Reconciling Well-Founded Semantics of DL-Programs and Aggregate Programs*

Jia-Huai You[1], John Morris[1], and Yi Bi[2]

1 **Department of Computing Science**
University of Alberta, Canada
2 **School of Computer Science and Technology**
Tianjin University, China

Abstract

Logic programs with aggregates and description logic programs (dl-programs) are two recent extensions to logic programming. In this paper, we study the relationships between these two classes of logic programs, under the well-founded semantics. The main result is that, under a satisfaction-preserving mapping from dl-atoms to aggregates, the well-founded semantics of dl-programs by Eiter et al., coincides with the well-founded semantics of aggregate programs, defined by Pelov et al. as the least fixpoint of a 3-valued immediate consequence operator under the ultimate approximating aggregate. This result enables an alternative definition of the same well-founded semantics for aggregate programs, in terms of the first principle of unfounded sets. Furthermore, the result can be applied, in a uniform manner, to define the well-founded semantics for dl-programs with aggregates, which agrees with the existing semantics when either dl-atoms or aggregates are absent.

1998 ACM Subject Classification I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Well-founded semantics, description logic programs, aggregate logic programs, three-valued logic.

Digital Object Identifier 10.4230/LIPIcs.ICLP.2012.235

1 Introduction

In logic programming beyond positive logic programs, almost all semantics of the current interest can be traced back to the origin of two semantics, the stable model semantics [10] and the well-founded semantics [17]. While the former is based on guess-and-verify to sort contradictory information into different stable models/answer sets, the latter is defined by a built-in mechanism to circumvent contradictory conclusions, thus making safe inferences in the presence of data that require conflicting interpretations.

More recently, well-founded semantics have been studied for two extensions of logic programming: *logic programs with aggregates* (or, *aggregate programs*) [5, 13, 14] and *description logic programs* (*dl-programs*) [8]. The former brings into logic programming reasoning with constraints, while the latter is an example of logic programming with *external atoms* [7]. In a dl-program an atom can be a *dl-atom*, which is a well-designed interface to an underlying description logic knowledge base. In this way, some decidable fragments of first-order logic can be integrated into rule-based non-monotonic reasoning. These extensions substantially widen the application range of logic programming.

* The work was partially supported by Natural Sciences and Engineering Research Council of Canada.

Technical Communications of the 28th International Conference on Logic Programming (ICLP'12).
Editors: A. Dovier and V. Santos Costa; pp. 235–246

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An aggregate is a constraint, which is a relation on a domain where the tuples in the relation are called *admissible solutions*. A dl-atom can also be viewed as a constraint, in terms of the sets of (ordinary) atoms under which it is satisfied. Despite this close connection, the semantics for these two kinds of programs have been studied independently. In [8], the semantics is defined under the first principle of unfounded sets, while in the work of Pelov et al. [14], a purely algebraic framework is developed under the theory of approximating operators on bilattices [4], parameterized by approximating operators and aggregate relations. In particular, given an aggregate program Π, the well-founded semantics, based on the least fixpoint of a 3-valued immediate consequence operator Φ_{Π}^{aggr}, is defined along with the *ultimate aggregate relation*. Let us call this semantics the (ultimate) well-founded semantics of Π. It extends the well-founded semantics for normal logic programs.

In this paper, we study the relationships between dl-programs and aggregate programs under the well-founded approach. The main result is that the well-founded semantics of dl-programs can be obtained from the ultimate well-founded semantics of aggregate programs, under a mapping from dl-atoms to aggregates. This leads to the following conclusions: on the one hand, the well-founded semantics for dl-programs can be viewed as a special case of the ultimate well-founded semantics for aggregate programs, and on the other hand, the latter semantics can be defined, alternatively, employing the notion of unfounded sets.[1] As a result, the well-founded semantics can be defined, in a uniform manner using the first principle of unfoundedness, for logic programs that may contain both dl-atoms and aggregates.

The paper is organized as follows. The next section provides some definitions. Section 3 introduces the well-founded semantics for dl-programs. Section 4 shows that under a mapping from dl-atoms to aggregates, the well-founded semantics for dl-programs is precisely that of the corresponding aggregate programs. Then in Section 5 we extend the well-founded semantics to logic programs that may contain dl-atoms as well as aggregates. Section 6 is about related work followed by comments on future work.

2 Preliminaries

We introduce dl-programs. Although technically this paper does not intimately depend on description logics (DLs) [1], some familiarity would be convenient.

A *DL knowledge base* L consists of a finite set of axioms built over a vocabulary $\Sigma_L = (\mathbf{A} \cup \mathbf{R}, \mathbf{I})$, where $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ are pairwise disjoint (denumerable) sets of *atomic concepts, atomic roles* and *individuals*, respectively. As usual, concepts can be built from atomic concepts and other constructs, such as $\sqcap$ (conjunction), $\sqcup$ (disjunction), $\neg$ (negation), and various restrictions (see [1] for more details).

Let $\mathbf{P}$ be a finite set of *predicate* symbols and $\mathbf{C}$ a nonempty finite set of *constants* such that $\mathbf{P} \cap (\mathbf{A} \cup \mathbf{R}) = \emptyset$ and $\mathbf{C} \subseteq \mathbf{I}$. A *term* is either a constant from $\mathbf{C}$ or a *variable*. An atom is of the form $p(t_1, ..., t_m)$, where p is a predicate symbol from $\mathbf{P}$, and t_i is a term. An equality (resp. inequality) is of the form $t_1 = t_2$ (resp. $t_1 \neq t_2$), where t_1 and t_2 are terms. A *dl-query* is of the form $Q(\mathbf{t})$, where $\mathbf{t}$ is a list of terms, and Q is an equality/inequality symbol, or a concept, a role, or a concept inclusion axiom, built from $\mathbf{A} \cup \mathbf{R}$.

A *dl-atom* is of the form $DL[S_1 op_1 p_1, \cdots, S_m op_m p_m; Q](\mathbf{t})$, where S_i is a concept or role built from $\mathbf{A} \cup \mathbf{R}$, or an equality/inequality symbol; $op_i \in \{\uplus, \uplus, \Cap\}$; $p_i \in \mathbf{P}$ is a unary

[1] In fact, such a definition of unfounded sets was already proposed in [9].

predicate symbol if S_i is a concept, and a binary predicate symbol otherwise; and $Q(\mathbf{t})$ is a dl-query.

A *dl-rule* (or *rule*) is of the form $h \leftarrow A_1, ..., A_m, not\ B_1, ..., not\ B_n$, where h is an atom, and A_i and B_i are atoms or equalities/inequalities or dl-atoms. An atom or a dl-atom A, and its negated form *not* A, is called a *literal*. For any rule r, we denote the head of the rule by $H(r)$, and the body by $B(r)$. In addition, $B^+ = \{A_1, ..., A_m\}$ and $B^- = \{B_1, ..., B_n\}$. A *rule base* P is a finite set of rules.

A *dl-program* is a combined knowledge base $KB = (L, P)$, where L is a DL knowledge base and P is a rule base.

A *ground instance* of a rule r is obtained by first replacing every variable in r with a constant from $\mathbf{C}$, then replacing with $\top$ (resp. $\bot$) every equality/inequality if it is valid (resp. invalid) under the unique name assumption (UNA). $\top$ and $\bot$ are two special predicates such that $\top$ (resp. $\bot$) is true (resp. false) in every interpretation.

In this paper, we assume a rule base P is already grounded using the constants appearing in the given non-ground program. Likewise, when we refer to an atom/dl-atom/literal, by default we mean it is one without variables.

The *Herbrand base* of a rule base P, denoted by HB_P, is the set of all ground atoms $p(t_1, ..., t_m)$, where p is from $\mathbf{P}$ and t_i is a constant from $\mathbf{C}$, both occurring in P. Any subset of HB_P is an *interpretation* of P.

▶ **Definition 1.** *Let $KB = (L, P)$ be a dl-program and $I \subseteq HB_P$ an interpretation. Define the* satisfaction relation under L, *denoted $\models_L$, as follows:*

1. *$I \models_L \top$ and $I \not\models_L \bot$.*
2. *For any atom $a \in HB_P$, $I \models_L a$ if $a \in I$.*
3. *For any (ground) dl-atom $A = DL[S_1 op_1 p_1, \cdots, S_m op_m p_m; Q](\mathbf{c})$ occurring in P, $I \models_L A$ if $L \cup \bigcup_{i=1}^{m} A_i \models Q(\mathbf{c})$, where*
$$A_i = \begin{cases} \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \uplus\ ; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \cup\ ; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}, & \text{if } op_i = \cap\ . \end{cases}$$
4. *For any ground atom or dl-atom A, $I \models_L not\ A$ if $I \not\models_L A$.*

The above satisfaction relation naturally extends to conjunctions of literals. For a rule $r \in P$, $I \models_L r$ if $I \not\models_L B(r)$ or $I \models_L H(r)$. I is a model of a dl-program $KB = (L, P)$ if $I \models_L r$ for all $r \in P$.

A ground dl-atom A is *monotonic* relative to KB if for any $I \subseteq J \subseteq HB_P$, $I \models_L A$ implies $J \models_L A$. Otherwise, A is *nonmonotonic*.

Additional notations: Given a set S of atoms, $\neg.S = \{\neg a \mid a \in S\}$; given a set P of rules, $Lit_P = HB_P \cup \neg.HB_P$; if I is a set of literals, $I^+ = \{a \mid a \text{ is an atom in } I\}$ and $I^- = \{a \mid \neg a \in I\}$; a set of literals $I \subseteq Lit_P$ is consistent if there is no atom a such that $a \in I$ and $\neg a \in I$. In this paper, by an *interval* $[S_1, S_2]$, where S_1 and S_2 are sets and $S_1 \subseteq S_2$, we mean the set $\{S \mid S_1 \subseteq S \subseteq S_2\}$.

3 Well-Founded Semantics for Arbitrary DL-Programs

The well-founded semantics is first defined for dl-programs with dl-atoms that may only contain operators $\uplus$ and $\cup$ [8]. These dl-atoms are monotonic. It is then commented (see Section 9.2 of [8]) that the definition can be generalized to the class of all dl-programs. For contrast, here we introduce the well-founded semantics for arbitrary dl-programs directly.

▶ **Definition 2. (Unfounded set)** Let $KB = (L, P)$ be a dl-program and $I \subseteq Lit_P$ be consistent. A set $U \subseteq HB_P$ is an *unfounded set of KB relative to I* iff the following holds:

For every $a \in U$ and every rule $r \in P$ with $H(r) = a$, either (i) $\neg b \in I \cup \neg.U$ for some ordinary atom $b \in B^+(r)$, or (ii) $b \in I$ for some ordinary atom $b \in B^-(r)$, or (iii) for some $b \in B^+(r)$, it holds that $S^+ \not\models_L b$ for each consistent $S \subseteq Lit_P$ with $I \cup \neg.U \subseteq S$, or (iv) for some $b \in B^-(r)$, it holds that $S^+ \models_L b$ for each consistent $S \subseteq Lit_P$ with $I \cup \neg.U \subseteq S$.

Intuitively, the definition says that an atom a is in an unfounded set U, relative to I, because, for every rule with a in the head, at least one body literal is not satisfied by I under L, and this fact remains to hold for any consistent extension of $I \cup \neg.U$.

▶ **Definition 3.** Let $KB = (L, P)$ be a dl-program. We define the operators T_{KB}, U_{KB}, and W_{KB} on all consistent $I \subseteq Lit_P$ as follows:

(i) $a \in T_{KB}(I)$ iff $a \in HB_P$ and some $r \in P$ exists such that (a) $H(r) = a$, (b) for all $b \in B^+(r)$, $S^+ \models_L b$ for each consistent S with $I \subseteq S \subseteq Lit_P$, (c) $\neg b \in I$ for all ordinary atoms $b \in B^-(r)$, and (d) for all $b \in B^-(r)$, $S^+ \not\models_L b$ for each consistent S with $I \subseteq S \subseteq Lit_P$.

(ii) $U_{KB}(I)$ is the greatest unfounded set of KB relative to I; and

(iii) $W_{KB}(I) = T_{KB}(I) \cup \neg.U_{KB}(I)$.

With the standard definition of monotonicity of operators over complete lattices, one can verify easily that the operators T_{KB}, U_{KB}, and W_{KB} are all monotonic.

As a notation, we define $W^0_{KB} = \emptyset$, and $W^{i+1}_{KB} = W_{KB}(W^i_{KB})$, for all $i \geq 0$. In the sequel, we will use a similar notion for other monotonic operators, but sometimes we may start applying such an operator from a nonempty set (this will be made clear when such a situation arises).

▶ **Definition 4. (Well-founded Semantics)** Let $KB = (L, P)$ be a dl-program. The *well-founded semantics* of KB, denoted by $WFS(KB)$, is defined as the least fixpoint of the operator W_{KB}, denoted $lfp(W_{KB})$. An atom $a \in HB_P$ is *well-founded* (resp. *unfounded*) relative to KB iff a (resp. $\neg a$) is in $lfp(W_{KB})$.

▶ **Example 5.** Consider a dl-program $KB = (\emptyset, P)$, where P consists of

$$\begin{array}{ll} r_1: & p(a) \leftarrow not\ DL[S_1 \uplus q, S_2 \uplus r; \neg S_1 \sqcap S_2](a). \\ r_2: & q(a) \leftarrow DL[S \uplus q; S](a). \\ r_3: & r(a) \leftarrow DL[S \uplus q; \neg S](a). \end{array}$$

Starting with $W^0_{KB} = \emptyset$, for example, we do not derive $p(a)$ since there is a consistent extension that satisfies the dl-atom in rule r_1, but $\{q(a)\}$ is an unfounded set relative to $\emptyset$. The reader can verify that $W^1_{KB} = \{\neg q(a)\}$, $W^2_{KB} = \{\neg q(a), r(a)\}$, and $W^3_{KB} = \{\neg q(a), \neg p(a), r(a)\}$, which is the least fixpoint of W_{KB}.

We now discuss an alternative way to construct the least fixpoint of W_{KB}. The technical result given here will be used later when relating to the ultimate well-founded semantics for aggregate programs.

Since the operator T_{KB} only generates positive atoms, given a consistent $I \subseteq Lit_P$, we can apply T_{KB} iteratively, with I^- fixed. That is,

$$T^0_{KB} = I^+, T^1_{KB} = T_{KB}(T^0_{KB} \cup \neg.I^-), ..., T^{k+1}_{KB} = T_{KB}(T^k_{KB} \cup \neg.I^-), ... \quad (1)$$

Since this sequence is $\subseteq$-increasing, a fixpoint exists. Let us denote it by $FP_{T_{KB}}(I)$. Note that the operator $FP_{T_{KB}} : Lit_P \rightarrow HB_P$ is monotonic relative to a fixed I^-. Namely, for any consistent sets of literals I_1 and I_2 such that $I_1^- = I_2^-$ and $I_1 \subseteq I_2$, we have $FP_{T_{KB}}(I_1) \subseteq FP_{T_{KB}}(I_2)$.

Now, following Definition 3, we define an operator V_{KB}, which is similar to W_{KB}, as follows: Given a consistent set of literals $I \subseteq Lit_P$,

$$V_{KB}(I) = FP_{T_{KB}}(I) \cup \neg.U_{KB}(I) \tag{2}$$

As the operator V_{KB} is monotonic, its least fixpoint exists, which we denote by $lfp(V_{KB})$. We can show that (the proof is omitted for lack of space)

▶ **Lemma 6.** $lfp(V_{KB}) = lfp(W_{KB})$.

4 Representing DL-Programs by Aggregate Programs

In general, an aggregate in a logic program is a constraint atom. Since in this paper our interest is in the semantics, we assume that an aggregate is a constraint whose semantics is pre-defined in terms of its domain and admissible solutions. An explicit representation of such constraints has been called *abstract constraint atoms* (or just *c-atoms*) [12].

We assume a propositional language, $\mathcal{L}_\Sigma$, determined by a fixed countable set Σ of propositional atoms. A *c-atom* A is a pair (D, C), where D is a nonempty finite set of atoms in Σ and $C \subseteq 2^D$. We use A_d and A_c to refer to the components D and C of A, respectively. As an abstraction, a c-atom A can be used to represent the semantics of any constraint with a set A_c of admissible solutions over a finite domain A_d [11, 12]. Therefore, in the sequel we will use the aggregate notation and c-atoms exchangeably.

The *complement* of a c-atom A is the c-atom A' with $A'_d = A_d$ and $A'_c = 2^{A_d} \setminus A_c$.

An interpretation $I \subset \Sigma$ *satisfies* an atom a if $a \in I$; $\neg a$ if $a \notin I$. I satisfies a c-atom A, written as $I \models A$, if $A_d \cap I \in A_c$; *not* A, written $I \models not\ A$, if $A_d \cap I \notin A_c$. Therefore, it follows that I satisfies *not* A iff I satisfies the complement of A. I satisfies a conjunction E of atoms or c-atoms, written $I \models E$, if I satisfies every conjunct in it.

A c-atom A is *monotone* if for any $J \supseteq I$, that I satisfies A implies J satisfies A. Otherwise, A is nonmonotone.

An *aggregate program* (or exchangeably, a *logic program with c-atoms*) is a finite set of rules of the form $h \leftarrow B_1, ..., B_n, not\ C_1, ..., not\ C_k$, where h, B_i, and C_i are ordinary atoms or c-atoms. Given a rule r, we use $H(r)$ to denote the head and $B(r)$ to denote the body.

Note that in [14] negative aggregates $\neg C$ are allowed, but here we write them as *not* C.

The notations Lit_Π, S^+, and S^- (given a set of literals S) are defined similarly as for those used for dl-programs.

▶ **Definition 7. (From dl-programs to aggregate programs)** Given a dl-program $KB = (L, P)$, we obtain an aggregate program, denoted $\beta(KB)$, by a mapping β_{KB} from atoms, dl-atoms, and their default negation occurring in P to aggregates as follows:

- If A is an ordinary atom a then $\beta_{KB}(A) = a$, and
- If A is a dl-atom then $\beta_{KB}(A) = (HB_P, C)$, where $C = \{I \subseteq HB_P \mid I \models_L A\}$.
- For any default negation of the form *not* A, $\beta_{KB}(not\ A) = \beta_{KB}(A)'$.

In the sequel, as the underlying KB is always clear, we will drop the subscript in β_{KB}. Also, by abuse of notation, given a rule $r \in P$, we denote by $\beta(B(r))$ the translated conjunction in the body of r, and by $\beta(r)$ the translated rule. Since this mapping does not

introduce new symbols, given a dl-program $KB = (L, P)$, we can identify Σ for the translated aggregate programs with HB_P.

As an example, consider a dl-program $KB = (\emptyset, \{p(a) \leftarrow DL[S \uplus p; \neg S](a)\})$. The translated aggregate program consists of a single rule, $p(a) \leftarrow (\{p(a)\}, \{\emptyset\})$. The c-atom in the rule represents the semantics of, e.g., the cardinality constraint, $card_{=}(\{x | p(x)\}, 0)$.

▶ **Lemma 8.** *Let $KB = (L, P)$ be a dl-program and $I \subseteq HB_P$.*
(i) *For any dl-atom A, $I \models_L A$ iff $I \models \beta(A)$, and $I \not\models_L A$ iff $I \models \beta(A)'$.*
(ii) *For any rule $r \in P$, I satisfies r iff I satisfies $\beta(r)$.*
(iii) *I is a model of KB iff I is a model of $\beta(KB)$.*

4.1 Relationship

Following [13, 14], given a complete lattice $\langle L, \preceq \rangle$, the *bilattice* induced from it is the structure $\langle L^2, \preceq, \preceq_p \rangle$, where for all $x, y, x', y' \in L$,

$$\begin{array}{ll} (x, y) \preceq (x', y') & \text{if and only if } \ x \preceq x' \text{ and } y \preceq y' \\ (x, y) \preceq_p (x', y') & \text{if and only if } \ x \preceq x' \text{ and } y' \preceq y \end{array}$$

The order $\preceq$ on L^2 is called the *produce order*, while $\preceq_p$ is called the *precision order*. Both orders are complete lattice orders on L^2. We are interested only in the subset of pairs (x, y) that are *consistent*, i.e., $x \preceq y$, and when $x = y$ it is said to be *exact*. We denote the set of consistent pairs by L^c.

Given a bilattice $\langle L^2, \preceq, \preceq_p \rangle$, the $\preceq$-least element is $(\bot, \bot)$ and the $\preceq_p$-least element is $(\bot, \top)$. E.g., consider the complete lattice $\langle 2^\Sigma, \subseteq \rangle$ where Σ is a set of atoms. For the bilattice induced from it, the $\preceq$-least element is $(\emptyset, \emptyset)$ and the $\preceq_p$-least element is $(\emptyset, \Sigma)$.

The idea of the well-founded semantics for an aggregate program is to start with the $\preceq_p$-least element $\langle \emptyset, \Sigma \rangle$, and apply an *approximating operator*, denoted A, in a way that not only are the true atoms computed, but also the false atoms that are not reachable by derivations. It approximates an operator O on L, whose fixpoints are exact pairs on L^2.

▶ **Definition 9.** Let $O : L \to L$ be an operator on a complete lattice $\langle L, \preceq \rangle$. We say that $A : L^c \to L^c$ is an *approximating operator* of O iff the following conditions are satisfied:
- A extends O, i.e., $A(x, x) = (O(x), O(x))$, for every $x \in L$.
- A is $\preceq_p$-monotone.

The condition on A is a mild one: it only requires to extend O on exact pairs, in addition to monotonicity.

For aggregate programs, given a language $\mathcal{L}_\Sigma$, a program Π, and a monotonic approximating operator A of some operator O, we will compute a sequence

$$(\emptyset, \Sigma) = (u_0, v_0), (u_1, v_1), ..., (u_k, v_k), ..., (u_\infty, v_\infty) \tag{3}$$

such that the internal $[u_i, v_i]$ is decreasing, i.e., $[u_{i+1}, v_{i+1}] \subset [u_i, v_i]$, for all i, eventually reaching a fixpoint, which is denoted by (u_∞, v_∞).

Intuitively, one can think of this sequence as representing the process that, initially nothing is *known to be true* and every atom in Σ is *potentially true* (as such, nothing is *known to be false*); and given (u_i, v_i), after the current iteration, $u_{i+1} \setminus u_i$ is the set of atoms that become known to be true and $v_i \setminus v_{i+1}$ is the set of atoms that become known to be false. In this way, u_i represents a *lower estimate* and v_i an *upper estimate* of the eventual fixpoint. At the end, u_∞ is the set of true atoms and $\Sigma \setminus v_\infty$ is the set of false atoms, and the truth value of the remaining atoms is *undefined*. This gives a 3-valued interpretation of the least fixpoint.

A fixpoint operator that constructs sequence (3) can be defined in different ways. For example, we can simply take the approximating operator A, i.e., $A(u_i, v_i) = (u_{i+1}, v_{i+1})$, for all i. If A is $\preceq_p$-monotone, then the least fixpoint of A exists, which is called the *Kripke-Kleene fixpoint* of A.

As alluded to earlier, our interest is the well-founded semantics, which is determined by the so-called *well-founded fixpoint* of A. It is computed by a *stable revision operator*, denoted by $\mathcal{S}t_\Pi$, for a given aggregate program Π. Namely, $\mathcal{S}t_\Pi(u_i, v_i) = (u_{i+1}, v_{i+1})$, where (u_{i+1}, v_{i+1}) is computed from (u_i, v_i) using two component operators of A. The first one, denoted by $A^1(\cdot, v_i)$, is A with v_i fixed, and similarly, the second, denoted by $A^2(u_i, \cdot)$, is A with u_i fixed. Given an upper estimate b, we compute a new lower estimate by an iterative process:

$$x_0 = \bot, x_1 = A^1(x_0, b), ..., x_{i+1} = A^1(x_i, b), ... \tag{4}$$

until a fixpoint is reached. That is, if $b = v_i$, then $u_{i+1} = x_\infty$ where $x_\infty = A^1(x_\infty, b)$. The operator that generates x_∞ is called the *lower revision operator*.

On the other hand, given a lower estimate a, we compute a new upper estimate

$$y_0 = a, y_1 = A^2(a, y_0), ..., y_{i+1} = A^2(a, y_i), ... \tag{5}$$

until a fixpoint is reached. That is, if $a = u_i$, then $v_{i+1} = y_\infty$ where $y_\infty = A^2(a, y_\infty)$. The operator that generates y_∞ is called the *upper revision operator*.

It can be seen that if A is $\preceq_p$-monotone, so is $\mathcal{S}t_\Pi$, thus the least fixpoint of $\mathcal{S}t_\Pi$ can be constructed by a sequence (3), where (u_∞, v_∞) is called the *well-founded fixpoint* of A, which is the least fixpoint of the stable revision operator $\mathcal{S}t_\Pi$.

By this parameterized algebraic approach one can define possibly different well-founded semantics by employing different $\preceq_p$-monotone approximating operators. In the context of aggregate programs, the operator we are approximating is the standard immediate consequence operator extended to aggregate programs Π, i.e., $\mathcal{T}_\Pi : \Sigma \to \Sigma$, where

$$\mathcal{T}_\Pi(I) = \{H(r) \mid r \in \Pi \text{ and } I \models B(r)\}. \tag{6}$$

To approximate $\mathcal{T}_\Pi$ while preserving the well-founded and stable model semantics for normal logic programs, in [13], a three-valued immediate consequence operator Φ_Π^{aggr} is defined for aggregate programs, which maps 3-valued interpretations to 3-valued interpretations. Recall that a 3-valued interpretation can be represented by a pair (I_1, I_2) of 2-valued interpretations with $I_1 \subseteq I_2$, where I_1 is the set of atoms assigned to *true*, $\Sigma \backslash I_2$ is the set of atoms assigned to *false*, and all the other atoms are assigned to *undefined*. Thus, Φ_Π^{aggr} maps a pair of 2-valued interpretations to a pair of 2-valued interpretations, i.e., $\Phi_\Pi^{aggr}(I_1, I_2) = (I_1', I_2')$. The definition of Φ_Π^{aggr} guarantees that it approximates the operator $\mathcal{T}_\Pi$, in that for any fixpoint (I, J) of Φ_Π^{aggr}, and for any x such that $\mathcal{T}_\Pi(x) = x$, $(I, J) \preceq_p (x, x)$.

From the definition of Φ_Π^{aggr} above, two component operators are induced. They are

$$\Phi_\Pi^{aggr,1}(I_1, I_2) = I_1' \quad \text{and} \quad \Phi_\Pi^{aggr,2}(I_1, I_2) = I_2' \tag{7}$$

The original definition of Φ_Π^{aggr} is given in 3-valued logic, parameterized by the choice of *approximating aggregates* [13]. In [16], the authors showed an equivalent definition of $\Phi_\Pi^{aggr,1}$ in terms of *conditional satisfaction*, when the approximating aggregate used is the *ultimate approximating aggregate*. We state this result below (see Appendix of [16]). Here, we replace aggregates with c-atoms. In a similar way, an equivalent definition of $\Phi_\Pi^{aggr,2}$ can be obtained.

▶ **Theorem 10.** *Let* Π *be an aggregate program, and* I *and* M *interpretations with* $I \subseteq M \subseteq \Sigma$. *Then,*

$$\Phi_{\Pi}^{aggr,1}(I, M) = \{H(r) \mid r \in \Pi, \forall J \in [I, M], J \models B(r)\} \tag{8}$$
$$\Phi_{\Pi}^{aggr,2}(I, M) = \{H(r) \mid r \in \Pi, \exists J \in [I, M], J \models B(r)\} \tag{9}$$

▶ **Lemma 11.** *The component operators* $\Phi_{\Pi}^{aggr,1}(\cdot, b)$ *and* $\Phi_{\Pi}^{aggr,2}(a, \cdot)$ *are* $\subseteq$*-monotone, and* Φ_{Π}^{aggr} *is* $\subseteq_p$*-monotone.*

Therefore, the stable revision operator $\mathcal{S}t_{\Pi}$ induced from Φ_{Π}^{aggr} is also $\subseteq_p$-monotone, and we take the least fixpoint of this operator for the semantics. Recall that this fixpoint has been referred to as the well-founded fixpoint of Φ_{Π}^{aggr}.

▶ **Definition 12.** Let Π be an aggregate program and (u_∞, v_∞) the well-founded fixpoint of Φ_{Π}^{aggr}. The *ultimate well-founded semantics* of Π based on Φ_{Π}^{aggr}, denoted by $UWFS(\Pi)$, is defined as $u_\infty \cup \neg.(\Sigma \setminus v_\infty)$.

In the sequel, we will drop the phrase "based on Φ_{Π}^{aggr}", with the understanding that the underlying approximating operator is Φ_{Π}^{aggr} as identified in Theorem 10.

▶ **Example 13.** Consider the following aggregate program Π:

$$p(-1). \qquad p(-2) \leftarrow sum_{\leq}(\{x \mid p(x)\}, 2).$$
$$p(3) \leftarrow sum_{>}(\{x \mid p(x)\}, -4). \qquad p(-4) \leftarrow sum_{\leq}(\{x \mid p(x)\}, 0).$$

The aggregates under *sum* are self-explaining, e.g., $sum_{\leq}(\{x \mid p(x)\}, 2)$ means that the sum of x for satisfied atoms $p(x)$ is less than or equal to 2. For the construction of the well-founded fixpoint, we start with the pair $(\emptyset, \Sigma)$. The reader can apply equations in (7) to verify: by applying the operator $\Phi_{\Pi}^{aggr,1}(\cdot, \Sigma)$ iteratively, we get a new lower estimate $Q = \{p(-1), p(-2), p(-4)\}$; and by applying $\Phi_{\Pi}^{aggr,2}(\emptyset, \cdot)$ iteratively, we get an upper estimate Σ, which is the same as before. Thus the new pair is (Q, Σ). Continuing in the next iteration, Q remains the same but $p(3)$ is no longer derivable. We thus have $(Q, \Sigma - \{p(3)\})$, which is a fixpoint. So the ultimate well-founded semantics is that all atoms in Q are true, $p(3)$ is false, and nothing is undefined.

▶ **Theorem 14.** *Let* $KB = (L, P)$ *be a dl-program. The well-founded semantics of* KB *coincides with the ultimate well-founded semantics of the aggregate program* $\beta(KB)$. *That is,* $WFS(KB) = UWFS(\beta(KB))$.

Proof. (Sketch) Let $\Pi = \beta(KB)$ and (u_∞, v_∞) in sequence (3) be the ultimate well-founded fixpoint of Φ_{Π}^{aggr}. Recall that $WFS(KB) = lfp(W_{KB}) = lfp(V_{KB})$ (the latter is by Lemma 6) and $UWFS(\Pi) = u_\infty \cup \neg.(\Sigma \setminus v_\infty)$. We prove the coincidence by induction on the sequences of constructing $lfp(V_{KB})$ and (u_∞, v_∞). In the proof, we assume rules in P are of the form $a \leftarrow \phi$ or $a \leftarrow not\ \phi$, where ϕ is a dl-atom. The proof can be generalized to arbitrary dl-rules. Below, we identify Σ for the corresponding aggregate program with HB_P for the given dl-program, i.e., we let $\Sigma = HB_P$.

Clearly, $V_{KB}^0 = u_0 \cup \neg.(\Sigma \setminus v_0) = \emptyset$. Assume $(V_{KB}^i)^+ = u_i$ and $(V_{KB}^i)^- = \Sigma \setminus v_i$ and we show $(V_{KB}^{i+1})^+ = u_{i+1}$ and $(V_{KB}^{i+1})^- = \Sigma \setminus v_{i+1}$, for all $i \geq 0$. Note that from (1), and by induction hypothesis, we have

$$(V_{KB}^{i+1})^+ = FP_{T_{KB}}(V_{KB}^i) = FP_{T_{KB}}(u_i \cup \neg.(\Sigma \setminus v_i)) \tag{10}$$

(a) $(V_{KB}^{i+1})^+ = u_{i+1}$. First, observe that for the approximating operator Φ_{Π}^{aggr}, x_∞ in (4) can be computed equivalently by starting with u_i, i.e.,

$$x_0 = u_i, x_1 = \Phi_{\Pi}^{aggr,1}(x_0, v_i), ..., x_{i+1} = \Phi_{\Pi}^{aggr,1}(x_i, v_i), ..., x_\infty = \Phi_{\Pi}^{aggr,1}(x_\infty, v_i) \quad (11)$$

and $u_{i+1} = x_\infty$. According to (10), we need to show $FP_{T_{KB}}(u_i \cup \neg.(\Sigma \setminus v_i)) = u_{i+1}$. We prove this by showing a one-one correspondence between the steps in (1) and those in (11). That is, $x_k = T_{KB}^k$ for all $k \geq 0$. The base case is due to the induction hypothesis, namely $x_0 = u_i = (V_{KB}^i)^+ = T_{KB}^0$ (note that T_{KB}^0 here refers to the one in (1)). Assume $x_k = T_{KB}^k$ and we show $x_{k+1} = T_{KB}^{k+1}$, for all $k \geq 0$. For any atom $a \in \Sigma$, $a \in x_{k+1}$ iff for some rule $r \in P$ with $H(r) = a$ such that for every $J \in [x_k, v_i]$, $J \models \beta(B(r)))$. Let us label the last statement as (C1).

Suppose $r = a \leftarrow \phi$. By Lemma 8, $J \models \beta(\phi)$ iff $J \models_L \phi$. From the induction hypothesis, we have $(V_{KB}^i)^- = \Sigma \setminus v_i$, and it follows

$$[x_k, v_i] = \{S^+ \mid S \text{ is consistent and } x_k \cup \neg.\Sigma \setminus v_i \subseteq S \subseteq Lit_P\}$$

From $x_k = T_{KB}^k$, it follows that (C1) iff $a \in T_{KB}^{k+1}$, as the condition (b) of Definition 3 is satisfied: for all $b \in B^+(r)$, $S^+ \models_L b$ for each consistent S with $T_{KB}^k \subseteq S \subseteq Lit_P$. The case where $r = a \leftarrow not\ \phi$ can be proved similarly, based on condition (d) of Definition 3.

(b) $(V_{KB}^{i+1})^- = \Sigma \setminus v_{i+1}$. Namely, $U_{KB}(V_{KB}^i) = \Sigma \setminus v_{i+1}$, i.e., the greatest unfounded set of KB relative to V_{KB}^i is precisely the fixpoint y_∞ $(= v_{i+1})$ below:

$$y_0 = u_i, y_1 = \Phi_{\Pi}^{aggr,2}(u_i, y_0), ..., y_{i+1} = \Phi_{\Pi}^{aggr,2}(u_i, y_i), ..., y_\infty = \Phi_{\Pi}^{aggr,2}(u_i, y_\infty) \quad (12)$$

(b-1) Prove that for any $a \in \Sigma$, if $a \in v_{i+1}$ then $a \notin U$, for any unfounded set U of KB relative to V_{KB}^i. By definition, $a \in v_{i+1}$ iff $a \in y_k$, for some $k \geq 0$, iff there is a rule $r \in P$ with $H(r) = a$ such that $\exists J \in [u_i, y_k]$, $J \models \beta(B(r))$. By Lemma 8, $J \models \beta(B(r))$ iff $J \models_L \phi$, if r is of form $a \leftarrow \phi$. This violates condition (iii) in Definition 2, as by induction hypothesis there is a consistent extension S of V_{KB}^i such that $S^+ = J$. The proof is similar if r is of form $a \leftarrow not\ \phi$, in which case condition (iv) is violated.

(b-2) Show that $a \notin v_{i+1} \Rightarrow a \in U_{KB}(V_{KB}^i)$, for all $a \in \Sigma$. That $a \notin v_{i+1}$ (i.e., $a \notin y_\infty$) means, for every rule $r \in P$ with $H(r) = a$, and for all $I \in [u_i, y_\infty]$, $I \not\models \beta(B(r))$, hence by Lemma 8, $I \not\models_L \phi$ if $r = a \leftarrow \phi$ and $I \models_L \phi$ if $r = a \leftarrow not\ \phi$. Note that

$$[u_i, y_\infty] = \{S^+ \mid S \text{ is consistent and } u_i \cup \neg.\Sigma \setminus y_\infty \subseteq S \subseteq Lit_P\}$$

From the induction hypothesis we know $u_i = (V_{KB}^i)^+$ and $(V_{KB}^i)^- = \Sigma \setminus v_i$, and notationally $v_{i+1} = y_\infty$. It follows from Definition 2 that $\Sigma \setminus v_{i+1}$ is an unfounded set of KB relative to V_{KB}^i, and $a \in \Sigma \setminus v_{i+1}$. Obviously, it is the greatest unfounded set of KB relative to V_{KB}^i, since for any atom $\varphi \in y_\infty$, there is a derivation of φ based on V_{KB}^i. Therefore, $a \in U_{KB}(V_{KB}^i)$. The proof is completed. ◀

5 Well-Founded Semantics of DL-Programs with Aggregates

A dl-program with aggregates is a combined knowledge base $KB = (L, P)$, where L is a DL knowledge base and P a finite set of rules of the form $a \leftarrow b_1, ..., b_k, \texttt{not}\ c_1, ..., \texttt{not}\ c_n$, where a is an atom, and each b_i or c_j is either an ordinary atom, a dl-atom, or an aggregate atom. In the following, we continue to denote by HB_P the set of atoms composed from the constants and predicate symbols of the underlying language.

Now we extend the satisfaction relation $\models_L$ to cover aggregates. Let $KB = (L, P)$ be a dl-program with aggregates and $I \subseteq HB_P$ an interpretation. For any aggregate ϕ, we define $I \models_L \phi$ iff $I \models \phi$, and extend $\models_L$ naturally to conjunctions of atoms, dl-atoms, aggregates, and their negations. Then, Definitions 2 and 3 can be adopted directly, by replacing "dl-program" with "dl-program with aggregates". To distinguish, let us denote the fixpoint operator W_{KB} in Definition 3 by W'_{KB}.

▶ **Definition 15. (Well-founded semantics for dl-programs with aggregates)** Let $KB = (L, P)$ be a dl-program with aggregates. The *well-founded semantics* of KB is defined as the least fixpoint of the operator W'_{KB}, denoted $lfp(W'_{KB})$.

▶ **Theorem 16.** *Let $KB = (L, P)$ be a dl-program with aggregates. (i) If P contains no aggregates, then $lfp(W'_{KB}) = lfp(W_{KB})$; and (ii) If P contains no dl-atoms, then $lfp(W'_{KB}) = lfp(W'_P) = UWFS(P)$.*

For illustration, we close this section by presenting a dl-program with aggregates.

▶ **Example 17.** Consider $KB = (L, P)$ with $L = \{Vip \sqsubseteq CR\}$, possibly plus some assertions of individuals in the concepts Vip and/or CR, where CR stands for Customer-Record, and P containing

1. $purchase(X) \leftarrow purchase(X, Obj), item(Obj)$.
2. $client(X) \leftarrow DL[CR \uplus purchase; CR](X)$.
3. $imp\_client(X) \leftarrow DL[Vip](X)$.
4. $imp\_client(X) \leftarrow client(X), sum_{\geq}(\{Y \mid item(Obj), cost(Obj,Y), purchase(X,Obj)\},100)$.
5. $discount(X) \leftarrow imp\_client(X)$.
6. $promo\_offer(X) \leftarrow DL[CR \uplus imp\_client; CR](X), card_{=}(\{Y \mid purchase(Y)\}, 0)$.

Rule 1 is self-explaining. Rule 2 queries the DL knowledge base in order to enhance the *client* predicate. In rules 3 and 4 we establish that important clients are those who have spent at least \$100 or are VIPs. Rules 5 and 6 provide benefits to certain customers. In rule 5, a discount is offered to important clients - VIPs and those whose purchases sum to \$100 or more. Rule 6 describes a promotional offer for non-VIP customers who have not made any purchases - they are potential clients. For applications, P may contain some facts about *items*, *cost*, and *purchase*.

6 Related Work and Further Direction

The close relationships between well-founded model, partial stable models, and stable models are well-understood (see, e.g., [6, 15, 19]). That the well-founded model of a normal logic program is contained in all its stable models makes it possible in a stable model solver to compute the well-founded model as the first approximation. The well-founded semantics has been defined for disjunctive programs [18] and default logic [2]. The close relationship between dl-programs and aggregate programs is noticed in [8], but left as an interesting future direction.

In [9], the notion of unfounded sets for arbitrary aggregate programs is defined (which generalizes that of [3] for logic programs with monotone and anti-monotone aggregates): A set of atoms U is an unfounded set for an aggregate program P and a partial interpretation I, if for every $a \in U$, and for every rule $r \in P$ with a as the head, a literal ξ in $B(r)$ is false w.r.t. I or w.r.t. $(I - U) \cup \neg.U$. The latter expression equals $I \cup \neg.U$ if $I \cap U = \emptyset$. Here, falsity in I amounts to falsify in all of its totalization. It thus gives the same effect as requiring

that ξ is not satisfied by any consistent extension of $I \cup \neg.U$ in our definition. Thus, it follows from our result that, if we define the well-founded semantics for arbitrary aggregate programs using the notion of unfounded sets in [9], the resulting semantics is equivalent to the ultimate well-founded semantics defined by Pelov et al. [13, 14].

The complexity issues for various classes of dl-programs and aggregate programs under the well-founded semantics will be addressed in future work.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. Gerhard Brewka and Georg Gottlob. Well-founded semantics for default logic. *Fundamenta Informaticae*, 31(3/4):221–236, 1997.
3. Francesco Calimeri, Wolfgang Faber, Nicola Leone, and Simona Perri. Declarative and computational properties of logic programs with aggregates. In *Proc. IJCAI-05*, pages 406–411, 2005.
4. M. Denecker, V. W. Marek, and M. Truszczynski. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation*, 192(1):84–121, 2004.
5. M. Denecker, N. Pelov, and M. Bruynooghe. Ultimate well-founded and stable semantics for logic programs with aggregates. In *Proc. ICLP'01*, pages 212–226, 2001.
6. Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
7. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proc. IJCAI-05*, pages 90–96, 2005.
8. Thomas Eiter, Thomas Lukasiewicz, Giovambattista Ianni, and Roman Schindlauer. Well-founded semantics for description logic programs in the semantic web. *ACM Transactions on Computational Logic*, 12(2), 2011. Article 3.
9. W. Faber. Unfounded sets for disjunctive logic programs with arbitrary aggregates. In *proc. LPNMR-05*, pages 40–52, 2005.
10. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. ICLP'88*, pages 1070–1080, 1988.
11. V. W. Marek and J. B. Remmel. Set constraints in logic programming. In *Proc. LPNMR-04*, pages 167–179, 2004.
12. V. W. Marek and M. Truszczynski. Logic programs with abstract constraint atoms. In *Proceedings of AAAI-04*, pages 86–91, 2004.
13. N. Pelov, M. Denecker, and M. Bruynooghe. Partial stable models for logic programs with aggregates. In *Proc. LPNMR-04*, pages 207–219, 2004.
14. N. Pelov, M. Denecker, and M. Bruynooghe. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming*, 7:301–353, 2007.
15. Teodor C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundam. Inform.*, 13(4):445–463, 1990.
16. Tran Cao Son and Enrico Pontelli. A constructive semantic characterization of aggregates in answer set programming. *TPLP*, 7(3), 2007.
17. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.

18 Kewen Wang and Lizhu Zhou. Comparisons and computation of well-founded semantics for disjunctive logic programs. *ACM Trans. Comput. Log.*, 6(2):295–327, 2005.

19 Jia-Huai You and Li Yan Yuan. On the equivalence of semantics for normal logic programs. *J. Log. Program.*, 22(3):211–222, 1995.