Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Grounded Fixpoints

Bart Bogaerts
Department of Computer Science
KU Leuven
3001 Heverlee, Belgium
bart.bogaerts@cs.kuleuven.be

Joost Vennekens
Department of Computer Science
Campus De Nayer, KU Leuven
2860 Sint-Katelijne-Waver, Belgium
joost.vennekens@cs.kuleuven.be

Marc Denecker
Department of Computer Science
KU Leuven
3001 Heverlee, Belgium
marc.denecker@cs.kuleuven.be

Abstract

Algebraical fixpoint theory is an invaluable instrument for studying semantics of logics. For example, all major semantics of logic programming, autoepistemic logic, default logic and more recently, abstract argumentation have been shown to be induced by the different types of fixpoints defined in approximation fixpoint theory (AFT). In this paper, we add a new type of fixpoint to AFT: a *grounded fixpoint* of lattice operator $O : L \to L$ is defined as a lattice element $x \in L$ such that $O(x) = x$ and for all $v \in L$ such that $O(v \wedge x) \leq v$, it holds that $x \leq v$. On the algebraical level, we show that all grounded fixpoints are minimal fixpoints approximated by the well-founded fixpoint and that all stable fixpoints are grounded. On the logical level, grounded fixpoints provide a new mathematically simple and compact type of semantics for any logic with a (possibly non-monotone) semantic operator. We explain the intuition underlying this semantics in the context of logic programming by pointing out that grounded fixpoints of the immediate consequence operator are interpretations that have no non-trivial *unfounded sets*. We also analyse the complexity of the induced semantics.

Summarised, grounded fixpoint semantics is a new, probably the simplest and most compact, element in the family of semantics that capture basic intuitions and principles of various non-monotonic logics.

1 Introduction

Motivated by structural analogies in the semantics of several non-monotonic logics, Denecker, Marek, and Truszczyński (2000) developed an algebraic theory that defines different types of fixpoints for a so-called approximating bilattice operator, called supported, Kripke-Kleene, stable and well-founded fixpoints. In the context of logic programming, they found that Fitting's immediate consequence operator is an approximating operator of the two-valued immediate consequence operator and that its four different types of fixpoints correspond exactly with the four major, equally named semantics of logic programs. They also identified approximating operators for default logic (DL) and autoepistemic logic (AEL) and showed that the fixpoint theory induces all main and some new semantics in these fields (Denecker, Marek, and Truszczyński 2003). By showing that Konolige's mapping from DL to AEL preserves the approximating operator, they resolved an old research question regarding the nature of these two logics: AEL and DL are "just" two different dialects of autoepistemic reasoning (Denecker, Marek, and Truszczyński 2011).

The study of these approximating operators is called approximation fixpoint theory (AFT). It is now commonly used to define semantics of extensions of logic programs, such as logic programs with aggregates (Pelov, Denecker, and Bruynooghe 2007) and HEX logic programs (Antic, Eiter, and Fink 2013). Vennekens, Gilis, and Denecker (2006) used AFT in an algebraic modularity study for logic programming, AEL and DL. Recently, Strass (2013) showed that many semantics from Dung's argumentation frameworks (AFs) and abstract dialectical frameworks (ADFs) can be obtained by direct applications of AFT and Bogaerts et al. (2014) defined the causal logic FO(C) as an instantiation of AFT. This work suggests that fixpoint theory, despite its high level of abstraction, captures certain fundamental intuitions and cognitive principles in a range of logics and sorts of human knowledge. It is this observation that provides the basic motivation for the present study.

In Section 3, we extend AFT with a new type of fixpoint: a point x in a lattice L is a *grounded fixpoint* of operator $O : L \to L$ if $O(x) = x$ and for all $v \in L$ such that $O(x \wedge v) \leq v$, it holds that $x \leq v$. In Section 4, we discuss the relation between grounded fixpoints and the other fixpoints defined by AFT. In particular, we show that all (ultimate) stable fixpoints are grounded and that all grounded fixpoints are minimal fixpoints approximated by the (ultimate) well-founded fixpoint in the bilattice. In general there are minimal fixpoints that are not grounded, and grounded fixpoints that are not stable. If the well-founded fixpoint is "exact", the well-founded fixpoint is the unique grounded and stable fixpoint of O.

Grounded fixpoints have several appealing properties. First of all, a grounded fixpoint is a purely algebraical concept. As such, it can be used in all fields where AFT is applied. Secondly, a first step in the application of AFT for a lattice operator O is to choose a bilattice operator that approximates O. In contrast, grounded fixpoints are defined directly in terms of the original operator O. Thirdly, their definition formalises and generalises well-known intuitions.

In the context of logic programming, which we discuss in Section 5, the algebraic results show that grounded fixpoints induce a semantics that is slightly more "liberal" than stable semantics: all stable models are grounded (i.e., we identified a property all stable models have in common) but also every grounded fixpoint is approximated by the well-founded model; the differences collapse in case the well-founded model is two-valued. We will see that for logic programming, this semantics is simple and intuitive: we show that the grounded fixpoints can be characterised in terms of a generalised notion of *unfounded set*. Contrary to the more common semantics of logic programs, grounded fixpoint semantics does not rely on any form of three-valued logic: It is defined directly in terms of the (two-valued!) immediate consequence operator. The grounded fixpoint semantics is very flexible towards language extensions. Currently, much research is being conducted in order to extend stable and well-founded semantics for logic programs with new language constructs (Pelov, Denecker, and Bruynooghe 2007; Faber, Pfeifer, and Leone 2011; Marek, Niemelä, and Truszczyński 2008; Balduccini 2013). Since the grounded fixpoint semantics is completely defined using the two-valued immediate consequence operator, it suffices to extend this operator to obtain an extended semantics; this is often trivial. These nice properties come at a cost: we show that in general, determining whether a logic program has a grounded fixpoint is Σ_2^P-complete. However, for large classes of programs, grounded fixpoint semantics coincides with stable semantics. For those programs, we obtained a simple, concise, purely 2-valued and algebraical, extensible reformulation of the existing semantics.

Due to space limitations, we do not elaborate on how our theory applies to AEL, DL or ADFs and refer to (Bogaerts, Vennekens, and Denecker 2014) for proofs.

Summarised, the main contributions of this paper are as follows. We extend AFT with the notion of a grounded fixpoint, a fixpoint closely related to stable fixpoints with similar properties, but that is determined by O, not by the choice of an approximator. Applied to logic programming this yields an intuitive, purely two-valued, semantics that is easily extensible and that formalises well-known intuitions.

2 Preliminaries

2.1 Lattices and Operators

A *poset* $\langle L, \leq \rangle$ is a set L equipped with a partial order $\leq$, i.e., a reflexive antisymmetric, transitive relation. If S is a subset of L, then x is an *upper bound*, respectively a *lower bound* of S if for every $s \in S$, it holds that $s \leq x$ respectively $x \leq s$. An element x is a *least upper bound*, respectively *greatest lower bound* of S if it is an upper bound that is smaller than every other upper bound, respectively a lower bound that is greater than every other lower bound. If S has a least upper bound, respectively a greatest lower bound, we denote it $\text{lub}(S)$, respectively $\text{glb}(S)$. As is custom, we sometimes call a greatest lower bound a *meet*, and a least upper bound a *join* and use the related notations $\bigwedge S = \text{glb}(S)$, $x \wedge y = \text{glb}(\{x, y\})$, $\bigvee S = \text{lub}(S)$ and $x \vee y = \text{lub}(\{x, y\})$.

We call $\langle L, \leq \rangle$ a *complete lattice* if every subset of L has a least upper bound and a greatest lower bound. A complete lattice has both a least element $\bot$ and a greatest element $\top$.

An operator $O : L \to L$ is *monotone* if $x \leq y$ implies that $O(x) \leq O(y)$. An element $x \in L$ is a *prefixpoint*, a *fixpoint*, a *postfixpoint* if $O(x) \leq x$, respectively $O(x) = x$, $x \leq O(x)$. Every monotone operator O in a complete lattice has a least fixpoint, denoted $\text{lfp}(O)$, which is also O's least prefixpoint and the limit (the least upper bound) of the increasing sequence $(x_i)_{i \geq 0}$ defined by

- $x_0 = \bot$,
- $x_{i+1} = O(x_i)$,
- $x_\lambda = \text{lub}(\{x_i \mid i < \lambda\})$, for limit ordinals λ.

2.2 Logic Programming

In the following sections, we illustrate our abstract results in the context of logic programming. We recall some preliminaries. We restrict ourselves to propositional logic programs, but allow arbitrary propositional formulas in rule bodies. However, AFT has been applied in a much broader context (Denecker, Bruynooghe, and Vennekens 2012; Pelov, Denecker, and Bruynooghe 2007; Antic, Eiter, and Fink 2013) and our results can also be applied in these extensions of logic programming.

Let Σ be an alphabet, i.e., a collection of symbols which are called *atoms*. A *literal* is an atom p or the negation $\neg q$ of an atom q. A logic program $\mathcal{P}$ is a set of *rules* r of the form $h \leftarrow \varphi$, where h is an atom called the *head* of r, denoted $\mathit{head}(r)$, and φ is a propositional formula called the *body* of r, denoted $\mathit{body}(r)$. An interpretation I of the alphabet Σ is an element of 2^Σ, i.e., a subset of Σ. The set of interpretations 2^Σ forms a lattice equipped with the order $\subseteq$. The truth value ($\mathbf{t}$ or $\mathbf{f}$) of a propositional formula φ in a structure I, denoted φ^I is defined as usual. With a logic program $\mathcal{P}$, we associate an immediate consequence operator (van Emden and Kowalski 1976) $T_\mathcal{P}$ that maps a structure I to

$$T_\mathcal{P}(I) = \{p \mid \exists r \in \mathcal{P} : \mathit{head}(r) = p \wedge \mathit{body}(r)^I = \mathbf{t}\}.$$

3 Grounded Fixpoints

Let $\langle L, \leq \rangle$ be a complete lattice and $O : L \to L$ a lattice operator, fixed throughout this entire section. We start by giving the most central definition of this text, namely the notion of groundedness.

Definition 3.1 (Grounded). We call $x \in L$ *grounded* for O if for each $v \in L$ such that $O(v \wedge x) \leq v$, it holds that $x \leq v$.

The intuition behind this concept is easiest to explain if we assume that the elements of L are sets of "facts" of some kind and the $\leq$ relation is the subset relation between such sets. In this case, a point x is grounded if it contains only facts that are sanctioned by the operator O, in the sense that if we remove them from x, then the operator will add at least one of them again. The above definition captures this idea, by using a set $v \in L$ to remove all elements not in v from x.

If their removal is not contradicted by O, i.e., O does not re-derive any removed element ($O(x \wedge v) \leq v$), then these elements cannot be part of the grounded point ($x \leq v$).

Proposition 3.2. *If O is a monotone operator and x is grounded for O then x is a postfixpoint of O that is less than or equal to* $\mathrm{lfp}(O)$*, i.e.,* $x \leq O(x)$ *and* $x \leq \mathrm{lfp}(O)$.

Example 3.3. The converse of Proposition 3.2 does not hold. Consider the following logic program $\mathcal{P}$:

$$\left\{ \begin{array}{l} p. \\ q \leftarrow p \vee q. \end{array} \right\}$$

Its immediate consequence operator $T_{\mathcal{P}}$ is represented by the following graph, where full edges express the order relation (to be precise, the $\leq$ relation is the reflexive transitive closure of these edges) and the dotted edges represent the operator:

$$\begin{tikzcd}
& \top = \{p,q\} \arrow[loop below, dotted] & \\
\{p\} \arrow[ur] \arrow[ur, dotted, bend left] & & \{q\} \arrow[ul] \arrow[ul, dotted, bend right] \\
& \bot = \emptyset \arrow[ul] \arrow[ur] \arrow[ul, dotted, bend left] &
\end{tikzcd}$$

$T_{\mathcal{P}}$ is a monotone operator with least fixpoint $\top$. Also, $\{q\}$ is a postfixpoint of $T_{\mathcal{P}}$ since $T_{\mathcal{P}}(\{q\}) = \top \geq \{q\}$. However, $\{q\}$ is not grounded since $T_{\mathcal{P}}(\{q\} \wedge \{p\}) = T_{\mathcal{P}}(\bot) = \{p\} \leq \{p\}$, while $\{q\} \not\leq \{p\}$.

Proposition 3.4. *All grounded fixpoints of O are minimal fixpoints of O.*

Example 3.5. The converse of Proposition 3.4 does not hold. Consider the logic program $\mathcal{P}$:

$$\left\{ \begin{array}{l} p \leftarrow p. \\ q \leftarrow \neg p \vee q. \end{array} \right\}$$

This logic program corresponds has as immediate consequence operator $T_{\mathcal{P}}$:

$$\begin{tikzcd}
& \top = \{p,q\} \arrow[loop below, dotted] & \\
\{p\} \arrow[ur] \arrow[loop left, dotted] & & \{q\} \arrow[ul] \arrow[loop right, dotted] \\
& \bot = \emptyset \arrow[ul] \arrow[ur] \arrow[ur, dotted, bend left] &
\end{tikzcd}$$

In this case, $\{p\}$ is a minimal fixpoint of $T_{\mathcal{P}}$, but $\{p\}$ is not grounded since $T_{\mathcal{P}}(\{p\} \wedge \{q\}) = T_{\mathcal{P}}(\bot) = \{q\}$.

Proposition 3.6. *A monotone operator has exactly one grounded fixpoint, namely its least fixpoint.*

4 Grounded Fixpoints and AFT

4.1 Preliminaries: AFT

Given a lattice L, approximation fixpoint theory makes uses of the bilattice L^2. We define two *projection* functions for pairs as usual: $(x,y)_1 = x$ and $(x,y)_2 = y$. Pairs $(x,y) \in L^2$ are used to approximate all elements in the interval $[x,y] = \{z \mid x \leq z \wedge z \leq y\}$. We call $(x,y) \in L^2$ *consistent* if $x \leq y$, that is, if $[x,y]$ is non-empty. We use L^c to denote the set of consistent elements. Elements $(x,x) \in L^c$ are called *exact*. We sometimes abuse notation and use the tuple (x,y) and the interval $[x,y]$ interchangeably. The *precision ordering* on L^2 is defined as $(x,y) \leq_p (u,v)$ if $x \leq u$ and $v \leq y$. In case (u,v) is consistent, this means that (x,y) approximates all elements approximated by (u,v), or in other words that $[u,v] \subseteq [x,y]$. If L is a complete lattice, then $\langle L^2, \leq_p \rangle$ is also a complete lattice.

AFT studies fixpoints of lattice operators $O : L \to L$ through operators approximating O. An operator $A : L^2 \to L^2$ is an *approximator* of O if it is $\leq_p$-monotone, and has the property that for all x, $O(x) \in A(x,x)$. Approximators are internal in L^c (i.e., map L^c into L^c). As usual, we restrict our attention to *symmetric* approximators: approximators A such that for all x and y, $A(x,y)_1 = A(y,x)_2$. Denecker, Marek, and Truszczyński (2004) showed that the consistent fixpoints of interest (supported, stable, well-founded) are uniquely determined by an approximator's restriction to L^c, hence, sometimes we only define approximators on L^c.

AFT studies fixpoints of O using fixpoints of A. The A-Kripke-Kleene fixpoint is the $\leq_p$-least fixpoint of A and has the property that it approximates all fixpoints of O. A partial A-stable fixpoint is a pair (x,y) such that $x = \mathrm{lfp}(A(\cdot,y)_1)$ and $y = \mathrm{lfp}(A(x,\cdot)_2)$, where $A(\cdot,y)_1$ denotes the operator $L \to L : x \mapsto A(x,y)_1$ and analogously for $A(x,\cdot)_2$. The A-well-founded fixpoint is the least precise partial A-stable fixpoint. An *A-stable fixpoint* of O is a fixpoint x of O such that (x,x) is a partial A-stable fixpoint. This is equivalent with the condition that $x = \mathrm{lfp}(A(\cdot,x)_1)$. The A-Kripke-Kleene fixpoint of O can be constructed by iteratively applying A, starting from $(\bot,\top)$. For the A-well-founded fixpoint, a similar constructive characterisation has been worked out by Denecker and Vennekens (2007).

In general, a lattice operator $O : L \to L$ has a family of approximators of different precision. Denecker, Marek, and Truszczyński (2004) showed that there exists a most precise approximator, U_O, called the ultimate approximator of O. This operator is defined by $U_O : L^c \to L^c : (x,y) \mapsto (\bigwedge O([x,y]), \bigvee O([x,y]))$. Semantics defined using the ultimate approximator have as advantage that they only depend on O since the approximator can be derived from O. It was shown that for any approximator A, all A-stable fixpoints are U_O-stable fixpoints, and the U_O-well-founded fixpoint is always more precise than the A-well-founded fixpoint. We refer to U_O-stable fixpoints as ultimate stable fixpoints of O and to the U_O-well-founded fixpoint as the ultimate well-founded fixpoint of O.

AFT and Logic Programming In the context of logic programming, elements of the bilattice $\left(2^{\Sigma}\right)^2$ are partial interpretations, pairs $\mathcal{I} = (I_1, I_2)$ of interpretations. The pair (I_1, I_2) approximates all interpretations I' with $I_1 \subseteq I' \subseteq I_2$. We are mostly concerned with consistent (or, three-valued) interpretations: tuples $\mathcal{I} = (I_1, I_2)$ with $I_1 \subseteq I_2$. For such an interpretation, the atoms in I_1 are *true* ($\mathbf{t}$) in $\mathcal{I}$, the atoms in $I_2 \setminus I_1$ are *unknown* ($\mathbf{u}$) in $\mathcal{I}$ and the other atoms are *false* ($\mathbf{f}$) in $\mathcal{I}$. If $\mathcal{I}$ is a three-valued interpretation, and φ a formula, we write $\varphi^{\mathcal{I}}$ for the standard three-valued valuation based on the Kleene truth tables (Kleene 1938). An interpretation I corresponds to the partial interpretation (I, I). If $\mathcal{I} = (I_1, I_2)$ is a (partial) interpretation, and $U \subseteq \Sigma$, we write $\mathcal{I}[U : \mathbf{f}]$ for the (partial) interpretation that equals $\mathcal{I}$ on all elements not in U and that interprets all elements in U as $\mathbf{f}$, i.e., the interpretation $(I_1 \setminus U, I_2 \setminus U)$.

Several approximators have been defined for logic programs. The most common is Fitting's immediate consequence operator $\Psi_{\mathcal{P}}$ (Fitting 2002), a direct generalisation of $T_{\mathcal{P}}$ to partial interpretations. Denecker, Marek, and Truszczyński (2000) showed that the well-founded fixpoint of $\Psi_{\mathcal{P}}$ is the well-founded model of $\mathcal{P}$ (Van Gelder, Ross, and Schlipf 1991) and that $\Psi_{\mathcal{P}}$-stable fixpoints are exactly the stable models of $\mathcal{P}$ (Gelfond and Lifschitz 1988).

Contrary to classical stable and well-founded semantics, their ultimate counterparts have the nice property that they are insensitive to equivalence preserving rewritings of the bodies of rules. If two logic programs $\mathcal{P}$ and $\mathcal{P}'$ have the same immediate consequence operator, then their ultimate stable (respectively ultimate well-founded) models are the same. For example consider programs $\mathcal{P} = \{p \leftarrow p \lor \neg p\}$ and $\mathcal{P}' = \{p.\}$. Even though the body of the rule defining p in $\mathcal{P}$ is a tautology, $\{p\}$ is not a stable model of $\mathcal{P}$ (while it is a stable model of $\mathcal{P}'$). However, the ultimate stable semantics treats these two programs identically. However, this property comes at a cost. Denecker, Marek, and Truszczyński (2004) showed that deciding whether $\mathcal{P}$ has an ultimate stable model is Σ^2_P-complete, while that same task is only NP-complete for classical stable models.

4.2 Grounded Fixpoints and AFT

In this section, we discuss how groundedness relates to AFT. More concretely, we show that all (ultimate) stable fixpoints are grounded and that all grounded fixpoints are approximated by the (ultimate) well-founded fixpoint.

Proposition 4.1. *All ultimate stable fixpoints of O are grounded.*

Example 4.2. The converse of Proposition 4.1 does not always hold. Consider the logic program $\mathcal{P}$:

$$\left\{\begin{array}{l} p \leftarrow \neg p \lor q. \\ q \leftarrow \neg q \lor p. \end{array}\right\}$$

This logic program has as immediate consequence operator $T_{\mathcal{P}}$:

$$\begin{tikzcd}
& \top = \{p, q\} \arrow[loop below, dotted] & \\
\{p\} \arrow[ur] \arrow[rr, dotted, bend right=10] & & \{q\} \arrow[ul] \arrow[ll, dotted, bend right=10] \\
& \bot = \emptyset \arrow[ul] \arrow[ur] \arrow[uu, dotted] &
\end{tikzcd}$$

$\top$ is grounded for $T_{\mathcal{P}}$, since the only v with $T_{\mathcal{P}}(\top \land v) = T_{\mathcal{P}}(v) \leq v$ is $\top$ itself. However, since $T_{\mathcal{P}}([\bot, \top]) = L \setminus \{\bot\}$ and $\{p\} \land \{q\} = \bot$, it follows that $\bigwedge(T_{\mathcal{P}}[\bot, \top]) = \bot$. Thus, $\text{lfp}(\bigwedge T_{\mathcal{P}}([\cdot, \top])) = \bot$. Therefore, $\top$ is not an ultimate stable fixpoint of $T_{\mathcal{P}}$.

The fact that all A-stable fixpoints are ultimate stable fixpoints (Denecker, Marek, and Truszczyński 2004) yields:

Corollary 4.3. *If A is an approximator of O, then all A-stable fixpoints are grounded fixpoints of O.*

Theorem 4.4. *The well-founded fixpoint (u, v) of a symmetric approximator A of O approximates all grounded fixpoints of O.*

Corollary 4.5. *If the well-founded fixpoint of a symmetric approximator A of O is exact, then this point is the unique grounded fixpoint of O.*

5 Grounded Fixpoints of Logic Programs

In this section, we discuss grounded fixpoints in the context of logic programming. It follows immediately from the algebraical results (Corollary 4.3 and Theorem 4.4) that stable models are grounded fixpoints of the immediate consequence operator and that grounded fixpoints are minimal fixpoints approximated by the well-founded model. Grounded fixpoints can be explained in terms of unfounded sets. Intuitively, an unfounded set is a set of atoms that might circularly support themselves, but have no support from outside. Stated differently, an unfounded set of a logic program $\mathcal{P}$ with respect to an interpretation I is a set U of atoms such that $\mathcal{P}$ provides no support for the truth of any atom in U, except possibly support based on the truth of other atoms in U. Since $T_{\mathcal{P}}$ maps an interpretation I to the set of atoms supported by $\mathcal{P}$ in I, the above intuitions are directly formalised as follows.

Definition 5.1 (2-Unfounded set). Let $\mathcal{P}$ be a logic program, $T_{\mathcal{P}}$ the corresponding direct consequence operator and $I \in 2^{\Sigma}$ an interpretation. A set $U \subseteq \Sigma$ is a *2-unfounded set* of $\mathcal{P}$ with respect to I if $T_{\mathcal{P}}(I[U : \mathbf{f}]) \cap U = \emptyset$.

Thus, U is a 2-unfounded set of $\mathcal{P}$ with respect to I if removing all elements of U from I results in a state $I[U : \mathbf{f}]$ where no atom in U is supported, i.e., $T_{\mathcal{P}}(I[U : \mathbf{f}])$ contains no atoms from U. Definition 5.1 slightly differs from the original definition of unfounded set by Van Gelder, Ross, and Schlipf (1991) but it formalises the same intuitions. The most important difference is that we work in a two-valued setting, while van Gelder et al. defined unfounded sets in a three-valued setting. For clarity, we refer to our unfounded sets as "2-unfounded sets" and to the original definition as "GRS-unfounded sets". Our theory does not require any form of three-valued logic. In Section 6, we extend our definition to a three-valued context and show that the different notions of unfounded set are equivalent in the context of the well-founded model construction.

Example 5.2. Let $\mathcal{P}$ be the following program:

$$\left\{\begin{array}{l} p \leftarrow q \lor r. \\ q \leftarrow p. \\ t \leftarrow \neg s \land \neg r. \end{array}\right\}$$

Let I be the interpretation $\{p, q, s, t\}$. Then $U_1 = \{p, q\}$ is an unfounded set of $\mathcal{P}$ with respect to I since $I[U_1 : \mathbf{f}] = \{s, t\}$ and in this structure, the bodies of rules defining p and q are false. More formally, $T_{\mathcal{P}}(I[U_1 : \mathbf{f}]) \cap U_1 = \emptyset \cap U_1 = \emptyset$.

$U_2 = \{s, t\}$ is not a 2-unfounded set of $\mathcal{P}$ with respect to I since $T_{\mathcal{P}}(I[U_2 : \mathbf{f}]) \cap U_2 = \{p, q, t\} \cap U_2 = \{t\} \neq \emptyset$.

In what follows, we use U^c for the set complement of U, i.e., $U^c = \Sigma \setminus U$.

Proposition 5.3. *Let $\mathcal{P}$ be a logic program, $T_\mathcal{P}$ the corresponding direct consequence operator and $I \in 2^\Sigma$ an interpretation. A set $U \subseteq \Sigma$ is a 2-unfounded set of $\mathcal{P}$ with respect to I if and only if $T_\mathcal{P}(I \wedge U^c) \leq U^c$.*

Proposition 5.3 shows that U is a 2-unfounded set if and only if its complement satisfies the condition on v in Definition 3.1! This allows us to reformulate the condition that I is grounded as follows.

Proposition 5.4. *A structure I is grounded for $T_\mathcal{P}$ if and only if I does not contain any atoms that belong to a 2-unfounded set U of P with respect to I.*

If I is a fixpoint of $T_\mathcal{P}$, then all sets $U \subseteq I^c$ are 2-unfounded sets. We call these 2-unfounded sets *trivial*. With this terminology, we find:

Corollary 5.5. *A structure I is a grounded fixpoint of $T_\mathcal{P}$ if and only if it is a fixpoint of $T_\mathcal{P}$ and $\mathcal{P}$ has no non-trivial 2-unfounded sets with respect to I.*

Similarly to ultimate semantics, grounded fixpoints are insensitive to equivalence-preserving rewritings in the bodies of rules: if $\mathcal{P}$ and $\mathcal{P}'$ are such that $T_\mathcal{P} = T_{\mathcal{P}'}$, then the grounded fixpoints of $\mathcal{P}$ and $\mathcal{P}'$ coincide. Also similar to ultimate semantics, the above property comes at a cost.

Theorem 5.6. *The problem "given a finite propositional logic program $\mathcal{P}$, decide whether $\mathcal{P}$ has a grounded fixpoint" is Σ_2^P-complete.*

Let us briefly compare grounded fixpoint semantics with the two most frequently used semantics of logic programming: well-founded and stable semantics. Firstly, it deserves to be stressed that the three semantics provide different formalisations of a similar intuition: a certain minimality criterion for fixpoints (which we called groundedness). Consequently it is to be expected that they often coincide. We established that for programs with a two-valued well-founded model, the three semantics coincide. This sort of programs is common in applications for deductive databases (Datalog and extensions (Abiteboul and Vianu 1991)) and for representing inductive definitions (Denecker and Vennekens 2014). In contrast, well-founded semantics coincides only seldom with stable semantics in the context of answer set programming (ASP). We illustrated in Example 4.2 that in this case, also stable and grounded fixpoint semantics may disagree. This example is quite unwieldy, as are all such programs that we found. It led us to expect that for large classes of ASP programs, both semantics still coincide. For those programs, we have defined a an *elegant*, *intuitive* and *concise* reformulation of the existing semantics. It is a topic for future research to search for characteristics of ASP programs that guarantee that both semantics agree or disagree.

Grounded fixpoint semantics is, to the best of our knowledge, the first *purely two-valued and algebraical* semantics. The well-founded semantics explicitly uses three-valued interpretations in the well-founded model construction. Stable semantics uses three-valued logic implicitly: the Gelfond-Lifschitz reduct corresponds to an evaluation in a partial interpretation. The ultimate versions of these semantics are purely algebraical but still refer to three-valued interpretations (replacing Kleene valuation by supervaluation). Due to this, ultimate stable and well-founded models are relatively complex to understand.

Logic Programs with Abstract Constraint Atoms. The fact that grounded fixpoints semantics is two-valued and algebraical makes it not only easier to understand, but also to *extend* the semantics. To illustrate this, we consider logic programs with abstract constraint atoms as defined by Marek, Niemelä, and Truszczyński (2008). An *abstract constraint* is a collection $C \subseteq 2^\Sigma$. A *constraint atom* is an expression of the form $C(X)$, where $X \subseteq \Sigma$ and C is an abstract constraint. The goal of such an atom is to model constraints on subsets of X. The truth value of $C(X)$ in interpretation I is $\mathbf{t}$ if $I \cap X \in C$ and $\mathbf{f}$ otherwise. Abstract constraints are a generalisation of pseudo-Boolean constraints, cardinality constraints, and much more. A *deterministic* logic program is a set of rules of the form[1]

$$p \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge b_m,$$

where p is an atom and the a_i and b_i are constraint atoms. The intuition behind such a rule is that p is justified if the constraints a_i are satisfied while the b_i are not. This intuition is captured in an extended immediate consequence operator:

$$T_\mathcal{P}(I) = \{p \mid \exists r \in \mathcal{P} : \mathit{head}(r) = p \wedge \mathit{body}(r)^I = \mathbf{t}\}.$$

Grounded fixpoints of this operator still represent the same intuitions: an interpretation I is grounded if it contains no unfounded sets, or said differently, no atoms without external support. Thus, if it contains no set U of atoms such that $T_\mathcal{P}(I[U : \mathbf{f}]) \cap U = \emptyset$.

Example 5.7. Let Σ be the alphabet $\{a, b, c, d\}$ For every i, let $C_{\geq i}$ be the cardinality constraint $\{X \subseteq \Sigma \mid |X| \geq i\}$.. Consider the following logic program $\mathcal{P}$ over Σ:

$$\left\{ \begin{array}{ll} a. & b \leftarrow C_{\geq 1}(\Sigma). \\ c \leftarrow \neg C_{\geq 4}(\Sigma). & d \leftarrow C_{\geq 4}(\Sigma). \end{array} \right\}$$

Any interpretation in which d holds is *not* grounded since for every I, $C_{\geq 4}(\Sigma)^{I[d:\mathbf{f}]} = \mathbf{f}$ and thus $d \notin T_\mathcal{P}(I[d : \mathbf{f}])$. It can easily be verified that $\{a, b, c\}$ is the only grounded fixpoint of $\mathcal{P}$.

This example illustrates that even for complex, abstract extensions of logic programs, groundedness is an intuitive property: for any extension, a point is grounded if it contains no self-supporting atoms. Also, it often possible to derive common properties of all grounded fixpoints such as the fact that d cannot be contained in any of them. Lastly, groundedness easily extends to these rich formalisms (defining grounded fixpoints takes one line given the immediate consequence operator). This is in sharp contrast with more common semantics of logic programming (such as stable

[1]Here, we limit ourselves to deterministic programs. In general, Marek, Niemelä, and Truszczyński also described nondeterministic programs. We come back to this issue in Section 6.

and well-founded semantics) which are often hard(er) to extend to richer formalisms, as can be observed by the many different versions of those semantics that exist for logic programs with aggregates (Ferraris 2005; Son, Pontelli, and Elkabani 2006; Pelov, Denecker, and Bruynooghe 2007; Faber, Pfeifer, and Leone 2011; Gelfond and Zhang 2014).

6 Discussion

Unfounded Sets. Unfounded sets were first defined by Van Gelder, Ross, and Schlipf (1991) in their seminal paper introducing the well-founded semantics. Their definition slightly differs from Definition 5.1.

Definition 6.1 (GRS-Unfounded set). Let $\mathcal{P}$ be a logic program and $\mathcal{I}$ a three-valued interpretation. A set $U \subseteq \Sigma$ is a *GRS-unfounded set* of $\mathcal{P}$ with respect to $\mathcal{I}$, if for each rule r with $head(r) \in U$, $body(r)^{\mathcal{I}} = \mathbf{f}$ or $body(r)^{\mathcal{I}[U:\mathbf{f}]} = \mathbf{f}$.

The first difference between 2-unfounded sets and GRS-unfounded sets is that GRS-unfounded sets are defined for three-valued interpretations, while we restricted our attention to (total) interpretations. Our definition easily generalises to three-valued interpretations using Fitting's operator:

Definition 6.2 (3-Unfounded set). Let $\mathcal{P}$ be a logic program, $\Psi_{\mathcal{P}}$ Fitting's immediate consequence operator and $\mathcal{I}$ a three-valued interpretation. A set $U \subseteq \Sigma$ is a *3-unfounded set* of $\mathcal{P}$ with respect to $\mathcal{I}$ if $\Psi_{\mathcal{P}}(\mathcal{I}[U : \mathbf{f}])_2 \cap U = \emptyset$.

This definition formalises the same intuitions as Definition 5.1: U is a 3-unfounded set if making all atoms in U false results in a state where none of them can be derived. The following proposition relates the two notions of unfounded sets.

Proposition 6.3. *Let $\mathcal{P}$ be a logic program, $\mathcal{I}$ a three-valued interpretation and $U \subseteq \Sigma$. The following hold.*

- *If U is a 3-unfounded set, then U is a GRS-unfounded set.*
- *If $\mathcal{I}[U : \mathbf{f}]$ is more precise than $\mathcal{I}$, then U is a GRS-unfounded set if and only if U is a 3-unfounded set.*

We showed that for a certain class of interpretations, the two notions of unfounded sets coincide. Furthermore, Van Gelder et al. only use unfounded sets to define the well-founded model construction. It follows immediately from Lemma 3.4 in (Van Gelder, Ross, and Schlipf 1991) that every partial interpretation $\mathcal{I}$ in that construction with GRS-unfounded set U satisfies the condition in the second claim in Proposition 6.3. This means that 3-unfounded sets and GRS-unfounded sets are equivalent for all interpretations that are relevant in the original work! Essentially, we provided a new formalisation of unfounded sets that correctly formalises the underlying intuitions, and that coincides with the old definition on all interpretations used in the original work. Furthermore, our definition is simpler and translates easily to algebra.

Corollary 5.5, which states that grounded fixpoints are fixpoints of $T_{\mathcal{P}}$ that permit no non-trivial 2-unfounded sets, might sound familiar. Indeed, it has been shown that an interpretation is a *stable model* of a logic program if and only if it is a fixpoint of $T_{\mathcal{P}}$ and it permits no non-trivial GRS-unfounded sets (Lifschitz 2008). This again shows that many of the intuitions used in Answer Set Programming are also closely related to the notion of groundedness.

Groundedness and Nondeterminism. In Section 5, we restricted ourselves to logic programs with abstract constraint atoms in the *bodies* of rules. As argued by Marek, Niemelä, and Truszczyński (2008), allowing them as well in heads gives rise to a *nondeterministic* generalisation of the immediate consequence operator. A consistent nondeterministic operator maps every point $x \in L$ to a non-empty set $O(x) \subseteq L$. The definition of groundedness can straightforwardly be extended to this nondeterministic setting: a point $x \in L$ is grounded for nondeterministic operator O, if $x \leq v$ for all v such that $O(x \wedge v) \leq v$, where we define for a set $X \subseteq L$ that $X \leq v$ if $x \leq v$ for every $x \in X$. A thorough study of groundedness for nondeterministic operators is out of the scope of this paper.

Other Definitions of Groundedness. The terminology "grounded" is heavily overloaded in the literature. This is not a coincidence since this term often represents similar intuitions. For example Denecker, Marek, and Truszczyński (2002) argued that ultimate stable models satisfy "some groundedness condition" without defining this condition. We formally defined groundedness and showed in Proposition 4.1 that with this definition, their claim indeed holds.

In 1988, Konolige defined notions of *weak*, *moderate* and *strong* groundedness in order to formalise some of his intuitions regarding "good" models of autoepistemic theories. However, as he mentions himself, the closest he got to formalising these intuitions was strong groundedness, a syntactical criterion that depends on how a theory is rewritten to a normal form. We now claim[2] that our notion of groundedness formalises his intuitions, or at least, that it works for all examples he gave!

In Dung's argumentation frameworks, the grounded semantics is also defined. Since this is defined as the least fixpoint of the (monotone) characteristic operator, in this case this is the unique grounded fixpoint. However, Strass (2013) showed that this does not generalise to abstract dialectical frameworks, where the grounded extension corresponds to the (ultimate) Kripke-Kleene fixpoint.

7 Conclusion

In this paper, we defined a new algebraical concept, namely groundedness. We showed that grounded fixpoints behave well with respect to other fixpoints studied in approximation fixpoint theory: given an operator O and an approximator A of O, all A-stable fixpoints are grounded for O and all grounded fixpoints of O are approximated by the A-well-founded fixpoint. Moreover, grounded fixpoints free us from the need of choosing such an approximator: they are defined directly in terms of the original lattice operator.

[2]The journal version of this paper will formally describe the application of our theory to various fields, including a more detailed discussion about the different notions of groundedness, Konolige's intuitions and the relation to grounded fixpoints.

Grounded fixpoint semantics is the first purely two-valued and algebraical semantics for logic programming. Moreover, this semantics is compact, intuitive (directly based on the notion of unfounded sets) and easily extensible: as long as the (two-valued) immediate consequence operator is defined, the grounded fixpoint semantics is obtained for free.

Our theory can also be applied to AEL, DL, Dung's argumentation frameworks and ADFs where it also results in a semantics with attractive properties.[2]

Acknowledgements This work was supported by the KU Leuven under project GOA 13/010 and by the Research Foundation - Flanders (FWO-Vlaanderen).

References

Abiteboul, S., and Vianu, V. 1991. Datalog extensions for database queries and updates. *J. Comput. Syst. Sci.* 43(1):62–124.

Antic, C.; Eiter, T.; and Fink, M. 2013. Hex semantics via approximation fixpoint theory. In Cabalar, P., and Son, T. C., eds., *LPNMR*, volume 8148 of *LNCS*, 102–115. Springer.

Balduccini, M. 2013. ASP with non-herbrand partial functions: A language and system for practical use. *TPLP* 13(4-5):547–561.

Bogaerts, B.; Vennekens, J.; Denecker, M.; and Van den Bussche, J. 2014. FO(C): A knowledge representation language of causality. *TPLP* 14(4-5-Online-Supplement):60–69.

Bogaerts, B.; Vennekens, J.; and Denecker, M. 2014. Grounded fixpoints. Technical Report CW 677, Departement of Computer Science, Katholieke Universiteit Leuven.

Denecker, M., and Vennekens, J. 2007. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In Baral, C.; Brewka, G.; and Schlipf, J. S., eds., *LPNMR*, volume 4483 of *LNCS*, 84–96. Springer.

Denecker, M., and Vennekens, J. 2014. The well-founded semantics is the principle of inductive definition, revisited. In Baral, C.; De Giacomo, G.; and Eiter, T., eds., *KR*, 22–31. AAAI Press.

Denecker, M.; Bruynooghe, M.; and Vennekens, J. 2012. Approximation fixpoint theory and the semantics of logic and answers set programs. In Erdem, E.; Lee, J.; Lierler, Y.; and Pearce, D., eds., *Correct Reasoning*, volume 7265 of *LNCS*. Springer. 178–194.

Denecker, M.; Marek, V.; and Truszczyński, M. 2000. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In Minker, J., ed., *Logic-Based Artificial Intelligence*, volume 597 of *The Springer International Series in Engineering and Computer Science*. Springer US. 127–144.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2002. Ultimate approximations in nonmonotonic knowledge representation systems. In Fensel, D.; Giunchiglia, F.; McGuinness, D. L.; and Williams, M.-A., eds., *KR*, 177–190. Morgan Kaufmann.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2003. Uniform semantic treatment of default and autoepistemic logics. *Artif. Intell.* 143(1):79–122.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2004. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Information and Computation* 192(1):84–121.

Denecker, M.; Marek, V. W.; and Truszczyński, M. 2011. Reiter's default logic is a logic of autoepistemic reasoning and a good one, too. In Brewka, G.; Marek, V.; and Truszczyński, M., eds., *Nonmonotonic Reasoning – Essays Celebrating Its 30th Anniversary*. College Publications. 111–144.

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

Ferraris, P. 2005. Answer sets for propositional theories. In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, 119–131.

Fitting, M. 2002. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science* 278(1-2):25–51.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *ICLP/SLP*, 1070–1080. MIT Press.

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

Kleene, S. C. 1938. On notation for ordinal numbers. *The Journal of Symbolic Logic* 3(4):pp. 150–155.

Konolige, K. 1988. On the relation between default and autoepistemic logic. *Artif. Intell.* 35:343–382.

Lifschitz, V. 2008. Twelve definitions of a stable model. In García de la Banda, M., and Pontelli, E., eds., *ICLP*, volume 5366 of *LNCS*, 37–51. Springer.

Marek, V. W.; Niemelä, I.; and Truszczyński, M. 2008. Logic programs with monotone abstract constraint atoms. *TPLP* 8(2):167–199.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *TPLP* 7(3):301–353.

Son, T. C.; Pontelli, E.; and Elkabani, I. 2006. An unfolding-based semantics for logic programming with aggregates. *CoRR* abs/cs/0605038.

Strass, H. 2013. Approximating operators and semantics for abstract dialectical frameworks. *Artif. Intell.* 205:39–70.

van Emden, M. H., and Kowalski, R. A. 1976. The semantics of predicate logic as a programming language. *J. ACM* 23(4):733–742.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

Vennekens, J.; Gilis, D.; and Denecker, M. 2006. Splitting an operator: Algebraic modularity results for logics with fixpoint semantics. *ACM Trans. Comput. Log.* 7(4):765–797.