q, and maps to false any (partial) interpretation assigning false to p and true to q.

It is thanks to this association with Boolean functions that the several semantics for ASP programs with aggregates are defined clearly and uniformly: stable models are defined for programs with Boolean functions in general, and any aggregation function can be added to the language by specifying the associated Boolean function. Using Boolean functions also makes the same definitions of semantics applicable to similar language extensions, such as external or HEX atoms (Eiter et al. 2014).

Another advantage of this generality is the identification of semantic classes of programs with benign computational properties. For example, programs with *monotone* and *convex* (Liu and Truszczynski 2006) Boolean functions are associated with lower complexity classes in many cases (Faber, Pfeifer, and Leone 2011). Many other tractability results were proven for programs with non-convex aggregates of specific forms (Pelov 2004; Son and Pontelli 2007), providing ad-hoc proofs for each considered case. These results hold for stable models as defined by Pelov, Denecker, and Bruynooghe (2007) and Son and Pontelli (2007), for which tractability of the three-valued evaluation of aggregates implies tractability of the stability check.

Boolean functions were also considered in a related knowledge representation formalism called *abstract dialectical frameworks* (ADFs, Brewka and Woltran; Brewka et al., 2010; 2013). There, argumentation scenarios are modelled in terms of arguments and possible relationships between arguments. In the class of *bipolar* ADFs, relationships between arguments are restricted to supports and attacks, which decreases the computational complexity by one level in the polynomial hierarchy (Strass and Wallner 2015). It is interesting to observe that, under some syntactic restric-

tions, the notion of stable model for ADFs by Brewka et al. (2013) coincides with the definition of stable model for ASP programs by Pelov, Denecker, and Bruynooghe (2007) and Son and Pontelli (2007), as was recently observed (Alviano and Faber 2015). We use a similar observation to define a new class of Boolean functions (and thus a new class of aggregates in ASP). In fact, introducing the class of *bipolar Boolean functions* is natural, and tractability is easily obtained by transferring the complexity results of Strass and Wallner (2015). It was unexpected, however, that many tractable cases proved by Pelov (2004) and Son and Pontelli (2007) are actually bipolar, and therefore could be obtained uniformly and in a straightforward way by our results.

While a significant number of standard aggregates leads to bipolar Boolean functions, there are still cases that are known to be polytime-checkable but are not bipolar. For example, $\text{COUNT}(\{p,q\}) = 1$ is convex but not bipolar, and $\text{COUNT}(\{p,q\}) \neq 1$ is polytime-checkable (Son and Pontelli 2007) but neither convex nor bipolar.

As the main contribution of this paper, we introduce a new class of Boolean functions whose three-valued evaluation can be done in polynomial time; we call them *atom-orderable Boolean functions*. Our results are also transferable to ADFs with the stable model semantics defined by Brewka et al. (2013) thanks to the results of Alviano and Faber (2015). In particular, this identifies a larger tractable class of ADFs under this semantics than previously known.

Preliminaries

Let A be a finite set of (propositional) *atoms*. Boolean functions map sets of atoms to Boolean truth values. For convenience, we will usually represent a Boolean function as the set of sets mapped to true. Hence, a *Boolean function* is a set $C \subseteq 2^A$. In addition to this abstract representation of Boolean functions, we will also use common notations for denoting aggregates. Formally, let $a_1, \ldots, a_m \in A$ be atoms and $w_1, \ldots, w_m \in \mathbb{R}$ be real numbers ($m \geq 0$). A *weighted atom set over A* is of the form $S = \{w_1 : a_1, \ldots, w_m : a_m\}$. For such a set, we denote $A(S) = \{a_1, \ldots, a_m\}$. Using a comparison $\circ \in \{<, \leq, =, \neq, \geq, >\}$ and a value $v \in \mathbb{R}$, the following expressions represent Boolean functions:

$$\text{SUM}(S) \circ v \mathrel{\hat{=}} \left\{ M \subseteq 2^{A(S)} \;\middle|\; \left(\sum_{a_i \in M} w_i \right) \circ v \right\}$$

$$\text{PROD}(S) \circ v \mathrel{\hat{=}} \left\{ M \subseteq 2^{A(S)} \;\middle|\; \left(\prod_{a_i \in M} w_i \right) \circ v \right\}$$

$$\text{COUNT}(S) \circ v \mathrel{\hat{=}} \left\{ M \subseteq 2^{A(S)} \;\middle|\; |M| \circ v \right\}$$

$$\text{AVG}(S) \circ v \mathrel{\hat{=}} \left\{ M \subseteq 2^{A(S)} \;\middle|\; \frac{\sum_{a_i \in M} w_i}{|M|} \circ v \right\}$$

$$\text{MIN}(S) \circ v \mathrel{\hat{=}} \left\{ M \subseteq 2^{A(S)} \;\middle|\; \min\{w_i \mid a_i \in M\} \circ v \right\}$$

$$\text{MAX}(S) \circ v \mathrel{\hat{=}} \left\{ M \subseteq 2^{A(S)} \;\middle|\; \max\{w_i \mid a_i \in M\} \circ v \right\}$$

Note that the definition of $\text{COUNT}(S) \circ v$ does not depend on the weights in S, and therefore in this case we will usually omit the weights and only specify the atoms in $A(S)$.

A *logic program* P is a set of *rules* of the following form:

$$a \leftarrow a_1, \ldots, a_l, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_m, C_1, \ldots, C_n \quad (1)$$

where $l, m, n \in \mathbb{N}$ are natural numbers, $a \in A$ (the *head*), $a_1, \ldots, a_l, b_1, \ldots, b_m \in A$ and $C_1, \ldots, C_n$ are Boolean functions (the *body*). We also write rules (1) as $r = a \leftarrow B$ and use the notations $B^+ = \{a_1, \ldots, a_l\}$, $B^- = \{b_1, \ldots, b_m\}$ and $B^C = \{C_1, \ldots, C_n\}$ to access body constituents.

A *partial interpretation* is represented by a pair (X, Y) of sets of atoms with $X \subseteq Y \subseteq A$, where the atoms in the *lower bound* X are true and the atoms *not* in the *upper bound* Y are false. Thus, the atoms in $Y \setminus X$ are neither true nor false, that is, they do not have a classical truth value yet and are therefore *undefined*. A partial interpretation (X, Y) is a model of a Boolean function C, denoted $(X, Y) \models C$, iff for all $Z \subseteq A$ with $X \subseteq Z \subseteq Y$, we find $Z \in C$. Given a partial interpretation (X, Y) and a Boolean function C, the (three-valued) *model checking* problem consists of verifying whether $(X, Y) \models C$ holds.

The semantics of a logic program P is given by the set of its stable models, where a stable model is a set of atoms satisfying some stability condition. In this paper, the stability condition is given by means of the least fixpoint of an inference operator. Formally, for each $Y \subseteq A$, define an operator $T_P^Y : 2^A \to 2^A$ such that:

$$X \mapsto \{a \in A \mid a \leftarrow B \in P, B^+ \subseteq X, B^- \cap Y = \emptyset, \\ (X, Y) \models D \text{ for all } D \in B^C\}.$$

A set $M \subseteq A$ is a *stable model* of P if and only if M is the $\subseteq$-least fixpoint of T_P^M. Clearly if $(X, Y) \models D$ then $(Z, Y) \models D$ for all $X \subseteq Z \subseteq Y$, thus the operator T_P^M is $\subseteq$-monotone and always has a unique $\subseteq$-least fixpoint.

Example 1. Consider $A = \{a, b, c\}$ and logic program P:

$$a \leftarrow \text{SUM}(\{1 : b, 1 : c\}) > 0$$
$$b \leftarrow a, \mathit{not}\ c$$
$$c \leftarrow \mathit{not}\ b$$

The only two candidates for stable models are $M = \{a, b\}$ and $N = \{a, c\}$. For the first candidate, we find that

- $a \notin T_P^M(\emptyset)$ since $(\emptyset, M) \not\models \text{SUM}(\{1 : b, 1 : c\}) > 0$,
- $b \notin T_P^M(\emptyset)$ since $a \notin \emptyset$, and $c \notin T_P^M(\emptyset)$ since $b \in M$.

Thus $T_P^M(\emptyset) = \emptyset$, which means that the $\subseteq$-least fixpoint of T_P^M is $\emptyset$. Since $\emptyset \neq M$, the set M is not a stable model of P. For the other candidate $N = \{a, c\}$, we get $T_P^N(\emptyset) = \{c\}$:

- $a \notin T_P^N(\emptyset)$ since $(\emptyset, N) \not\models \text{SUM}(\{1 : b, 1 : c\}) > 0$,
- $b \notin T_P^N(\emptyset)$ since $a \notin \emptyset$, and $c \in T_P^N(\emptyset)$ since $b \notin N$,

and then $T_P^N(\{c\}) = \{a, c\} = N = T_P^N(N)$ because

- $(\{c\}, N) \models \text{SUM}(\{1 : b, 1 : c\}) > 0$ and
- $(N, N) \models \text{SUM}(\{1 : b, 1 : c\}) > 0$.

Thus the $\subseteq$-least fixpoint of T_P^N is the set $N = \{a, c\}$, whence this set is also the only stable model of P. ▲

Note that the definition of stable model above is equivalent to the one given by Pelov, Denecker, and Bruynooghe (2007), and Son and Pontelli (2007). We reformulated it in this way to clarify that model checking is the potentially most complex part of verifying whether a given set of atoms is a stable model of a logic program. In fact, dealing with undefined atoms during the computation of the least fixpoint of T_P^M is the main source of complexity for checking the stability of a set of atoms. This is the case because in general each Boolean function in P has to be evaluated with respect to a number of sets of atoms that is exponential in the number of undefined atoms. However, it is important to observe that in practice many Boolean functions do not require to be evaluated on exponentially many sets of atoms in order to answer the associated model checking problem. For these reasons, we focus on identifying sufficient conditions for guaranteeing tractability of model checking, which in turn implies tractability of the stability check for logic programs.

Actually, for some subclasses of Boolean functions, the model checking problem is already known to be tractable. One example are convex Boolean functions, that intuitively do not contain "gaps" in their sets of models. Formally, a Boolean function $C \subseteq 2^A$ is *convex* if and only if for all $X \subseteq Y \subseteq Z \subseteq A$ we have: if $X \in C$ and $Z \in C$ then $Y \in C$. It is well-known that convex Boolean functions are closed under arbitrary conjunctions, but not under complementation and disjunction.

In the next section, we will introduce the notion of bipolar Boolean functions, a different class closed under complementation, conjunction and disjunction subject to some compatibility conditions. Later on, we will present the class of atom-orderable Boolean functions, an extension of both convex and bipolar that also includes other standard aggregates commonly used in logic programming.

Bipolar Boolean Functions

Bipolarity has hitherto predominantly been defined and used in the context of ADFs (Brewka and Woltran 2010). Here, we define bipolarity for Boolean functions in general by extending the notions of monotone and antimonotone Boolean functions (Faber, Pfeifer, and Leone, 2011, Def. 2.4).

Definition 1. Let A be a set of atoms, $C \subseteq 2^A$ be a Boolean function, and $a \in A$.

- C is *monotone* in a iff for all $M \subseteq A$, we find that:
$$M \in C \text{ implies } M \cup \{a\} \in C;$$
- C is *antimonotone* in a iff for all $M \subseteq A$, we find that:
$$M \notin C \text{ implies } M \cup \{a\} \notin C.$$

Define the sets

$$A_C^+ = \{a \in A \mid C \text{ is monotone in } a\},$$
$$A_C^- = \{a \in A \mid C \text{ is antimonotone in } a\}.$$

A Boolean function $C \subseteq 2^A$ is:

- *monotone* iff $A = A_C^+$;
- *antimonotone* iff $A = A_C^-$;
- *bipolar* iff $A = A_C^+ \cup A_C^-$. ▲

Synonymously to C is monotone in a, we say that a *is supporting in* C; likewise, C is antimonotone in a iff a *is attacking in* C. Being supporting or attacking is the *polarity* of the argument a in C. As all atoms $a \in A_C^+ \cap A_C^-$ are redundant, we also use the sets of strictly supporting arguments $A_C^+ \setminus A_C^-$ and strictly attacking arguments $A_C^- \setminus A_C^+$.

First of all, we observe that the class of bipolar Boolean functions captures quite a range of standard aggregates, as shown below.

Proposition 1. *Let A be a vocabulary, S be a weighted atom set over A and $v \in \mathbb{R}$. The following Boolean functions are bipolar:*

1. $\text{SUM}(S) \circ v$ *for* $\circ \in \{<, \leq, \geq, >\}$;
2. $\text{COUNT}(S) \circ v$ *for* $\circ \in \{<, \leq, \geq, >\}$;
3. $\text{AVG}(S) \circ v$ *for* $\circ \in \{<, \leq, \geq, >\}$;
4. $\text{MIN}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \geq, >\}$;
5. $\text{MAX}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \geq, >\}$.

Proof. 1. For $\circ \in \{<, \leq\}$, atoms a_i with non-negative weights ($w_i \geq 0$) are attacking, atoms a_i with non-positive weights ($w_i \leq 0$) are supporting. For $\circ \in \{\geq, >\}$, atoms with non-negative weights are supporting, atoms with non-positive weights are attacking.

2. For $\circ \in \{<, \leq\}$ all atoms are attacking, for $\circ \in \{\geq, >\}$ all atoms are supporting.

3. For $\circ \in \{<, \leq\}$, all atoms a_i with weight $w_i \geq v$ are attacking, atoms a_i with weight $w_i \leq v$ are supporting; $\circ \in \{\geq, >\}$ is symmetric (atoms a_i with weight $w_i \geq v$ are supporting, those with weight $w_i \leq v$ are attacking).

4. For $\text{MIN}(S) = v$, all atoms a_i with weight $w_i \geq v$ are supporting, additionally all atoms a_i with $w_i \neq v$ are attacking. For $\text{MIN}(S) < v$, all atoms are supporting, and additionally all atoms a_i with $w_i \geq v$ are attacking; similarly, for $\text{MIN}(S) \leq v$, all atoms are supporting, and additionally all atoms a_i with $w_i > v$ are attacking. For $\circ \in \{\geq, >\}$, all atoms a_i with weight $w_i \circ v$ are supporting, all others attacking.

5. Dual to $\text{MIN}(S) \circ v$. □

Comparing the different classes of Boolean functions that we introduced so far, we can observe that by definition all monotone Boolean functions are bipolar *and* convex, but the converse does not hold. It is similar for antimonotone Boolean functions.

Example 2. For vocabulary $A = \{a, b\}$, the Boolean function $C_{a \wedge \neg b} = \{\{a\}\}$ is bipolar and convex, but neither monotone (b is strictly attacking) nor antimonotone (a is strictly supporting). ▲

Even more importantly, we have to clarify that the two notions bipolar and convex are independent of each other.

Example 3. Consider the vocabulary $A = \{a, b\}$. The Boolean function $C_{\neg a \vee b} = \{\emptyset, \{b\}, \{a, b\}\}$ is bipolar (a is strictly attacking, b is strictly supporting), but not convex (for $\emptyset \subseteq \{a\} \subseteq \{a, b\}$, we have that $\emptyset, \{a, b\} \in C_{\neg a \vee b}$ while $\{a\} \notin C_{\neg a \vee b}$). On the other hand, the Boolean function $C_{a \oplus b} = \{\{a\}, \{b\}\}$ is convex, but not bipolar (for example, a is not supporting, as $\{b\} \in C_{a \oplus b}$ but $\{a, b\} \notin C_{a \oplus b}$; neither is a attacking, as $\emptyset \notin C_{a \oplus b}$ but $\{a\} \in C_{a \oplus b}$). ▲

Hence, even if there is some overlap, bipolar Boolean functions and convex Boolean functions seem to have orthogonal expressive capabilities. The two classes also differ with respect to closure under common set operators. In fact, it can be shown that the complement of a bipolar Boolean function is again bipolar but with the polarities switched.

Proposition 2. *Let A be a set of atoms and $C \subseteq 2^A$ be a bipolar Boolean function. Then the set $\overline{C} = 2^A \setminus C$ is a bipolar Boolean function with $A^+_{\overline{C}} = A^-_C$ and $A^-_{\overline{C}} = A^+_C$.*

Therefore, bipolar Boolean functions are closed under complementation, while this is not the case for intersection and union in general.

Example 4. Consider the vocabulary $A = \{a, b, c\}$. For the bipolar Boolean functions $C_{a \vee b} = \{\{a\}, \{b\}, \{a, b\}\}$ and $C_{\neg a \vee \neg b} = \{\emptyset, \{a\}, \{b\}\}$ we get the resulting (non-bipolar) intersection $C_{a \vee b} \cap C_{\neg a \vee \neg b} = \{\{a\}, \{b\}\} = C_{a \oplus b}$. ▲

However, closure under union and intersection can be regained by stipulating a compatibility condition on bipolar Boolean functions.

Definition 2. Let A be a set of atoms and $C, D \subseteq 2^A$ be bipolar Boolean functions. C and D are *compatible* iff

- $A^+_C \cap A^-_D \subseteq A^-_C \cup A^+_D$, and
- $A^-_C \cap A^+_D \subseteq A^+_C \cup A^-_D$. ▲

Intuitively, two Boolean functions over the same vocabulary are compatible iff for each atom, the polarities of the arguments in the Boolean functions match point-wise. The polarities match if the argument is supporting in both Boolean functions or attacking in both Boolean functions. Put another way, for two Boolean functions to be compatible, whenever an argument is supporting in one Boolean function and attacking in the other, then it must be redundant in one of them, which is what the definition above says.

Example 5. Consider the vocabulary $A = \{a, b, c\}$. The bipolar Boolean functions $C_{a \wedge \neg b} = \{\{a\}, \{a, c\}\}$ (a supporting, b attacking, c redundant) and $C_{\neg b \vee c} = \{\{\}, \{a\}, \{c\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$ (a redundant, b attacking, c supporting) are compatible: $A^+_{C_{a \wedge \neg b}} = \{a, c\}$, $A^-_{C_{a \wedge \neg b}} = \{b, c\}$, $A^+_{C_{\neg b \vee c}} = \{a, c\}$, $A^-_{C_{\neg b \vee c}} = \{a, b\}$, thus:

$$A^+_{C_{a \wedge \neg b}} \cap A^-_{C_{\neg b \vee c}} = \{a\} \subseteq \{a, b, c\} = A^-_{C_{a \wedge \neg b}} \cup A^+_{C_{\neg b \vee c}};$$

$$A^-_{C_{a \wedge \neg b}} \cap A^+_{C_{\neg b \vee c}} = \{c\} \subseteq \{a, b, c\} = A^+_{C_{a \wedge \neg b}} \cup A^-_{C_{\neg b \vee c}}.$$

Clearly $C_{a \wedge \neg b} \cap C_{\neg b \vee c} = C_{a \wedge \neg b}$ is again bipolar. For the Boolean functions in Example 4 we get $A^+_{C_{a \vee b}} = \{a, b\}$, $A^-_{C_{a \vee b}} = \emptyset$, $A^+_{C_{\neg a \vee \neg b}} = \emptyset$, $A^-_{C_{\neg a \vee \neg b}} = \{a, b\}$, whence

$$A^+_{C_{a \vee b}} \cap A^-_{C_{\neg a \vee \neg b}} = \{a, b\} \not\subseteq \emptyset = A^-_{C_{a \vee b}} \cup A^+_{C_{\neg a \vee \neg b}}$$

and the two are not compatible. ▲

It is easy to show that Boolean combinations of compatible bipolar Boolean functions again yield bipolar Boolean functions.

Proposition 3. *Let A be a set of atoms and $C, D \subseteq 2^A$ be bipolar Boolean functions such that C and D are compatible. Then $C \cap D$ and $C \cup D$ are bipolar Boolean functions.*

To sum up, bipolar Boolean functions are closed under complementation, and compatible union and intersection.

Atom-Orderable Boolean Functions

The class of bipolar Boolean functions captures many standard aggregates widely used in logic programming, but not all of them. In this section we introduce and study a broader class in order to cover the missing polytime-decidable cases.

Definition 3. Let A be a set of atoms. A Boolean function $C \subseteq 2^A$ is called *atom-orderable* if and only if there exists a total order $\preccurlyeq$ on A such that for all $X \subseteq Y \subseteq A$ the following two conditions are equivalent:

1. $(X, Y) \models C$, that is, for all $Z \subseteq A$ with $X \subseteq Z \subseteq Y$ we find $Z \in C$;
2. for all $i \in \{0, \ldots, k\}$ (where $k = |Y \setminus X|$), we find $X_i \in C$, where $X_0 = X$ and for $0 \leq j \leq k - 1$ we define $X_{j+1} = X_j \cup \{\min_{\preccurlyeq}(Y \setminus X_j)\}$.[1] ▲

Intuitively, the first condition needs to evaluate the Boolean function on an exponential number of sets of atoms. On atom-orderable aggregates, the second condition yields the equivalent result using only a linear number of sets of atoms by exploiting the ordering on the atoms, adding them one by one in ascending order from $X_0 = X$ to $X_k = Y$.

Example 6. Consider the vocabulary $A = \{a, b, c, d\}$ and the Boolean function C given by $\text{COUNT}(\{a, b, c, d\}) \neq 1$. This function is neither bipolar nor convex, but atom-orderable with $a \prec b \prec c \prec d$: To show that $(\emptyset, A) \not\models \text{COUNT}(\{a, b, c, d\}) \neq 1$, we need only check 1. $\emptyset \in C$? (yes), 2. $\{a\} \in C$? (no) and are done (instead of naively searching among the 16 counterexample candidates). To show that $(\{a, b\}, \{a, b, c, d\}) \models C$, it suffices to show that $\{a, b\}$, $\{a, b, c\}$, $\{a, b, c, d\}$ are contained in C. ▲

It is easy to see that condition (2) can be checked in deterministic polynomial time (in n) whenever $Z \in C$ can be decided in polynomial time and the ordering $\preccurlyeq$ is given.

Proposition 4. *Let A be a set of atoms, $C \subseteq 2^A$ be an atom-orderable Boolean function with $\preccurlyeq$ given, and $X \subseteq Y \subseteq A$. Furthermore assume that the problem "given $Z \subseteq A$, is $Z \in C$?" is in P. Then checking $(X, Y) \models C$ is in P.*

We will usually represent atom-orderable Boolean functions by giving the ordering $\preccurlyeq$; if we specify $\preccurlyeq$ as a partial order only, then any total order extending $\preccurlyeq$ will do as a witness for the Boolean function being atom-orderable. We first show that our new class generalises the class of bipolar Boolean functions.

Proposition 5. *All bipolar Boolean functions are atom-orderable.*

Proof. Let A be a set of atoms and $C \subseteq 2^A$ be bipolar. Then $A = A^+_C \cup A^-_C$. We define the partial order $\preccurlyeq$ such that

$$A^-_C \setminus A^+_C \prec A^+_C \setminus A^-_C \prec A^+_C \cap A^-_C.$$

Let $X \subseteq Y \subseteq A$ be arbitrary. We have to show that conditions (1) and (2) of Definition 3 are equivalent.

[1] Here, $\min_{\preccurlyeq}(Y \setminus X_j)$ denotes the $\preccurlyeq$-least element of the set $Y \setminus X_j$, which is unique since $\preccurlyeq$ is total and $Y \setminus X_j$ is non-empty.

(1) $\Rightarrow$ (2): Assume that for all $Z \subseteq A$ with $X \subseteq Z \subseteq Y$, we find $Z \in C$. Recall that $X_0 = X$ and for $0 \leq j \leq n-1$ we set $X_{j+1} = X_j \cup \min_{\preccurlyeq}(Y \setminus X_j)$. Clearly $X \subseteq X_i \subseteq Y$ for all $i \in \{0, \ldots, k\}$, whence (2) follows.

(2) $\Rightarrow$ (1): Assume that for all $i \in \{0, \ldots, k\}$, we find $X_i \in C$. By our definition of $\preccurlyeq$, there is in particular an $i \in \{0, \ldots, k\}$ such that $X_i = X \cup ((A_C^- \setminus A_C^+) \cap Y) \in C$. Now let $Z \subseteq A$ with $X \subseteq Z \subseteq Y$ be arbitrary. We have to show $Z \in C$. Since X_i contains all attackers and no supporters (relative to $Y \setminus X$), we can reconstruct Z from X_i by "adding" supporters and "removing" attackers: there exist $Z^+ \subseteq A_C^+$ and $Z^- \subseteq A_C^-$ such that $Z = (X_i \cup Z^+) \setminus Z^-$. Since $X_i \in C$ and C is bipolar, it follows that $Z \in C$ as desired. □

This new class is furthermore a strict generalisation, as it allows us to treat a maximum possible number of additional cases. This, together with Definition 3, is the main result of the paper.

Theorem 6. *Let A be a vocabulary, S be a weighted atom set over A and $v \in \mathbb{R}$. The following Boolean functions are atom-orderable:*

1. $\text{COUNT}(S) \circ v$ *for* $\circ \in \{=, \neq\}$;
2. $\text{SUM}(S) = v$;
3. $\text{AVG}(S) = v$;
4. $\text{MIN}(S) \neq v$;
5. $\text{MAX}(S) \neq v$;
6. $\text{PROD}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \geq, >\}$.

Proof. 1. $\text{COUNT}(S) \circ v$ for $\circ \in \{=, \neq\}$: We can set $\preccurlyeq$ arbitrary in both cases: for $X \subseteq Y \subseteq A$, $(X, Y) \models \text{COUNT}(S) = v$ iff $|X \cap A(S)| = v$ and $Y \cap A(S) = X \cap A(S)$; $(X, Y) \models \text{COUNT}(S) \neq v$ iff $|X \cap A(S)| > v$ or $|Y \cap A(S)| < v$. In both cases, it is only important that we check the two extremal interpretations (X, X) and (Y, Y), which is achieved by any ordering.

2. $\text{SUM}(S) = v$: Again, the ordering $\preccurlyeq$ is irrelevant: $(X, Y) \models \text{SUM}(S) = v$ iff $(X, X) \models \text{SUM}(S) = v$ and all atoms in $A(S) \cap (Y \setminus X)$ have weight zero. Clearly, testing the atoms one by one will correctly check whether their weights are zero, the order in which they are tested is irrelevant.

3. $\text{AVG}(S) = v$: Similarly as for $\text{SUM}(S) = v$, the ordering is irrelevant since we have to check whether $(X, X) \models \text{AVG}(S) = v$ and all atoms in $A(S) \cap (Y \setminus X)$ have weight v.

4. $\text{MIN}(S) \neq v$: Let $S = \{w_1 : a_1, \ldots, w_m : a_m\}$. We define $a_i \preccurlyeq a_j$ iff $w_j \leq w_i$, for all $i, j \in \{1, \ldots, m\}$. Effectively, we have to check whether v is among the weights of the atoms in $A(S) \cap (Y \setminus X)$ (if the minimum weight in X is greater than v). This is guaranteed in our approach by considering the atoms in order of decreasing weights.

5. $\text{MAX}(S) \neq v$: This case is dual to $\text{MIN}(S) \neq v$, define $a_i \preccurlyeq a_j$ iff $w_i \leq w_j$ for $S = \{w_1 : a_1, \ldots, w_m : a_m\}$.

6. $\text{PROD}(S) \circ v$ for $\circ \in \{>, \geq, =\}$: Let $S = \{w_1 : a_1, \ldots, w_m : a_m\}$. Consider $X \subseteq Y \subseteq A$ with $|Y \setminus X| = n$. The proof uses a case distinction, where in each case of comparison operator and bound we define an ordering $\preccurlyeq$ and show that conditions 1 and 2 of Definition 3 are equivalent.

- $\text{PROD}(S) = 0$: For $(X, Y) \models \text{PROD}(S) = 0$ it is necessary that X contain an atom with weight zero. In this case, the ordering $\preccurlyeq$ is irrelevant.
- $\text{PROD}(S) = v$: In order for (X, Y) to be a model for this aggregate, the product of the weights in X must equal v, and the weights of all atoms in $Y \setminus X$ must equal one. Since in this case all atoms in $Y \setminus X$ have the same weight, adding them in any order will work.
- $\text{PROD}(S) \geq 0$: It is clear that $(X, Y) \models \text{PROD}(S) \geq 0$ iff there are no atoms with negative weights in $Y \setminus X$. By adding all negatively weighted atoms (if any) one by one *before* adding all the zero-weighted atoms, we will correctly identify the presence of an atom with negative weight in $Y \setminus X$. Thus a possible ordering is given by $a_j \preccurlyeq a_k$ iff $w_j \leq w_k$ for $1 \leq j, k \leq m$.
- $\text{PROD}(S) \geq v$ and $\text{PROD}(S) > v$ for $v > 0$: Define $\preccurlyeq$ such that

$$\{a_i \mid w_i > 0\} \prec \{a_i \mid w_i < 0\} \prec \{a_i \mid w_i = 0\}$$

  and furthermore for all $a_i \in A(S)$ with $w_i > 0$, we have $a_j \preccurlyeq a_k$ iff $w_j \leq w_k$ for $1 \leq j, k \leq m$. To figure out whether $(X, Y) \models \text{PROD}(S) \geq v$, we essentially have to find the set Z with $X \subseteq Z \subseteq Y$ such that the value $\prod_{a_i \in Z \cap A(S)} w_i$ is $\leq$-minimal. If there are $a_i \in Y \setminus X$ with $w_i \leq 0$, then they will be detected. Assuming that all weights are positive, the least possible product is given by $\prod_{a_i \in Y \setminus X, 0 < w_i < 1} w_i$. Due to the increasing ordering of the positively weighted atoms, there will be a $j \in \{0, \ldots, n\}$ such that $X_j = X \cup \{a_i \mid 0 < w_i < 1\}$. That is, the atom set leading to the least possible product will be checked.
- $\text{PROD}(S) \geq v$ and $\text{PROD}(S) > v$ for $v < 0$: Define $\preccurlyeq$ such that

$$\{a_i \mid w_i \geq 1\} \prec \{a_i \mid w_i < 0\} \prec \{a_i \mid 0 < w_i < 1\} \prec \{a_i \mid w_i = 0\}$$

  and furthermore for all $a_i \in A(S)$ with $w_i < 0$, we have $a_j \preccurlyeq a_k$ iff $w_j \leq w_k$ for $1 \leq j, k \leq m$. In essence, we have to find the set Z with $X \subseteq Z \subseteq Y$ such that the value $\prod_{a_i \in Z \cap A(S)} w_i$ is $\leq$-minimal. Our ordering achieves this by considering first all weights greater than 1 (to reach the maximal absolute value) and then all negative weights. If there is an overall odd number of negative weights, all of them will contribute to the least possible product. If the number of negative weights is even, then the least possible overall product is obtained by taking all positive weights and all but one (the one with the least absolute value) negative weights.
- The remaining cases with $\circ \in \{<, \leq\}$ can be reduced to the cases above by multiplying the given inequality with -1. □

Since all bipolar Boolean functions are also atom-orderable, we get the following overall result.

Corollary 7. *Let A be a vocabulary, S be a weighted atom set over A and $v \in \mathbb{R}$. The following Boolean functions are atom-orderable:*

1. $\text{COUNT}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \neq, \geq, >\}$;
2. $\text{SUM}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \geq, >\}$;
3. $\text{AVG}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \geq, >\}$;
4. $\text{MIN}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \neq, \geq, >\}$;
5. $\text{MAX}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \neq, \geq, >\}$;
6. $\text{PROD}(S) \circ v$ *for* $\circ \in \{<, \leq, =, \geq, >\}$.

This result is optimal, as model checking is coNP-hard for the cases $\text{SUM}(S) \neq v$, $\text{AVG}(S) \neq v$ and $\text{PROD}(S) \neq v$ (Pelov 2004; Son and Pontelli 2007). As a final note, we observe that the class of atom-orderable Boolean functions is not closed under common set operations: for example, $\text{SUM}(S) > v \cup \text{SUM}(S) < v$ is equivalent to $\text{SUM}(S) \neq v$.

Related Work

Properties of logic programs with Boolean functions have been analysed extensively in the literature. Among the several semantics that were proposed (Ferraris 2011; Faber, Pfeifer, and Leone 2011; Gelfond and Zhang 2014), we have considered the one by Pelov, Denecker, and Bruynooghe (2007) and Son and Pontelli (2007) with the aim of extending the currently largest class of Boolean functions for which the stability check is tractable. In fact, concerning stable models by Ferraris (2011) and Faber, Pfeifer, and Leone (2011), it is known that convex Boolean functions are the complexity boundary for this task (Alviano and Faber 2013). Moreover, concerning stable models by Gelfond and Zhang (2014), it is known that the task is tractable in general if disjunction in rule heads is forbidden (Alviano and Leone 2015).

Complexity of logic programs with Boolean functions can be analysed by considering each specific case by itself (Pelov 2004; Son and Pontelli 2007), or by identifying some semantic classes such as monotone, antimonotone and convex that cover practical cases (Pelov 2004; Liu and Truszczynski 2006; Faber, Pfeifer, and Leone 2011). In this paper, we followed this second approach and introduced the notion of bipolarity in logic programming. Even if the definition stems from ADFs (Brewka and Woltran 2010), it is interesting to observe that many common aggregates are actually bipolar, as shown in Proposition 1. This is an original result, which eventually provides an alternative proof for several complexity results by Son and Pontelli (2007).

Since other known tractability results are not covered by the class of bipolar Boolean functions, we also introduced the larger class of atom-orderable Boolean functions, and proved that the missing cases fall in this class (see Theorem 6). Interesting cases are those associated with PROD, originally considered by Pelov (2004). In fact, several algorithms were given by Pelov (2004, Figures 5.1–5.3) in order to show tractability of model checking for Boolean functions induced by PROD. Within our approach, we only had to show the existence of an ordering for the aggregate atoms with the desired properties (see the proof of Theorem 6).

Stable models by Pelov, Denecker, and Bruynooghe (2007) and Son and Pontelli (2007) were recently extended to the disjunctive case by Shen et al. (2014). The notion of bipolar and atom-orderable Boolean functions can also be used in the disjunctive case, and the complexity results are expected to extend as well in all cases in which the disjunction is not a complexity source itself (for example, in the head-cycle free case; Ben-Eliyahu and Dechter, 1994).

The notion of bipolarity, and even more that of atom-orderability, may be useful for other constructs such as HEX atoms (Eiter et al. 2014), whose semantics is also defined by means of Boolean functions. In fact, knowing that an HEX atom is atom-orderable may allow to implement a more efficient evaluation algorithm depending on the desired semantics (Shen et al. 2014).

Recently, Strass (2015) presented a syntactic counterpart of bipolar Boolean functions, that is, a subset of the formulas of classical propositional logic whose elements have all and only bipolar Boolean functions associated to them. (Roughly, these "bipolar formulas" are in negation normal form and no propositional atom may occur both positively and negatively in the formula.) It would certainly be useful to have a syntactic counterpart of atom-orderable functions.

Discussion

Boolean functions are among the most used extensions of logic programs. Identifying classes of Boolean functions with good computational properties is important from a practical viewpoint because any concrete implementation must face the complexity of the model checking problem. In this work, we introduced a unifying semantic class covering all known tractable cases. It is called atom-orderable because its main property is that the Boolean function's arguments – its input atoms – can be ordered so that model checking can be efficiently done by evaluating Boolean functions with respect to linearly many sets of atoms. For common aggregates such an ordering is also efficiently computable, while in general the language can be extended by allowing the user to specify the ordering along with each Boolean function in the input program.

There are other advantages resulting from our approach. In fact, tractability of other aggregates can be easily proved by showing membership in the class of atom-orderable Boolean functions. This is the case, for example, of the *median*, that is, the number separating the higher half of a data sample from the lower half. It can be observed that $\text{MEDIAN}(S) \circ v$, for $\circ \in \{<, \leq, \geq, >\}$, is atom-orderable: for $S = \{w_1 : a_1, \ldots, w_m : a_m\}$, the ordering $\preccurlyeq$ is such that $a_i \prec a_j$ iff $w_i \not\circ v$ and $w_j \circ v$. It is also interesting to note that the missing cases, that is, $\text{MEDIAN}(S) \circ v$ with $\circ \in \{=, \neq\}$, can be captured by slightly extending the class of atom-orderable Boolean functions. In fact, in this case the aggregate atoms can be ordered by increasing weight, but in order to obtain a sound model checking procedure the ordering has to be checked in two directions, ascending and descending. It is natural to generalise atom-orderable

Boolean functions to this case – essentially condition (2) of Definition 3 is replaced by the following:

> For all $i \in \{0, \ldots, k\}$ (where $k = |Y \setminus X|$), we find $X_i \in C$ and $Y_i \in C$, where $X_0 = X$, $Y_0 = Y$, and for $0 \leq j \leq k-1$ we set $X_{j+1} = X_j \cup \{\min_{\preccurlyeq}(Y \setminus X_j)\}$ and $Y_{j+1} = Y_j \setminus \{\min_{\preccurlyeq}(Y_j \setminus X)\}$.

Similar observations hold for the *mode* of a distribution. Median and mode have practical applications in statistics and probability theory. As an example, we report the case of the Italian scientific habilitation for university professors, which requires to have some bibliometric parameters above the median of the professors currently employed.

Acknowledgements. 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.

References

Alviano, M., and Faber, W. 2013. The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In Cabalar, P., and Son, T. C., eds., *LPNMR 2013*, volume 8148 of *Lecture Notes in Computer Science*, 67–72. Springer.

Alviano, M., and Faber, W. 2015. Stable model semantics of abstract dialectical frameworks revisited: A logic programming perspective. In Yang, Q., and Wooldridge, M., eds., *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, 2684–2690. AAAI Press.

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.

Alviano, M.; Calimeri, F.; Faber, W.; Perri, S.; and Leone, N. 2011. Unfounded sets and well-founded semantics of answer set programs with aggregates. *Journal of Artificial Intelligence Research.* AAAI Press 42:487–527.

Bartholomew, M.; Lee, J.; and Meng, Y. 2011. First-order semantics of aggregates in answer set programming via modified circumscription. In *Papers from the 2011 AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*. AAAI.

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

Brewka, G., and Woltran, S. 2010. Abstract Dialectical Frameworks. In *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 102–111.

Brewka, G.; Ellmauthaler, S.; Strass, H.; Wallner, J. P.; and Woltran, S. 2013. Abstract Dialectical Frameworks Revisited. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, 803–809. IJCAI/AAAI.

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

Eiter, T.; Fink, M.; Krennwallner, T.; Redl, C.; and Schüller, P. 2014. Efficient hex-program evaluation based on unfounded sets. *J. Artif. Intell. Res. (JAIR)* 49:269–321.

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

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

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3/4):365–386.

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

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

Liu, L.; Pontelli, E.; Son, T. C.; and Truszczynski, M. 2010. Logic programs with abstract constraint atoms: The role of computations. *Artif. Intell.* 174(3-4):295–315.

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.

Pelov, N. 2004. *Semantics of logic programs with aggregates*. Ph.D. Dissertation, Katholieke Universiteit Leuven, Departement Computerwetenschappen, Celestijnenlaan 200A, 3001 Heverlee, Belgium.

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

Strass, H., and Wallner, J. P. 2015. Analyzing the Computational Complexity of Abstract Dialectical Frameworks via Approximation Fixpoint Theory. *Artif. Intell.* 226:34–74.

Strass, H. 2015. Expressiveness of two-valued semantics for abstract dialectical frameworks. *Journal of Artificial Intelligence Research* 54:193–231.