Polynomial Approximation to Well-Founded Semantics for Logic Programs with Generalized Atoms: Case Studies

Md. Solimul Chowdhury[1], Fangfang Liu[2], Wu Chen[3], Arash Karimi[1], Jia-Huai You[1]

[1] Department of Computing Science, University of Alberta, Canada
{mdsolimu, arash.karimi, you}@cs.ualberta.ca
[2] College of Computer and Information Science, Shanghai University, China
[3] College of Computer and Information Science, Southwest University, China
ffliu@shu.edu.cn, chenwu@swu.edu.cn

Abstract. The well-founded semantics of normal logic programs has two main utilities, one being an efficiently computable semantics with a unique intended model, and the other serving as polynomial time constraint propagation for the computation of answer sets of the same program. When logic programs are generalized to support constraints of various kinds, the semantics is no longer tractable, which makes the second utility doubtful. This paper considers the possibility of tractable but *incomplete methods*, which in general may miss information in the computed result, but never generates wrong conclusions. For this goal, we first formulate a well-founded semantics for logic programs with generalized atoms, which generalizes logic programs with arbitrary aggregates/constraints/dl-atoms. As a case study, we show that the method of removing non-monotone dl-atoms for the well-founded semantics by Eiter et al. actually falls into this category. We also present a case study for logic programs with standard aggregates.

Keywords: Polynomial Approximation, Well-Founded Semantics, Generalized Atoms.

1 Introduction

Logic programming with negation is a programming paradigm for declarative knowledge representation and problem solving [5]. In a logic program, a problem is represented as a set of rules and the solutions of the problem are determined under a predefined semantics. In more recent years, logic programs have been incorporated with various types of predefined atoms, to enable reasoning with constraints and external knowledge sources. Examples include logic program with aggregates [1, 12, 15, 21], logic programs with abstract constraint atoms [16, 22], dl-programs [8, 10], and logic programs with external sources (e.g. HEX-programs [9]).

For a unifying framework for the study of logic programs, following [2], in this paper a constraint atom in its generic form is called *a generalized atom* and logic programs with these atoms are called logic programs with generalized atoms. The main interest of this paper lies in the two case studies of polynomial approximation to the well-founded semantics of these programs.

It is well-known that the well-founded semantics of normal logic programs can be computed in polynomial time [24]. Besides many applications, the mechanism of computing the well-founded semantics under a partial truth value assignment can be employed in the computation of answer sets. This is utilized, for example, in SMODELS [20] by the Expand function (though the term of well-founded semantics was not used explicitly at that time); it is invoked before and after every choice point in search. To serve as constraint propagation, it is essential that this computational process is effective. However, when constraints are incorporated into logic programs, the resulting well-founded semantics becomes intractable, since determining the satisfiability of a generalized atom by all consistent extensions of a partial interpretation is co-NP-complete [1, 21].[4] For the utility in constraint propagation in answer set computation, this is a problem.

In this paper, we consider tractable but incomplete methods. That is, we would like to transform a given program P to another one P' in polynomial time, so that

– the well-founded semantics of P' is tractable, and
– the well-founded semantics of P' specifies a subset of conclusions by the well-founded semantics of P (when restricted to the language of P).

Of course, such a transformation should be non-trivial, as the empty program trivially satisfies these properties.

The approach described above is based on a single transformed program P'. In general, however, the computation of an approximation of the well-founded semantics may be carried out by a collection of program components each of which is employed for some specific computations, as long as the overall process takes polynomial time. We will call all of such approaches incomplete methods. But for simplicity, in the following we feel free to describe the effect of an incomplete method by referring to the approach based on a single transformed program P'.

Incomplete methods have practical implications. For example, for computing the well-founded semantics of a logic program with (arbitrary) generalized atoms, one can always compute such an approximation first. For the utility in answer set computation, let us assume that the well-founded semantics of P approximates the answer sets of P, under a suitable definition of answer set. That is, any atom true (resp. false) in the well-founded semantics of P, called *well-founded* (resp. *unfounded*), remains to hold in any answer set of P. Then, in the computation of answer sets of P, constraint propagation can be performed by computing the well-founded semantics of P' that extends a partial truth value assignment, at any choice point in search for answer sets. The above conditions guarantee that constraint propagation is correct and effective.

It is well-known that an efficient constraint propagation mechanism is essential in all popular search engines, e.g., BCP in Boolean Satisfiability (SAT) [6], the Expand function in Answer Set Programming (ASP) [20], and various consistency techniques in Constraint Programming (CP) [19].

[4] This assumes that the satisfaction of a generalized atom by one interpretation can be determined in polynomial time.

To study incomplete methods, the first question is which well-founded semantics is to be approximated, and how it is defined. In [11], for (disjunctive) logic programs with arbitrary aggregates, the notion of unfounded set is defined but the well-founded semantics itself is not spelled out explicitly. In [10], a well-founded semantics for logic programs with (a certain kind of) monotone dl-atoms is defined, using a similar notion of unfounded set, followed by a comment (cf. Section 9.2) that the same approach can be generalized to programs with arbitrary dl-atoms. But the details about this generalization are not provided. In [1], a well-founded semantics for logic programs with monotone and anti-monotone aggregates is presented, also based on a similar notion of unfounded set. Despite these efforts, to our knowledge, there has been no unified approach such that for all of the above classes of logic programs, the well-founded semantics falls into the same theoretical framework. Apparently, a unified approach is important in studying the common properties of logic programs in syntactically different forms. Thus, our first task is to formally define this well-founded semantics for logic programs with generalized atoms, which are first introduced in [2] for the study of answer set semantics for various classes of logic programs. Hence, our work presented here can be viewed as complementing that of [2] for the study of well-founded semantics.

The well-founded semantics for logic programs with arbitrary generalized atoms is not tractable in the general case. We then carry out two case studies on the possibility of polynomial approximation. In the first, we consider dl-programs with arbitrary dl-atoms. In fact, the authors of [10] give a simple rewrite scheme for removing nonmonotone dl-atoms from a program, in which every occurrence of the *constraint operator* $\cap\!\!\!\!-$ is replaced. It is shown that the data complexity for $\cap\!\!\!\!-$-free dl-programs is tractable. However, we show that the transformation is not faithful - the well-founded semantics of the transformed program in general is not equivalent to the well-founded semantics of the original program. On the other hand, we show that the rewrite scheme is correct. Namely, though it may miss some conclusions, it never generates incorrect ones. This thus provides the first case of polynomial approximation for a practical class of logic programs with generalized atoms.

We then turn our attention to logic programs with aggregates. We adopt the *disjunctive rewrite scheme* of abstract constraint atoms described in [14] for aggregates with standard aggregate functions [21]. We show that disjunctive rewrite is in general an incomplete method - it may miss conclusions but never generates wrong ones for logic programs with (standard) aggregates. For these programs, we formulate two different methods of disjunctive rewrite to construct a polynomial process for the computation of an approximation of the well-founded semantics.

The rest of the paper is organized as follows: Section 2 provides syntax and notations. Section 3 defines the well-founded semantics for logic program with generalized atoms and studies its basic properties. Then, we carry out case studies on polynomial approximation for dl-programs in Section 4, and for logic programs with aggregates in Section 5, followed by related work and discussion in Section 6.

2 Preliminaries

2.1 Language

We assume a language $\mathcal{L}$ that includes a countable set of ground atoms Σ (atoms for short). A *literal* is either an atom or its negation. An *interpretation* is a subset of Σ.

Following the spirit of [2], a *generalized atom* α on Σ is a mapping from interpretations to the Boolean truth values $\mathbf{t}$ and $\mathbf{f}$.[5] There is no particular syntax requirement, except that a generalized atom α is associated with a domain, denoted $Dom(\alpha) \subseteq \Sigma$, and the size of α is $|Dom(\alpha)|$.[6] We assume that a generalized atom may only mention atoms in Σ (i.e., it is not nested). An interpretation I *satisfies* α, written $I \models \alpha$, if α maps I to $\mathbf{t}$, I does not satisfy a, written $I \not\models a$, if a maps I to $\mathbf{f}$.

Intuitively, a generalized atom is a constraint, whose semantics is defined externally by how it may be satisfied by sets of atoms.

Since an atom in Σ can be viewed as a special case of a generalized atom, in the sequel, the term *generalized atoms* also includes all atoms in Σ. At times of possible confusion, we may use the term *ordinary atoms* for those in Σ for distinction. As usual, an interpretation I satisfies an ordinary atom a, written $I \models a$, if $a \in I$ (a maps I to $\mathbf{t}$), and I does not satisfy a, written $I \not\models a$, if $a \notin I$ (a maps I to $\mathbf{f}$).

A generalized atom α is *monotone* if for every interpretation I such that $I \models \alpha$, we have $J \models \alpha$ for all interpretations J such that $J \supseteq I$; it is *anti-monotone* if for every interpretation I such that $I \models \alpha$, we have $J \models \alpha$ for all interpretations J such that $J \subseteq I$; it is *non-monotone* if it is not monotone.

Aggregate atoms, abstract constraint atoms, HEX-atoms, dl-atoms, weight constraints, and global constraints in constraint satisfaction can all be modeled by generalized atoms. Below we sketch some examples.

Example 1.
- An *aggregate atom* consists of a mapping from multi-sets to numbers, a comparison operator, and a value. For example, following the notation of [21], $SUM(\{X|p(X)\}) \geq 1$ denotes a (non-ground) aggregate atom: after grounding one can view the set of ground instances of $p(X)$ as the domain of the atom. The semantics of the aggregate is defined by interpretations in which the sum of the arguments in satisfied $p(.)$ is greater than or equal to 1.
- An *abstract constraint atom* (or a *c-atom*) is of the form (D, S), where $D \subseteq \Sigma$ serves as the *domain* and S is a set of subsets of D representing *admissible solutions* of the c-atom [17]. A c-atom (D, S) is satisfied by an interpretation I iff $I \cap D \in S$. For example, the aggregate atom $SUM(\{X|p(X)\}) \geq 1$ above with domain $D = \{p(1), p(-1), p(2)\}$ can be represented by a c-atom (D, S) where $S = \{\{p(1)\}, \{p(2)\}, \{p(-1), p(2)\}, \{p(1), p(2)\}, \{p(1), p(-1), p(2)\}\}$.
  Note that a c-atom contains an *internal* specification of how it may be satisfied by sets of atoms, while the satisfiability of a generalized atom is defined externally. In both cases, the meaning of such an atom is defined by how it is satisfied by sets of atoms, in spite of the syntactic differences in appearance.

[5] Generalized atoms in [2] are essentially conjunctions of generalized atoms defined here.

[6] Domain is needed primarily for complexity measures. It is also indirectly involved in the notion of satisfaction, which gives the semantics of a generalized atom.

– Global constraints in Constraint Satisfaction can be represented by generalized atoms, by giving a name and domain of the constraint. Such a constraint is considered "built-in", i.e., its meaning is pre-defined. For example, a $allDiff$ global constraint modeling the pigeon hole problem, can be specified by a generalized atom with the domain that consists of atoms each of which represents a pigeon taking a hole. Suppose there are two pigeons x and y and two holes $\{1, 2\}$. It can be represented by a generalized atom, say g, such that $Dom(g) = \{x(1), x(2), y(1), y(2)\}$, where, for example, $x(1)$ means pigeon x is in hole 1; g is satisfied by interpretations such that every pigeon is in a unique hole and no hole can hold up more than one pigeon.

2.2 Logic programs with generalized atoms

A *logic program with generalized atoms* (or sometimes just called a *program*) is a finite set of rules of the form: $a \leftarrow \beta_1, ..., \beta_m, \neg\beta_{m+1}, ..., \neg\beta_n$, where $m, n \geq 0$, a is an ordinary atom and β_i $(1 \leq i \leq n)$ are generalized atoms. A rule is *normal* if all β_i are ordinary atoms, and a program is *normal* if all rules in it are normal. A program is called *a logic program with monotone and anti-monotone generalized atoms*, if every generalized atom in P is monotone or anti-monotone.

For a rule r of the above form, the head of the rule is denoted by $H(r) = a$ and the body of the rule by $B(r) = \{\beta_1, ..., \beta_m, \neg\beta_{m+1}, ..., \neg\beta_n\}$. Also, we define $Pos(r) = \{\beta_1, ..., \beta_m\}$ and $Neg(r) = \{\beta_{m+1}, ..., \beta_n\}$ to denote positive atoms and negative atoms of $B(r)$ respectively. We may use sets as conjunctions. A *generalized literal* is either a generalized atom α, or its negation, $\neg\alpha$. Note that, without confusion, ordinary literals are (special cases of) generalized literals.

Let I be an interpretation, β a generalized atom and r a rule. Recall that if β maps I to $\mathbf{t}$ we say that I satisfies β and write $I \models \beta$. We define that $I \models \neg\beta$ if β maps I to $\mathbf{f}$, and $I \models B(r)$ if $I \models l$ for all $l \in B(r)$. I is a *model* of a set of rules P if I satisfies every $r \in P$.

Let I be an interpretation. By $\bar{I} = \{a \mid a \in \Sigma \backslash I\}$, we denote the set of atoms exclusive of I.

Well-founded semantics are typically defined by building a *partial interpretation*. Let S be a set of literals. We define $S^+ = \{a \mid a \in S\}$, $S^- = \{a \mid \neg a \in S\}$, and $\neg.S = \{\neg a \mid a \in S\}$. S is *consistent* if $S^+ \cap S^- = \emptyset$. A *partial interpretation* S is a consistent subset of $\Sigma \cup \neg.\Sigma$. Any atom not appearing in S is said to be *undefined*. A *consistent extension* of S is an interpretation I such that $S \subseteq I \cup \neg.\bar{I}$. Note that a consistent extension here is a (two-valued) interpretation, i.e., all atoms are assigned a truth value.[7] In the sequel, we restrict Σ to the set of atoms appearing in P. This is typically called the *Herbrand base* of P and is denoted by HB_P.

[7] This is to be consistent with the notion of an extension of a partial interpretation introduced in [11].

2.3 Well-founded semantics for normal logic programs

To place our work in perspective, we briefly review the well-founded semantics for normal logic programs, which can be defined alternatively in several ways, one of which is based on the notion of unfounded set, which we adopt here.

Let P be a normal logic program and S a partial interpretation. A set $U \subseteq \Sigma$ is *an unfounded set* of P w.r.t. S, if for every $a \in U$ and every rule $r \in P$ with $H(r) = a$, either (i) $\neg b \in S \cup \neg.U$ for some $b \in Pos(r)$, or (ii) $b \in S$ for some $b \in Neg(r)$. The greatest unfounded set of P w.r.t. S always exists, which is denoted by $U_P(S)$.

Intuitively, unfounded atoms are those that can be safely assumed to be false without affecting the evaluation of the rules under the given interpretation.

We then define two operators

- $T_P(S) = \{H(r) \mid r \in P, Pos(r) \cup \neg.Neg(r) \subseteq S\}$
- $W_P(S) = T_P(S) \cup \neg.U_P(S)$

The operator W_P is monotone, and thus has a least fixpoint, which defines the well-founded semantics of P.

3 Well-Founded Semantics for Logic Programs with Generalized Atoms

In the definition of the well-founded semantics for normal logic programs, when an atom a is in a partial interpretation S, it is clear that a remains to be satisfied by all consistent extensions of S. However, this is not the case in general for a non-monotone generalized atom. We thus extend the notion of truth (resp. falsity) to the notion of *persistent truth* (resp. *persistent falsity*) under a partial interpretation.

Definition 1. *Let α be a generalized atom and S a partial interpretation.*

- *if α is an ordinary atom, it is* persistently true *(resp.* persistently false*) under S if $\alpha \in S^+$ (resp. $\alpha \in S^-$);*
- *Otherwise, α is* persistently true *(resp.* persistently false*) under S if for all consistent extensions I of S, $I \models \alpha$ (resp. $I \not\models \alpha$).*
- *$\neg\alpha$ is* persistently true *under S if α is* persistently false *under S.*
- *$\neg\alpha$ is* persistently false *under S if α is* persistently true *under S.*

Intuitively, a generalized atom α is persistently true (resp. persistently false) relative to a partial interpretation S, iff α is true (resp. false) relative to all consistent extensions of S, i.e., the truth or falsity of α remains unaffected when all undefined atoms w.r.t. S get assigned in any possible way.

The definition above naturally extends to conjunctions of generalized literals. The following definition can be seen as a paraphrase of the notion of unfounded set for logic programs with aggregates [11].

Definition 2. (Unfounded set) *Let P be a logic program with generalized atoms and S a partial interpretation. A set $U \subseteq HB_P$ is an* unfounded set of P relative to S *if for each $r \in P$ with $H(r) \in U$, some generalized literal in $B(r)$ is persistently false w.r.t. $(S \setminus U) \cup \neg.U$.*

The definition says that an atom a is in an unfounded set U, relative to S, because, for every rule with a in the head, at least one body literal is false in all consistent extensions of $(S \setminus U) \cup \neg.U$.

Definition 3. (Unfounded-free interpretation) *Let P be a logic program with generalized atoms and S a partial interpretation. S is unfounded-free, if $S \cap U = \emptyset$, for each unfounded set U of P relative to S.*

The following lemma has been proved for logic programs with aggregates [11]. The same proof can be adopted for programs with generalized atoms.

Lemma 1. *Let P be a logic program with generalized atoms and S an unfounded-free interpretation. (i) Unfounded sets of P relative to S are closed under union. (ii) The greatest unfounded set of P relative to S exists, which is the union of all unfounded sets of P relative to S.*

We now define the operators that are needed for the definition of well-founded semantics.

Definition 4. *Let P be a logic program with generalized atoms and S an unfounded free partial interpretation. Define the operators T_P, U_P, and W_P as follows:*

(i) $T_P(S) = \{H(r) \mid r \in P, B(r)$ *is persistently true under* $S\}$.
(ii) $U_P(S)$ *is the greatest unfounded set of P relative to S.*
(iii) $W_P(S) = T_P(S) \cup \neg.U_P(S)$.

As a notation, we define $W_P^0 = \emptyset$, and $W_P^{i+1} = W_P(W_P^i)$, for all $i \geq 0$. Note that W_P^0 is an unfounded-free interpretation and so is every W_P^i, for $i \geq 0$. Thus in every step the greatest unfounded set is computed relative to an unfounded-free set.

Lemma 2. *The operators T_P, U_P, and W_P are all monotone.*

Note that the operator W_P is well-defined, i.e., it is a mapping on unfounded-free partial interpretations, which, along with the subset relation, forms a complete lattice. Since W_P is monotone, it follows from the Knaster-Tarski fixpoint theorem [23] that the least fixpoint of W_P exists.

Definition 5. (Well-founded semantics) *Let P be a logic program with generalized atoms. The* well-founded semantics *(WFS) of P, denoted* $\mathrm{WFS}(P)$*, is defined as the least fixpoint of the operator W_P, denoted lfp(W_P). An atom $a \in \Sigma$ is* well-founded *(resp.* unfounded*) relative to P iff a (resp. $\neg a$) is in lfp(W_P).*

Observe that, the only difference in the operators defined here from those defined for normal logic programs is in the evaluation of body literals - being true (resp. false) has been replaced by being persistently true (resp. persistently false). It thus follows that the well-founded semantics for logic programs with generalized atoms is a generalization of the well-founded semantics for normal logic programs, and it treats monotone, anti-monotone, and non-monotone generalized atoms in a uniform manner.

Example 2. Consider program P below, where generalized atoms are aggregates.

$$r_1 \ : \ p(-1). \qquad r_2 \ : \ p(-2) \leftarrow SUM(\{X \mid p(X)\}) \leq 2.$$
$$r_3 \ : \ p(3) \leftarrow SUM(\{X \mid p(X)\}) > -4. \quad r_4 \ : \ p(-4) \leftarrow SUM(\{X \mid p(X)\}) \leq 0.$$

The aggregates under SUM are self-explaining, e.g., $SUM(\{X \mid p(X)\}) \leq 2$ means that the sum of X for satisfied atoms $p(X)$ is less than or equal to 2. We start with $W_P^0 = \emptyset$, and then $W_P^1 = \{p(-1)\}$. Observe that the body of r_2 is persistently true under W_P^1. We then have $W_P^2 = \{p(-1), p(-2)\}$, and $W_P^3 = \{p(-1), p(-2), p(-4)\}$. Now the body of r_3 is persistently false under W_P^3. So, $W_P^4 = \{p(-1), p(-2), p(-4), \neg p(3)\}$ and, $\text{WFS}(P) = \mathit{lfp}(W_P) = W_P^4$.

3.1 Complexity

Here, let us assume that for a generalized atom α, $Dom(\alpha)$ is finite and the relation $I \models \alpha$ (resp. $I \models \neg\alpha$) can be determined in polynomial time in the size of $Dom(\alpha)$. This is the case for practical aggregates in logic programming, reasoning with Horn clauses, and satisfiability testing of the combined complexity for the *DL-Lite* family of description logics [3].

In general, the problem of computing the WFS of a program is intractable since determining whether a generalized atom is persistently true or persistently false under a partial interpretation is in general intractable.

Proposition 1. *Let α be a generalized atom. Checking whether α is persistently true relevant to a partial interpretation S is Co-NP-complete.*

However, the WFS for logic programs with monotone and anti-monotone generalized atoms is tractable. Let α be a generalized atom and S a partial interpretation. To check whether α is persistently true under S, if α is monotone we test $S^+ \models \alpha$, and if α is anti-monotone we test $\Sigma \backslash S^- \models \alpha$. On the other hand, if α is monotone, then α is persistently false under S iff $\Sigma \backslash S^- \not\models \alpha$, and if α is anti-monotone, then α is persistently false under S iff $S^+ \not\models \alpha$.

As the number of distinct atoms is at most the size of P, and the greatest unfounded set can be generated incrementally in polynomial time, the following proposition can be proved in a way similar to the claim for polynomial time computation of WFS of logic programs with monotone and anti-monotone aggregates.

Proposition 2. *Let P be a logic program with monotone and anti-monotone generalized atoms. Then, $\mathit{lfp}(W_P)$ can be computed in polynomial time.*

4 Polynomial Approximation for DL-Programs

In this section we will introduce dl-programs [8, 10], show how these programs can be viewed as instances of logic programs with generalized atoms, and present a polynomial approximation to the well-founded semantics of these programs.

4.1 Description Logic Program

We assume that the reader has some familiarity with description logics (DLs) [4], which are decidable fragments of first order logic.

A *dl-program* is a combined knowledge base $KB = (L, P)$, where L is a *DL knowledge base*, which is a collection of axioms in the underlying DL, and P is a *rule base*, which is a finite set of rules of this form: $h \leftarrow A_1, ..., A_m, \neg B_1, ..., \neg B_n$, where h is an atom, and A_i and B_i are atoms or *dl-atoms*,[8] which are of the form

$$DL[S_1 op_1 p_1, \cdots, S_m op_m p_m; Q](\mathbf{t}) \tag{1}$$

in which S_i is a concept or role from the vocabulary of L, $op_i \in \{\uplus, \cup\!\!\!-, \cap\!\!\!-\}$, p_i is a predicate symbol only appearing in P whose arity matches that of S_i, and $Q(\mathbf{t})$ is called a *dl-query*, where $\mathbf{t}$ is a list of constants and Q is a concept, a role, or a concept inclusion axiom, built from the vocabulary of L.

Intuitively, $S_i \uplus p_i$ extends S_i by the extension of p_i, and $S_i \cup\!\!\!- p_i$ analogously extends $\neg S_i$; the expression $S \cap\!\!\!- p_i$ instead constrains S_i to p_i. It is clear that the operator $\cap\!\!\!-$ (which we call the *constraint operator*) may cause a dl-program to be non-monotone; a dl-atom which is free of the constraint operator is monotone.

In this paper, we assume that a dl-program is ground, and define the *Herbrand base* of P, denoted HB_P, to be the set of all ground atoms $p(\mathbf{t})$, where p appears in P and $\mathbf{t}$ is a tuple of constants. Interpretations are subsets of HB_P.

Definition 6. *Let $KB = (L, P)$ be a dl-program and $I \subseteq HB_P$ an interpretation. We 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*

$$A_i = \begin{cases} \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \textit{if } op_i = \uplus\,; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \textit{if } op_i = \cup\!\!\!-; \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \notin I\}, & \textit{if } op_i = \cap\!\!\!-. \end{cases}$$

Intuitively, dl-programs can be seen as instances of logic programs with generalized atoms, if each dl-atom is mapped to a generalized atom while preserving its semantics. To make this precise, let us define a mapping ξ from dl-programs to programs with generalized atoms.

Definition 7. *Let $KB = (L, P)$ be a dl-program, and $\xi(KB)$ denote the set of rules obtained from KB in the following way: for each $r \in P$, we have $r' \in \xi(KB)$ such that r' is r except that each dl-atom g appearing in r is replaced by a generalized atom α_g such that*

- *$Dom(\alpha_g)$ is the set of atoms appearing in P, and*

[8] For simplicity, we assume that equality does not appear in rules.

– *We identify Σ with HB_P, and define the semantics of α_g as: for all interpretations I, $I \models_L g$ iff $I \models \alpha_g$.*

That is, ξ is a transformation that preserves satisfiability under $\models_L$.

Definition 8. *Let KB $= (L, P)$ be a dl-program. The well-founded semantics (WFS) of KB is defined in terms of the well-founded semantics of $\xi(KB)$.*

Example 3. Consider a dl-program $KB = (\emptyset, P)$, where P consists of

$$\begin{array}{ll} r_1: & p(a) \leftarrow \neg DL[S_1 \cap\!\!\!\!-\, q, S_2 \uplus r; \neg S_1 \sqcap S_2](a). \\ r_2: & q(a) \leftarrow DL[S \uplus q; S](a). \\ r_3: & r(a) \leftarrow DL[S \cap\!\!\!\!-\, q; \neg S](a). \end{array}$$

Let $P' = \xi(KB)$. We have $W^0_{P'} = \emptyset$. Next, consider r_1 and the only dl-atom in it. Let $I = \{r(a)\}$, which is a consistent extension of $W^0_{P'}$. By $S_1 \cap\!\!\!\!-\, q$ in the dl-atom, that $q(a) \notin I$ leads to $\neg S_1(a)$; similarly $r(a) \in I$ leads to $S_2(a)$. Thus, the query $Q[a] = \neg S_1(a) \sqcap S_2(a)$ is satisfied by $\{\neg S_1(a), S_2(a)\}$. Hence $DL[S_1 \cap\!\!\!\!-\, q, S_2 \uplus r; \neg S_1 \sqcap S_2](a)$ evaluates to true under I and its negation to false. As $\neg DL[S_1 \cap\!\!\!\!-\, q, S_2 \uplus r; \neg S_1 \sqcap S_2](a)$ is not persistently true, we do not derive $p(a)$. But $\{q(a)\}$ is an unfounded set relative to $\emptyset$, since the dl-atom in the body of rule r_2 is persistently false relative to all consistent extensions of $W^0_{P'} \cup \{\neg q(a)\}$. The reader can verify that $W^1_{P'} = \{\neg q(a)\}$, $W^2_{P'} = \{\neg q(a), r(a)\}$, and $W^3_{P'} = \{\neg q(a), \neg p(a), r(a)\}$, which is the least fixpoint of $W_{P'}$.

Due to the one-to-one correspondence between the satisfaction under $\models_L$ and that under $\models$, in the following, we may refer to the WFS of a dl-program directly without applying the mapping ξ explicitly. The WFS of a dl-program *KB* is thus denoted by $lfp(W_{KB})$. We can then apply the notion of persistent truth and falsity directly to dl-programs. In this way, it can be shown easily that the well-founded semantics defined in [10] for dl-programs with dl-atoms that are free of the constraint operator coincides with the well-founded semantics of the corresponding logic program with generalized atoms.

4.2 Removing non-monotone dl-atoms as a polynomial approximation

Clearly, non-monotone dl-atoms are the result of applying the constraint operator $\cap\!\!\!\!-$, as the satisfiability of such atoms depends on truth value of propositional atoms exclusive of a given interpretation. In [10], the authors suggest a polynomial time rewrite to remove the constraint operator.

Definition 9. *(Transformation π) Let KB $= (L, P)$ be a dl-program. $\pi(KB)$ is a dl-program obtained from KB by*

1. *replacing each occurrence of $S_i \cap\!\!\!\!-\, p_i$ with $S_i \uplus \bar{p}_i$ ($\bar{p}_i$ is the complement of p_i), and*
2. *for each atom $p_i(\mathbf{t})$ that occurs in the head of some rule in P, add the following rule*
$$\bar{p}_i(\mathbf{t}) \leftarrow \neg DL[S'_i \uplus p_i; S'_i](\mathbf{t})$$
where S'_i is a fresh concept or role name.

Note that π is a polynomial transformation.

Since the transformation π does not affect DL knowledge base L in *KB*, we may write $\pi(P)$ to denote the set of all transformed rules, and $\pi(r)$ to denote the set of transformed rules for $r \in P$. By the transformation π, any dl-program $\mathit{KB} = (L, P)$ can be rewritten to a dl-program $\pi(\mathit{KB})$ free of the constraint operator, and thus $\pi(\mathit{KB})$ only contains monotone dl-atoms (then for such a dl-atom α, $\neg\alpha$ is anti-monotone). By Proposition 2, we know that, if query answering with the underlying DL is tractable, then the WFS of $\pi(\mathit{KB})$ can be computed in polynomial time.

The question is whether the transformation π is faithful. That is, whether it is the case that for all dl-programs *KB*, the WFS of *KB* is equivalent to the WFS of $\pi(\mathit{KB})$, barring freshly added concept/role names in $\pi(\mathit{KB})$. The following example shows that the answer to this question is no.

Example 4. Consider a single-rule dl-program $\mathit{KB} = (\emptyset, P)$,[9] where P consists of a single rule

$$p(a) \leftarrow DL[S \cup\!\!\!\!- \, p, S \cap\!\!\!\!- \, p; \neg S](a).$$

The WFS of *KB* is $\{p(a)\}$. On the other hand, $\pi(P)$ consists of two rules

$$p(a) \leftarrow DL[S \cup\!\!\!\!- \, p, S \cup\!\!\!\!- \, \bar{p}; \neg S](a). \qquad \bar{p}(a) \leftarrow \neg DL[S' \uplus p; S'](a).$$

Clearly, the WFS of $\pi(\mathit{KB})$ is $\emptyset$, as neither $p(a)$ nor $\bar{p}(a)$ is derivable under $\emptyset$, and neither is unfounded relative to $\emptyset$.

However, we can show that the transformation π is correct, i.e., given a dl-program *KB*, when restricted to the language of *KB*, all the well-founded (resp. unfounded) atoms relative to $\pi(\mathit{KB})$ are well-founded (resp. unfounded) relative to *KB*.

Let $\mathit{KB} = (L, P)$ be a dl-program and $\tau(\mathit{KB})$ its signature. We denote the WFS of $\pi(\mathit{KB})$, restricted to $\tau(\mathit{KB})$, by $\mathrm{WFS}(\pi(\mathit{KB}))|_{\tau(\mathit{KB})}$ (similarly, $\mathit{lfp}(W_{\pi(\mathit{KB})})|_{\tau(\mathit{KB})}$). For simplicity, we just write τ for $\tau(\mathit{KB})$.

We now give the main theorem of this section.

Theorem 1. *Let* $\mathit{KB} = (L, P)$ *be a dl-program. Then* $\mathrm{WFS}(\pi(\mathit{KB}))|_{\tau} \subseteq \mathrm{WFS}(\mathit{KB})$.

5 Polynomial Approximation for Logic Programs with Aggregates

Aggregates can be viewed as special cases of generalized atoms. In this case, logic programs with aggregates are instances of logic programs with generalized atoms. In this section, we show an incomplete method for approximating the well-founded semantics for these programs.

5.1 Syntax and semantics of logic programs with aggregates

Following [21], an *aggregate* (or *aggregate atom*) is a constraint on a set of atoms taking the form

$$aggr(\{X \mid p(X)\}) \ \mathtt{op} \ \mathit{Result} \tag{2}$$

[9] This example was originally provided by Yisong Wang.

where $aggr$ is an *aggregate function*. The standard aggregate functions are those in $\{$SUM, COUNT, AVG, MAX, MIN$\}$. The set $\{X|p(X)\}$ is called an *intensional set*, where p is a predicate, and X is a variable which takes value from a set $D(X) = \{a_1, ..., a_n\}$, called the *variable domain*. The relational operator op is from $\{=, \neq, <, >, \leq, \geq\}$ and $Result$ is a numeric constant.

The *domain* of an aggregate A, denoted $Dom(A)$, is the set of atoms $\{p(a) \mid a \in D(X)\}$. The size of an aggregate is $|Dom(A)|$.

For an aggregate A, the intensional set $\{X|p(X)\}$, the variable domain $D(X)$, and the domain of an aggregate $Dom(A)$ can also be a multiset which may contain duplicate members. Since multiple occurrences of an atom in a multiset can be represented by distinct symbols whose equivalence can be reinforced by adding simple normal rules, in the following, we only discuss the case of domain being a set.

Let I be an interpretation. I is a *model* of (*satisfies*) an aggregate A, denoted $I \models A$, if $aggr(\{a \mid p(a) \in I \cap Dom(A)\})$ op $Result$ holds, otherwise I is not a model of (does not satisfy) A, denoted $I \not\models A$.

For instance, consider the aggregate $A = SUM(\{X|p(X)\}) \geq 2$, where $D(X) = \{-1, 1, 1, 2\}$. For the sets $I_1 = \{p(2)\}$ and $I_2 = \{p(-1), p(1)\}$, we have $I_1 \models A$ and $I_2 \not\models A$.

A *logic program with aggregates* (or an *aggregate logic program*) is a finite set of rules of this form: $h \leftarrow A_1, ..., A_k, \neg B_1, ..., \neg B_m, G_1, ..., G_n$,where h, A_i and B_j are atoms and G_i are aggregates.

We then can define a mapping from from aggregate programs to programs with generalized atoms, in exactly the same way as in Definition 7. Namely, the domain of an aggregate A is the domain of the corresponding generalized atom g_A and the satisfiability of g_A is identical to that of A.

5.2 Disjunctive rewrite as a polynomial approximation

To optimize programs with constraint atoms, in [14], replacement techniques are studied, where a complex constraint may be decomposed into simpler ones. In one replacement scheme, the authors propose to rewrite a program with disjunctive encoding for c-atoms under the answer set semantics. The idea is to encode a complex c-atom by a disjunction of simpler c-atoms. We apply this idea to aggregates.

A *disjunctive encoding* of an aggregate A is a disjunction of aggregates A_i $(1 \leq i \leq m)$, denoted by $d(A_1, \ldots, A_m)$, such that for any interpretation I, $I \models A$ iff $I \models A_i$ $(1 \leq i \leq m)$. That is, disjunctive encoding preserves satisfaction.

In [21], the authors show that the determination of persistent truth of an aggregate atom involving SUM/AVG and $\neq$ at the same time is intractable, while determining the same for all other aggregate atoms is tractable. Now, by definition, an aggregate atom A is persistently false under S iff the complement of A is persistently true under S. As a result, determining *persistent falsity* of SUM/AVG involving the $=$ operator is also intractable. Thus the goal of disjunctive rewrite for aggregate logic programs is to transform away aggregates of the form $f(.) \neq c$ for computing well-founded atoms and $f(.) = c$ for computing unfounded atoms, where $f \in \{SUM, AVG\}$.

Definition 10. (**Disjunctive rewrite**) *Let P be a logic program with aggregates. The disjunctive rewrite of P produces two programs, one for polynomial time computation* *of well-founded atoms of P, denoted as P_w and the other for polynomial time computation of unfounded atoms of P, denoted as P_u. We define P_w as:*

> *For each occurrence of aggregate atom of the form $f(.) \neq c$ in P, where $f \in \{SUM, AVG\}$, we replace that atom with a unique symbol α and add the following two rules: $\alpha \leftarrow f(.) > c$ and $\alpha \leftarrow f(.) < c$.*

and define P_u as:

> *For each occurrence of aggregate atom of the form $f(.) = c$ in P, where $f \in \{SUM, AVG\}$, we replace that atom with the conjunction of two aggregates, $f(.) \leq c$ and $f(.) \geq c$.*

By an abuse of notation, let us denote the pair of programs P_w and P_u by $P_{(w,u)}$. Now, we revise the definition of the operator W_P of Definition 4 as follows:

$$W_{P_{(w,u)}}(S) = T_{P_w}(S) \cup \neg.U_{P_u}(S).$$

It can be shown that the operator $W_{P_{(w,u)}}$ is monotone, thus its least fixpoint can be computed iteratively. Again, let us denote by $\mathrm{WFS}(P_{(w,u)})$ the least fixpoint of the operator $W_{P_{(w,u)}}$.

The following example shows that the disjunctive rewrite is an incomplete method.

Example 5. Consider the following aggregate logic program P

$$p(2) \leftarrow SUM(\{X|p(X)\}) \neq -1. \qquad p(-3) \leftarrow p(2). \qquad p(1) \leftarrow .$$

where $HB_P = \{p(1), p(2), p(-3)\}$. $\mathrm{WFS}(P)$ is computed as follows: $W_P^0 = \emptyset$, $W_P^1 = \{p(1)\}$, $W_P^2 = \{p(1), p(2)\}$, and $W_P^3 = \{p(1), p(2), p(-3)\}$, which is the least fixpoint of W_P.

By disjunctive rewrite, we have P_w below

$$\begin{array}{lll} p(2) \leftarrow \alpha. & p(-3) \leftarrow p(2). & p(1) \leftarrow . \\ \alpha \leftarrow SUM(\{X : p(X)\}) > -1. & & \alpha \leftarrow SUM(\{X : p(X)\}) < -1. \end{array}$$

As P contains no aggregate atom of the form $f(.) = c$, we have $P_u = P$. Then, we have $T_{P_w}(\emptyset) = \{p(1)\}$ and $U_{P_u}(\emptyset) = \emptyset$, thus $W_{P_{(w,u)}}(\emptyset) = \{p(1)\}$. It can be verified easily that this is a fixpoint of $W_{P_{(w,u)}}$, i.e., $\mathrm{WFS}(P_{(w,u)}) = \{p(1)\}$.

It can be seen that disjunctive rewrite produces stronger constraints. To show this, let us extend the notion of persistent truth and falsity to disjunction of literals in a natural way. Then, we can see that, given a disjunction of aggregates, say $(f(.) < c) \vee (f(.) > c)$, and a partial interpretation S, the fact that $f(.) < c$ is persistently true under S or $f(.) > c$ is persistently true under S implies that $(f(.) < c) \vee (f(.) > c)$ is persistently true under S, but the converse does not hold in general - that $(f(.) < c) \vee (f(.) > c)$ is satisfied by all consistent extensions of S does not imply that $f(.) < c$ is persistently true under S or $f(.) > c$ is persistently true under S, because it may be due to that some consistent extensions satisfy $f(.) < c$ and others satisfy $f(.) > c$.

Similarly, for the computation of unfounded set, that $f(.) = c$ is persistently false iff $f(.) \neq c$ is persistently true if either $f(.) > c$ is persistently true or $f(.) < c$ is persistently true iff either $f(.) \leq c$ is persistently false or $f(.) \geq c$ is persistently false. The converse for the if statement above does not hold because that $f(.) = c$ is persistently false may be due to the fact that for some consistent extensions $f(.) < c$ holds and for the others $f(.) > c$ holds.

The above arguments actually give a proof sketch of the following theorem.

Theorem 2. *Let P be a logic program with aggregates. Then,* $\mathrm{WFS}(P_{(w,u)})|_{\tau(P)} \subseteq \mathrm{WFS}(P)$*, where $\tau(P)$ denotes the original language of P.*

6 Related Work and Discussion

The well-founded semantics defined in this paper for logic programs with generalized atoms is based on essentially the same notion of unfounded set formulated in [11]. By the work of [18], for logic programs with aggregates, this well-founded semantics is known to approximate answer sets based on the notion of *conditional satisfaction* [22]. This well-founded semantics is different from that of [25], which approximates *answer sets by reduct* [22]. The WFS of [25] is weaker than the WFS defined here, but without any reduction on complexity.

In [13], the authors present a well-founded semantics for hybrid Minimal Knowledge and Negation as Failure (MKNF) knowledge bases, which is a local closed world extension of the MKNF DL knowledge base. The well-founded semantics defined in [13] is shown to be tractable, if the chosen DL fragment is tractable. As shown in Proposition 1 of this paper, even if we assume the entailment relation $I \models \phi$ is tractable, for interpretation I and generalized atom ϕ, computing the WFS is still not. This is inevitable since classic formulas under the scope of negation are anti-monotone while generalized atoms may be neither monotone nor anti-monotone. The precise relationship between the MKNF WFS and the WFS defined here requires further study.

If we assume that the domain of a generalized atom is bounded [2] the well-founded semantics can be computed in polynomial time. This assumption is reasonable only for generalized atoms with small domains.

For improving propagation efficiency for HEX-programs, a decision criterion is introduced in [7] to allow to decide if further check is necessary (with the external sources) to complete the computation of the Unfounded Set (UFS) of the guessing program Q obtained from a given HEX-program P w.r.t. an interpretation A. The decision criterion is as follows: is there any atom dependency cycle that exists in P, which contains external edges (e-cycle)? Following this decision criterion, the authors devise a program decomposition technique which decomposes a given HEX-program into two types of components - one type of component is with e-cycles and other type of component does not have e-cycles. UFS checking is needed only for the components which do have e-cycles. Thus this technique avoids UFS checking when it is not necessary. This work however does not prevent complexity jump in constraint propagation. The decision criterion reduces computational cost linearly. In our case studies, we focus on subtle aspects of computation that cause complexity jump, which may be avoided by incomplete methods. We establish the links between such incomplete methods with the well-founded semantics of the underlying logic program.

We wonder whether the idea of incomplete methods can be pursued for HEX-programs in general. If yes, it will be interesting to study characterizations of the type of information that may be captured, or lost, by such an approximation.

Many practical logic programs use weight constraints, which are essentially the SUM aggregates. It would be interesting to see whether the methods proposed in this paper are applicable to some of these programs, and if yes, how the propagators for weight constraints of the currently available ASP systems (such as clasp[10]) can benefit from disjunctive rewrite presented in this paper.

References

1. Mario Alviano, Francesco Calimeri, Wolfgang Faber, Nicola Leone, and Simona Perri. Unfounded sets and well-founded semantics of answer set programs with aggregates. *J. Artif. Intell. Res.*, 42:487–527, 2011.
2. Mario Alviano and Wolfgang Faber. The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In *LPNMR*, pages 67–72, 2013.
3. Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36:1–69, 2009.
4. 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.
5. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, New York, NY, 2003.
6. Lucas Bordeaux, Youssef Hamadi, and Lintao Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Comput. Surv.*, 38(4), 2006.
7. Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller. Efficient hex-program evaluation based on unfounded sets. *J. Artif. Intell. Res. (JAIR)*, 49:269–321, 2014.
8. Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artifical Intelligence*, 172(12-13):1495–1539, 2008.
9. Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *IJCAI*, pages 90–96, 2005.
10. 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.
11. W. Faber. Unfounded sets for disjunctive logic programs with arbitrary aggregates. In *Proc. LPNMR-05*, pages 40–52, 2005.
12. Wolfgang Faber, Gerald Pfeifer, Nicola Leone, Tina Dell'Armi, and Giuseppe Ielpa. Design and implementation of aggregate functions in the dlv system. *TPLP*, 8(5-6):545–580, 2008.
13. Matthias Knorr, José Júlio Alferes, and Pascal Hitzler. Local closed world reasoning with description logics under the well-founded semantics. *Artif. Intell.*, 175(9-10):1528–1554, 2011.

[10] http://www.cs.uni-potsdam.de/clasp/

14. Guohua Liu, Randy Goebel, Tomi Janhunen, Ilkka Niemelä, and Jia-Huai You. Strong equivalence of logic programs with abstract constraint atoms. In *Proc. LPNMR-11*, pages 161–173, 2011.
15. Guohua Liu and Jia-Huai You. Lparse programs revisited: Semantics and representation of aggregates. In *Proc. ICLP-08*, volume 5366 of *Lecture Notes in Computer Science*, pages 347–361, Udine, Italy, 2008. Springer.
16. Lengning Liu, Enrico Pontelli, Tran Cao Son, and Miroslaw Truszczyński. Logic programs with abstract constraint atoms: The role of computations. *Artificial Intelligence*, 174:295–315, 2010.
17. V. W. Marek and M. Truszczynski. Logic programs with abstract constraint atoms. In *Proc. AAAI-04*, pages 86–91, 2004.
18. 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.
19. F. Rossi, P. Van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*, chapter Global Constraints. Elsevier, 2006.
20. Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
21. Tran Cao Son and Enrico Pontelli. A constructive semantic characterization of aggregates in answer set programming. *TPLP*, 7(3), 2007.
22. Tran Cao Son, Enrico Pontelli, and Phan Huy Tu. Answer sets for logic programs with arbitrary abstract constraint atoms. *Journal of Artificial Intelligence Research*, 29:353–389, 2007.
23. Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.
24. 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.
25. Yisong Wang, Fangzhen Lin, Mingyi Zhang, and Jia-Huai You. A well-founded semantics for basic logic programs with arbitrary abstract constraint atoms. In *AAAI*, 2012.

A Appendix : Proof of Theorem 1

Proof. We prove the claim by induction on the construction of $\mathit{lfp}(W_{\pi(KB)})$ and $\mathit{lfp}(W_{KB})$.

(a) Base: $W^0_{\pi(KB)}|_\tau = W^0_{KB} = \emptyset$.

(b) Step: Assume, for all $k \geq 0$, $W^k_{\pi(KB)}|_\tau \subseteq W^k_{KB}$, and prove $W^{k+1}_{\pi(KB)}|_\tau \subseteq W^{k+1}_{KB}$. By definition, we know

$$W^{k+1}_{\pi(KB)}|_\tau = T_{\pi(KB)}(W^k_{\pi(KB)})|_\tau \cup \neg U_{\pi(KB)}(W^k_{\pi(KB)})|_\tau \tag{3}$$

$$W^{k+1}_{KB} = T_{KB}(W^k_{KB}) \cup \neg.U_{KB}(W^k_{KB}) \tag{4}$$

To prove $W^{k+1}_{\pi(KB)}|_\tau \subseteq W^{k+1}_{KB}$, it is sufficient to prove both of

$$T_{\pi(KB)}(W^k_{\pi(KB)})|_\tau \subseteq T_{KB}(W^k_{KB}) \tag{5}$$

$$U_{\pi(KB)}(W^k_{\pi(KB)})|_\tau \subseteq U_{KB}(W^k_{KB}) \tag{6}$$

Below, let us assume that at most one dl-atom may appear in a rule. The proof can be generalized to arbitrary rules by the same argument, for the transformation of one dl-atom at a time.

(i) We first prove (5). Let $a \in T_{\pi(KB)}(W^k_{\pi(KB)})|_\tau$. By definition, $\exists r' \in \pi(P)$ such that $B(r')$ is persistently true under $W^k_{\pi(KB)}$. WLOG, assume for some $r \in P$, $\pi(r) = \{r', r''\}$, as illustrated in (7) below, in which a dl-atom appears positively, which is replaced by rules in (8) of $\pi(r)$.

$$r:\ a \leftarrow ..., DL[..., S_j \mathbin{\cap\!\!\!\!\!-} p_j, ...; Q](e), ... \tag{7}$$

$$\begin{aligned} r' &:\ a \leftarrow ..., DL[..., S_j \uplus \bar{p_j}, ...; Q](\mathbf{e}), ... \\ r'' &:\ \bar{p_j}(\mathbf{e}) \leftarrow \neg DL[S'_j \uplus p_j; S'](\mathbf{e}) \end{aligned} \tag{8}$$

Let D denote the dl-atom in (7), and D' the corresponding dl-atom in (8). If the operator $\mathbin{\cap\!\!\!\!\!-}$ does not occur in D, then trivially $B(r)$ is persistently true under W^k_{KB}, and thus $a \in T_{KB}(W^k_{KB})$.

Otherwise, since $B(r')$ is persistently true under $W^k_{\pi(KB)}$, we have D' is persistently true under $W^k_{\pi(KB)}$, and by the fact that D' is monotone, we have $W^k_{\pi(KB)} \models_L D'$. The atom $\bar{p_j}(\mathbf{e})$ may or may not play a role in the entailment $L \cup \bigcup_{i=1}^m D'_i \models Q(\mathbf{e})$ (cf. Definition (6)). The proof is trivial if it does not. Otherwise, $\bar{p_j}(\mathbf{e})$ is well-founded already w.r.t. $W^k_{\pi(KB)}$, and by the last rule in (8), $p_j(\mathbf{e})$ is unfounded w.r.t. $W^{k'}_{\pi(KB)}$, for some $k' < k$. By induction hypothesis, we know $W^{k'}_{\pi(KB)}|_\tau \subseteq W^{k'}_{KB}$, thus $p_j(\mathbf{e})$ is unfounded w.r.t. W^k_{KB}. It follows D is persistently true under W^k_{KB}, and by the assumption that D is the only dl-atom in (7) and that $B(r')$ is persistently true under $W^k_{\pi(KB)}$, we have $B(r)$ is persistently true under W^k_{KB}. Thus, $a \in T_{KB}(W^k_{KB})$.

If D appears negatively in rule body, the proof is similar because, given a partial interpretation S, $\neg D$ is persistently true under S iff D is persistently false under S, and we just need to swap *well-founded* and *unfounded* in the argument above.

(ii) To prove (6), assume that $a \in U_{\pi(KB)}(W^k_{\pi(KB)})|_\tau$ and we show $a \in U_{KB}(W^k_{KB})$. Consider the case of (7). WLOG, assume that r is the only rule in P with a in the head. If the fact $a \in U_{\pi(KB)}(W^k_{\pi(KB)})$ is independent of $\bar{p_j}(\mathbf{e})$, the proof is trivial. Otherwise, that $a \in U_{\pi(KB)}(W^k_{\pi(KB)})$ is because D' is persistently false under $\neg.U' \cup W^k_{\pi(KB)}$, where U' is the greatest unfounded set relative to $W^k_{\pi(KB)}$. This implies that $\bar{p_j}(\mathbf{e})$ must be unfounded w.r.t. $W^k_{\pi(KB)}$, and it follows that $p_j(\mathbf{e})$ is well-founded already w.r.t. $W^{k'}_{\pi(KB)}$, for some $k' < k$. Then, by induction hypothesis, we have that $p_j(\mathbf{e})$ is well-founded w.r.t. W^k_{KB}, which implies that $B(r)$ is persistently false under $\neg.U \cup W^k_{KB}$, where U is the greatest unfounded set w.r.t. W^k_{KB}. It follows $a \in U_{KB}(W^k_{KB})$. The proof for the case where a dl-atom appears negatively in rule body is similar.

Hence, the proof is completed. $\square$