Non-deterministic Approximation Operators: Ultimate Operators, Semi-equilibrium Semantics, and Aggregates

Heyninck, Jesse; Bogaerts, Bart

Published in:
Theory and Practice of Logic Programming

DOI:
10.1017/S1471068423000236

Publication date:
2023

TPLP **23** (4): 632–647, 2023.

doi:10.1017/S1471068423000236 First published online 11 July 2023

Non-deterministic Approximation Operators: Ultimate Operators, Semi-equilibrium Semantics, and Aggregates[*]

JESSE HEYNINCK
Open Universiteit, Heerlen, the Netherlands
(*e-mail:* jesse.heyninck@ou.nl)

BART BOGAERTS
Vrij Universiteit Brussels, Brussels, Belgium
(*e-mail:* bart.bogaerts@ai.vub.ac.be)

submitted 19 May 2023; accepted 12 June 2023

Abstract

Approximation fixpoint theory (AFT) is an abstract and general algebraic framework for studying the semantics of non-monotonic logics. In recent work, AFT was generalized to non-deterministic operators, that is, operators whose range are sets of elements rather than single elements. In this paper, we make three further contributions to non-deterministic AFT: (1) we define and study ultimate approximations of non-deterministic operators, (2) we give an algebraic formulation of the semi-equilibrium semantics by Amendola *et al.*, and (3) we generalize the characterizations of disjunctive logic programs to disjunctive logic programs with aggregates.

KEYWORDS: approximation fixpoint theory, disjunctive logic programming, semi-equilibrium semantics

1 Introduction

Knowledge representation and reasoning (KRR), by its very nature, is concerned with the study of a wide variety of languages and formalisms. In view of this, *unifying* frameworks that allow for the language-independent study of aspects of KRR is essential. One framework with strong unifying potential is *approximation fixpoint theory* (AFT) (Denecker *et al.* 2000), a purely algebraic theory which was shown to unify the semantics of, among others, logic programming default logic and autoepistemic logic. The central objects of study of AFT are *(approximating) operators* and their *fixpoints*. For logic programming for instance, it was shown that Fitting's three-valued immediate consequence operator is an approximating operator of Van Emden and Kowalski's two-valued immediate consequence operator, and that all major semantics of (normal) logic programming

[*] This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G0B2221N) and the Flemish Government (Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen).

can be derived directly from this approximating operator. Moreover, this observation does not only hold for logic programming: also for a wide variety of other domains, it is straightforward how to derive an approximating operator, and the major semantics can be recovered from that approximator using purely algebraic means (an overview is given by Heyninck *et al.* 2022). This has in turn inspired others to define the semantics of non-monotonic formalisms *directly* using AFT (Bogaerts 2019), putting AFT forward not only as a framework to study existing semantics but also as a framework to define them. The advantage is that AFT-based semantics are guaranteed to follow well-established principles, such as *groundedness* (Bogaerts 2015). Moreover, it is often easier to define a semantic operator than to define the semantics from scratch.

Recently, AFT was generalized to also capture *non-deterministic operators* (Heyninck *et al.* 2022) which allow for different options or choices in their output. A prime example of the occurrence of non-determinism in KRR is *disjunctive logic programming*, and it was indeed shown that many semantics of disjunctive logic programming (specifically the weakly supported, (partial) stable, and well-founded semantics (Alcântara *et al.* 2005) are captured by non-deterministic AFT. In this paper, we make further contributions to the study of non-deterministic AFT, with a particular emphasis on disjunctive logic programs. On the one hand, (in Section 3) we deepen the theory of non-deterministics AFT by investigating so-called *ultimate semantics*. For standard AFT, Denecker *et al.* (2002) have shown that with every two-valued operator, we can uniquely associate a most-precise approximator called the *ultimate approximator*. When defining semantics of new formalisms, this even takes the need of defining an approximator away, since it suffices to define an *exact* operator and its ultimate approximator comes for free.[1] Our first contribution is to show how ultimate approximations can be obtained for non-deterministic AFT, which we later illustrate using disjunctive logic programs with aggregates. This means we give the first constructive method for obtaining non-deterministic approximation operators. On the other hand, we also *apply* non-deterministic AFT to two areas that have thus far been out of reach of AFT. In Section 4, we use it to define an algebraic generalization of the *semi-equilibrium semantics*, a semantics originally formulated for disjunctive logic programs (Amendola *et al.* 2016) but now, thanks to our results, available to any operator-based semantics. In Section 5, we apply the theory of non-deterministic AFT to disjunctive logic programs with *aggregates* in the body, giving rise to a family of semantics for such programs.

2 Background and preliminaries

In this section, we recall disjunctive logic programming (Section 2.1), AFT for deterministic operators (Section 2.2), and non-deterministic operators (Section 2.3).

2.1 Disjunctive logic programming

In what follows we consider a propositional[2] language $\mathfrak{L}$, whose atomic formulas are denoted by p, q, r (possibly indexed), and that contains the propositional constants T

[1] However, ultimate semantics often come at the cost of increased computational complexity compared to their standard counterparts.
[2] For simplicity, we restrict ourselves to the propositional case.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

(representing truth), F (falsity), U (unknown), and C (contradictory information). The connectives in $\mathfrak{L}$ include negation $\neg$, conjunction $\wedge$, disjunction $\vee$, and implication $\leftarrow$. Formulas are denoted by ϕ, ψ, δ (again, possibly indexed). Logic programs in $\mathfrak{L}$ may be divided to different kinds as follows: a (propositional) *disjunctive logic program* $\mathcal{P}$ in $\mathfrak{L}$ (a dlp in short) is a finite set of rules of the form $\bigvee_{i=1}^{n} p_i \leftarrow \psi$, where the head $\bigvee_{i=1}^{n} p_i$ is a non-empty disjunction of atoms, and the body ψ is a formula not containing $\leftarrow$. A rule is called *normal* (nlp), if its body is a conjunction of literals (i.e., atomic formulas or negated atoms), and its head is atomic. A rule is *disjunctively normal* if its body is a conjunction of literals and its head is a non-empty disjunction of atoms. We will use these denominations for programs if all rules in the program satisfy the denomination, for example, a program is normal if all its rules are normal. The set of atoms occurring in $\mathcal{P}$ is denoted $\mathcal{A}_{\mathcal{P}}$.

The semantics of dlps are given in terms of *four-valued interpretations*. A *four-valued interpretation* of a program $\mathcal{P}$ is a pair (x, y), where $x \subseteq \mathcal{A}_{\mathcal{P}}$ is the set of the atoms that are assigned a value in $\{\mathsf{T}, \mathsf{C}\}$ and $y \subseteq \mathcal{A}_{\mathcal{P}}$ is the set of atoms assigned a value in $\{\mathsf{T}, \mathsf{U}\}$. We define $-\mathsf{T} = \mathsf{F}$, $-\mathsf{F} = \mathsf{T}$ and $\mathsf{X} = -\mathsf{X}$ for $\mathsf{X} = \mathsf{C}, \mathsf{U}$. Truth assignments to complex formulas are as follows:

- $(x, y)(p) = \begin{cases} \mathsf{T} & \text{if } p \in x \text{ and } p \in y, \\ \mathsf{U} & \text{if } p \notin x \text{ and } p \in y, \\ \mathsf{F} & \text{if } p \notin x \text{ and } p \notin y, \\ \mathsf{C} & \text{if } p \in x \text{ and } p \notin y. \end{cases}$
- $(x, y)(\neg\phi) = -(x, y)(\phi)$,
- $(x, y)(\psi \wedge \phi) = lub_{\leq_t}\{(x, y)(\phi), (x, y)(\psi)\}$,
- $(x, y)(\psi \vee \phi) = glb_{\leq_t}\{(x, y)(\phi), (x, y)(\psi)\}$.

A four-valued interpretation of the form (x, x) may be associated with a *two-valued* (or *total*) interpretation x. (x, y) is a *three-valued* (or *consistent*) interpretation, if $x \subseteq y$. Interpretations are compared by two-order relations which form a pointwise extension of the structure $\mathcal{FOUR}$ consisting of $\mathsf{T}, \mathsf{F}, \mathsf{C}$, and U with $\mathsf{U} <_i \mathsf{F}, \mathsf{T} <_i \mathsf{C}$ and $\mathsf{F} <_t \mathsf{C}, \mathsf{U} <_t \mathsf{T}$. The pointwise extension of these orders corresponds to the *information order*, which is equivalently defined as $(x, y) \leq_i (w, z)$ iff $x \subseteq w$ and $z \subseteq y$, and the *truth order*, where $(x, y) \leq_t (w, z)$ iff $x \subseteq w$ and $y \subseteq z$.

The immediate consequence operator for normal programs (van Emden and Kowalski 1976) is extended to dlps as follows:

Definition 1 (*Immediate Consquence operator for dlps*)
Given a dlp $\mathcal{P}$ and a two-valued interpretation x, we define: (1) $HD_{\mathcal{P}}(x) = \{\Delta \mid \bigvee\Delta \leftarrow \psi \in \mathcal{P} \text{ and } (x, x)(\psi) = \mathsf{T}\}$; and (2) $IC_{\mathcal{P}}(x) = \{y \subseteq \bigcup HD_{\mathcal{P}}(x) \mid \forall\Delta \in HD_{\mathcal{P}}(x), y \cap \Delta \neq \emptyset\}$.

Thus, $IC_{\mathcal{P}}(x)$ consists of sets of atoms that occur in activated rule heads, each set contains at least one representative from every disjuncts of a rule in $\mathcal{P}$ whose body is x-satisfied. Denoting by $\wp(\mathcal{S})$, the powerset of $\mathcal{S}$, $IC_{\mathcal{P}}$ is an operator on the lattice $\langle \wp(\mathcal{A}_{\mathcal{P}}), \subseteq \rangle$.[3]

Given a dlp $\mathcal{P}$ a consistent interpretation (x, y) is a *(three-valued) model* of $\mathcal{P}$, if for every $\phi \leftarrow \psi \in \mathcal{P}$, $(x, y)(\phi) \geq_t (x, y)(\psi)$. The GL-transformation $\frac{\mathcal{P}}{(x,y)}$ of a disjunctively

[3] The operator $IC_{\mathcal{P}}$ is a generalization of the immediate consequence operator from (Fernández and Minker 1995, Definition 3.3), where the minimal sets of atoms in $IC_{\mathcal{P}}(x)$ are considered. However, this requirement of minimality is neither necessary nor desirable in the consequence operator (Heyninck *et al.* 2022).

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

normal dlp $\mathcal{P}$ with respect to a consistent (x, y) is the positive program obtained by replacing in every rule in $\mathcal{P}$ of the form $p_1 \vee \ldots \vee p_n \leftarrow \bigwedge_{i=1}^{m} q_i \wedge \bigwedge_{j=1}^{n} \neg r_j$ a negated literal $\neg r_i$ $(1 \leq i \leq k)$ by $(x, y)(\neg r_i)$. (x, y) is a *three-valued stable model* of $\mathcal{P}$ iff it is a $\leq_t$-minimal model of $\frac{\mathcal{P}}{(x,y)}$. [4]

2.2 Approximation fixpoint theory

We now recall basic notions from AFT, as described by Denecker *et al.* (2000). We restrict ourselves here to the necessary formal details and refer to more detailed introductions by Denecker *et al.* (2000) and Bogaerts (2015) for more informal details. AFT introduces constructive techniques for approximating the fixpoints of an operator O over a lattice $L = \langle \mathcal{L}, \leq \rangle$.[5] Approximations are pairs of elements (x, y). Thus, given a lattice $L = \langle \mathcal{L}, \leq \rangle$, the induced *bilattice* is the structure $L^2 = \langle \mathcal{L}^2, \leq_i, \leq_t \rangle$, in which $\mathcal{L}^2 = \mathcal{L} \times \mathcal{L}$, and for every $x_1, y_1, x_2, y_2 \in \mathcal{L}$, $(x_1, y_1) \leq_i (x_2, y_2)$ if $x_1 \leq x_2$ and $y_1 \geq y_2$, and $(x_1, y_1) \leq_t (x_2, y_2)$ if $x_1 \leq x_2$ and $y_1 \leq y_2$.[6]

An *approximating operator* $\mathcal{O} : \mathcal{L}^2 \to \mathcal{L}^2$ of an operator $O : \mathcal{L} \to \mathcal{L}$ is an operator that maps every approximation (x, y) of an element z to an approximation (x', y') of another element $O(z)$, thus approximating the behavior of the approximated operator O. In more details, an operator $\mathcal{O} : \mathcal{L}^2 \to \mathcal{L}^2$ is $\leq_i$*-monotonic*, if when $(x_1, y_1) \leq_i (x_2, y_2)$, also $\mathcal{O}(x_1, y_1) \leq_i \mathcal{O}(x_2, y_2)$; $\mathcal{O}$ is *approximating*, if it is $\leq_i$-monotonic and for any $x \in \mathcal{L}$, $\mathcal{O}_l(x, x) = \mathcal{O}_u(x, x)$.[7] $\mathcal{O}$ *approximates* of $O : \mathcal{L} \to \mathcal{L}$, if it is $\leq_i$-monotonic and $\mathcal{O}(x, x) = (O(x), O(x))$ (for every $x \in \mathcal{L}$). Finally, for a complete lattice L, let $\mathcal{O} : \mathcal{L}^2 \to \mathcal{L}^2$ be an approximating operator. We denote $\mathcal{O}_l(\cdot, y) = \lambda x.\mathcal{O}_l(x, y)$ and similarly for $\mathcal{O}_u$. The *stable operator for* $\mathcal{O}$ is then defined as $S(\mathcal{O})(x, y) = (lfp(\mathcal{O}_l(., y)), lfp(\mathcal{O}_u(x, .))$, where $lfp(O)$ denotes the least fixpoint of an operator O.

Approximating operators induce a family of *fixpoint semantics*. Given a complete lattice $L = \langle \mathcal{L}, \leq \rangle$ and an approximating operator $\mathcal{O} : \mathcal{L}^2 \to \mathcal{L}^2$, (x, y) is a *Kripke–Kleene fixpoint* of $\mathcal{O}$ if $(x, y) = lfp_{\leq_i}(\mathcal{O}(x, y))$; (x, y) is a *three-valued stable fixpoint* of $\mathcal{O}$ if $(x, y) = S(\mathcal{O})(x, y)$; (x, y) is a *two-valued stable fixpoints* of $\mathcal{O}$ if $x = y$ and $(x, x) = S(\mathcal{O})(x, x)$; (x, y) is the *well-founded fixpoint* of $\mathcal{O}$ if it is the $\leq_i$-minimal (three-valued) stable fixpoint of $\mathcal{O}$.

2.3 Non-deterministic AFT

AFT was generalized to non-deterministic operators, that is, operators which map elements of a lattice to a set of elements of that lattice (like the operator $IC_{\mathcal{P}}$ for dlps) by

[4] An overview of other semantics for dlps can be found in previous work on non-deterministic AFT (Heyninck *et al.* 2022).

[5] Recall that a lattice is a partially ordered set in which every pair of elements has a least upper bound and greatest lower bound denoted by $\sqcup$ and $\sqcap$, respectively. If every set of elements has a least upper bound and greatest lower bound, we call the lattice complete.

[6] Note that we use small letters to denote elements of lattice, capital letters to denote sets of elements, and capital calligraphic letters to denote sets of sets of elements.

[7] In some papers (e.g., Denecker *et al.* 2000), an approximation operator is defined as a symmetric $\leq_i$-monotonic operator, that is, a $\leq_i$-monotonic operator s.t. for every $x, y \in \mathcal{L}$, $\mathcal{O}(x, y) = (\mathcal{O}_l(x, y), \mathcal{O}_l(y, x))$ for some $\mathcal{O}_l : \mathcal{L}^2 \to \mathcal{L}$. However, the weaker condition we take here (taken from Denecker *et al.* 2002) is actually sufficient for most results on AFT.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

Heyninck *et al.* (2022). We recall the necessary details, referring to the original paper for more details and explanations.

A *non-deterministic operator on* $\mathcal{L}$ is a function $O : \mathcal{L} \to \wp(\mathcal{L}) \setminus \{\emptyset\}$. For example, the operator $IC_{\mathcal{P}}$ from Definition 1 is a non-deterministic operator on the lattice $\langle \wp(\mathcal{A}_{\mathcal{P}}), \subseteq \rangle$.

As the ranges of non-deterministic operators are *sets* of lattice elements, one needs a way to compare them, such as the *Smyth order* and the *Hoare order*. Let $L = \langle \mathcal{L}, \leq \rangle$ be a lattice, and let $X, Y \in \wp(\mathcal{L})$. Then, $X \preceq_L^S Y$ if for every $y \in Y$ there is an $x \in X$ such that $x \leq y$; and $X \preceq_L^H Y$ if for every $x \in X$ there is a $y \in Y$ such that $x \leq y$. Given some $X_1, X_2, Y_1, Y_2 \subseteq \mathcal{L}$, $X_1 \times Y_1 \preceq_i^A X_2 \times Y_2$ iff $X_1 \preceq_L^S X_2$ and $Y_2 \preceq_L^H Y_1$. Let $L = \langle \mathcal{L}, \leq \rangle$ be a lattice. Given an operator $\mathcal{O} : \mathcal{L}^2 \to \mathcal{L}^2$, we denote by $\mathcal{O}_l$ the operator defined by $\mathcal{O}_l(x, y) = \mathcal{O}(x, y)_1$, and similarly for $\mathcal{O}_u(x, y) = \mathcal{O}(x, y)_2$. An operator $\mathcal{O} : \mathcal{L}^2 \to \wp(\mathcal{L}) \setminus \emptyset \times \wp(\mathcal{L}) \setminus \emptyset$ is called a *non-deterministic approximating operator* (ndao, for short), if it is $\preceq_i^A$-monotonic (i.e., $(x_1, y_1) \leq_i (x_2, y_2)$ implies $\mathcal{O}(x_1, y_1) \preceq_i^A \mathcal{O}(x_2, y_2)$) and is *exact* (i.e., for every $x \in \mathcal{L}$, $\mathcal{O}(x, x) = \mathcal{O}_l(x, x) \times \mathcal{O}_l(x, x)$). We restrict ourselves to ndaos ranging over consistent pairs (x, y).

We finally define the stable operator (given an ndao $\mathcal{O}$) as follows. The *complete lower stable operator* is defined by (for any $y \in \mathcal{L}$) $C(\mathcal{O}_l)(y) = \{x \in \mathcal{L} \mid x \in \mathcal{O}_l(x, y) \text{ and } \neg\exists x' < x : x' \in \mathcal{O}_l(x', y)\}$. The *complete upper stable operator* is defined by (for any $x \in \mathcal{L}$) $C(\mathcal{O}_u)(x) = \{y \in \mathcal{L} \mid y \in \mathcal{O}_u(x, y) \text{ and } \neg\exists y' < y : y' \in \mathcal{O}_u(x, y')\}$. The *stable operator*: $S(\mathcal{O})(x, y) = C(\mathcal{O}_l)(y) \times C(\mathcal{O}_u)(x)$. (x, y) is a *stable fixpoint* of $\mathcal{O}$ if $(x, y) \in S(\mathcal{O})(x, y)$.[8]

Other semantics, for example, the well-founded state and the Kripke–Kleene fixpoints and state are defined by Heyninck *et al.* (2022) and can be immediately obtained once an ndao is formulated.

Example 1

An example of an ndao approximating $IC_{\mathcal{P}}$ (Definition 1) is defined as follows (given a dlp $\mathcal{P}$ and an interpretation (x, y)): $\mathcal{HD}^l_{\mathcal{P}}(x, y) = \{\Delta \mid \bigvee\Delta \leftarrow \phi \in \mathcal{P}, (x, y)(\phi) \geq_t \mathsf{C}\}$, $\mathcal{HD}^u_{\mathcal{P}}(x, y) = \{\Delta \mid \bigvee\Delta \leftarrow \phi \in \mathcal{P}, (x, y)(\phi) \geq_t \mathsf{U}\}$, $\mathcal{IC}^\dagger_{\mathcal{P}}(x, y) = \{x_1 \subseteq \bigcup \mathcal{HD}^\dagger_{\mathcal{P}}(x, y) \mid \forall\Delta \in \mathcal{HD}^\dagger_{\mathcal{P}}(x, y),\ x_1 \cap \Delta \neq \emptyset\}$ (for $\dagger \in \{l, u\}$), and $\mathcal{IC}_{\mathcal{P}}(x, y) = (\mathcal{IC}^l_{\mathcal{P}}(x, y), \mathcal{IC}^u_{\mathcal{P}}(x, y))$.

Consider the following dlp: $\mathcal{P} = \{p \vee q \leftarrow \neg q\}$. The operator $\mathcal{IC}^l_{\mathcal{P}}$ behaves as follows:

- For any interpretation (x, y) for which $q \in x$, $\mathcal{HD}^l_{\mathcal{P}}(x, y) = \emptyset$ and thus $\mathcal{IC}^l_{\mathcal{P}}(x, y) = \{\emptyset\}$.
- For any interpretation (x, y) for which $q \notin x$, $\mathcal{HD}^l_{\mathcal{P}}(x, y) = \{\{p, q\}\}$ and thus $\mathcal{IC}^l_{\mathcal{P}}(x, y) = \{\{p\}, \{q\}, \{p, q\}\}$.

Since $\mathcal{IC}^l_{\mathcal{P}}(x, y) = \mathcal{IC}^u_{\mathcal{P}}(y, x)$ (see Heyninck *et al.* 2022, Lemma 1), $\mathcal{IC}_{\mathcal{P}}$ behaves as follows:

- For any (x, y) with $q \notin x$ and $q \notin y$, $\mathcal{IC}_{\mathcal{P}}(x, y) = \{\{p\}, \{q\}, \{p, q\}\} \times \{\{p\}, \{q\}, \{p, q\}\}$,
- For any (x, y) with $q \notin x$ and $q \in y$, $\mathcal{IC}_{\mathcal{P}}(x, y) = \{\emptyset\} \times \{\{\{p\}, \{q\}, \{p, q\}\}$,
- For any (x, y) with $q \in x$ and $q \notin y$, $\mathcal{IC}_{\mathcal{P}}(x, y) = \{\{p\}, \{q\}, \{p, q\}\} \times \{\emptyset\}$, and
- For any (x, y) with $q \in x$ and $q \in y$, $\mathcal{IC}_{\mathcal{P}}(x, y) = \{(\emptyset, \emptyset)\}$.

[8] Notice that we slightly abuse notation and write $(x, y) \in S(\mathcal{O})(x, y)$ to abbreviate $x \in (S(\mathcal{O})(x, y))_1$ and $y \in (S(\mathcal{O})(x, y))_2$, that is, x is a lower bound generated by $S(\mathcal{O})(x, y)$ and y is an upper bound generated by $S(\mathcal{O})(x, y)$.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

We see, for example, that $C(\mathcal{IC}^l_{\mathcal{P}})(\{p\}) = \{\{p\}, \{q\}\}$ and thus $(\{p\}, \{p\})$ is a stable fixpoint of $\mathcal{IC}_{\mathcal{P}}$. $(\emptyset, \{q\})$ is the second stable fixpoint of $\mathcal{IC}_{\mathcal{P}}$. $(\emptyset, \{p, q\})$ is a fixpoint of $\mathcal{IC}_{\mathcal{P}}$ that is not stable.

In general, (total) stable fixpoints of $\mathcal{IC}_{\mathcal{P}}$ correspond to (total) stable models of $\mathcal{P}$, and weakly supported models of $\mathcal{IC}_{\mathcal{P}}$ correspond to fixpoints of $\mathcal{IC}_{\mathcal{P}}$. (Heyninck *et al.* 2022).

3 Ultimate operators

AFT assumes an approximation operator but does not specify how to construct it. In the literature, one finds various ways to construct a deterministic approximation operator $\mathcal{O}$ that approximates a deterministic operator O. Of particular interest is the *ultimate* operator (Denecker *et al.* 2002), which is the *most precise* approximation operator. In this section, we show that non-deterministic AFT admits an ultimate operator, which is, however, different from the ultimate operator for deterministic AFT.

We first recall that for a *deterministic* operator $O : \mathcal{L} \to \mathcal{L}$, the ultimate approximation $\mathcal{O}^u$ is defined by Denecker *et al.* (2002) as follows:[9]

$$\mathcal{O}^{\mathsf{DMT}^{\mathsf{d}}}(x, y) = (\sqcap O[x, y], \sqcup O[x, y])^{10}$$

Where $O[x, y] := \{O(z) \mid x \leq z \leq y\}$. This operator is shown to be the most precise operator approximating an operator O (Denecker *et al.* 2002). In more detail, for any (deterministic) approximation operator $\mathcal{O}$ approximating O, and any consistent (x, y), $\mathcal{O}(x, y) <_i \mathcal{O}^{\mathsf{DMT}^{\mathsf{d}}}(x, y)$.

The ultimate approximator for $IC_{\mathcal{P}}$ for non-disjunctive logic programs $\mathcal{P}$ looks as follows:

Definition 2
Given a normal logic program $\mathcal{P}$, we let: $\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}}}_{\mathcal{P}}(x, y) = (\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}},l}_{\mathcal{P}}(x, y), \mathcal{IC}^{\mathsf{DM}^{\mathsf{d}},u}_{\mathcal{P}}(x, y))$ with: $\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}},l}_{\mathcal{P}}(x, y) = \bigcap_{x \subseteq z \subseteq y} \{\alpha \mid \alpha \leftarrow \phi \in \mathcal{P} \text{ and } z(\phi) = \mathsf{T}\}$, and $\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}},u}_{\mathcal{P}}(x, y) = \bigcup_{x \subseteq z \subseteq y} \{\alpha \mid \alpha \leftarrow \phi \in \mathcal{P} \text{ and } z(\phi) = \mathsf{T}\}$.

In this section, we define the ultimate semantics for the non-deterministic operators. In more detail, we constructively define an approximation operator that is most precise and has non-empty upper and lower bounds. Its construction is based on the following idea: we are looking for an operator $\mathcal{O}^{\mathcal{U}}$ s.t. for any ndao $\mathcal{O}$ that approximates O, $\mathcal{O}_l(x, y) \preceq^S_L \mathcal{O}^{\mathcal{U}}_l(x, y)$ (and similarly for the upper bound). As we know that $\mathcal{O}_l(x, y) \preceq^S_L O(z)$ for any $x \leq z \leq y$, we can obtain $\mathcal{O}^{\mathcal{U}}_l$ by simply gathering all applications of O to elements of the interval $[x, y]$, that is, we define

$$\mathcal{O}^{\mathcal{U}}_l(x, y) = \bigcup_{x \leq z \leq y} O(z)$$

[9] We use the abbreviation $\mathsf{DMT}^{\mathsf{d}}$ for *deterministic* Denecker *et al.* to denote this operator, as to not overburden the use of $\mathcal{IC}^{\mathcal{U}}_{\mathcal{P}}$. Indeed, we will later see that the ultimate operator for non-disjunctive logic programs generalizes to an ndao that is different from the ultimate non-deterministic operator $\mathcal{IC}^{\mathcal{U}}_{\mathcal{P}}$.

[10] Recall that denotes $\sqcap X$ the greatest lower bound of X and $\sqcup X$ denotes the least upper bound of X.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

The upper bound can be defined in the same way as the lower bound. Altogether, we obtain

$$\mathcal{O}^{\mathcal{U}}(x,y) = \mathcal{O}_l^{\mathcal{U}}(x,y) \times \mathcal{O}_l^{\mathcal{U}}(x,y)$$

The following example illustrates this definition for normal logic programs:

Example 2
Let $\mathcal{P} = \{q \leftarrow \neg p; p \leftarrow p\}$. Then $IC_{\mathcal{P}}(\emptyset) = IC_{\mathcal{P}}(\{q\}) = \{q\}$ and $IC_{\mathcal{P}}(\{p\}) = IC_{\mathcal{P}}(\{p,q\}) = \{p\}$. Therefore, $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}(\emptyset, \{p,q\}) = \{\{p\},\{q\}\} \times \{\{p\},\{q\}\}$ whereas $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}(\emptyset,\{q\}) = \{\{q\}\} \times \{\{q\}\}$.

The ultimate approximation is the most precise ndao approximating the operator O:

Proposition 1
Let a non-deterministic operator O over a lattice $\langle \mathcal{L}, \leq \rangle$ be given. Then $\mathcal{O}^{\mathcal{U}}$ is an ndao that approximates O. Furthermore, for any ndao $\mathcal{O}$ that approximates O and for every $x, y \in \mathcal{L}$ s.t. $x \leq y$, it holds that $\mathcal{O}(x,y) \preceq_i^A \mathcal{O}^{\mathcal{U}}(x,y)$.

In conclusion, non-deterministic AFT admits, just like deterministic AFT, an ultimate approximation. However, as we will see in the rest of this section, the ultimate non-deterministic approximation operator $\mathcal{O}^{\mathcal{U}}$ does *not* generalize the deterministic ultimate approximation operator defined by Denecker *et al.* (2002). In more detail, we compare the non-deterministic ultimate operator $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}$ with the deterministic ultimate $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}$ from Definition 2. Somewhat surprisingly, even when looking at normal logic programs, the operator $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}}}$ does not coincide with the ultimate ndao $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}$ (and thus $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}}}$ is *not* the most precise ndao, even for non-disjunctive programs). The intuitive reason is that the additional expressivity of non-deterministic operators, which are not restricted to single lower and upper bounds in their outputs, allows to more precisely capture what is derivable in the "input interval" (x,y).

Example 3 (*Example 2 continued*)
Consider again $\mathcal{P} = \{q \leftarrow \neg p; p \leftarrow p\}$. Applying the $\mathsf{DMT}^{\mathsf{d}}$-operator gives: $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}}}(\emptyset,\{p,q\}) = (\emptyset,\{p,q\})$. Intuitively, the ultimate semantics $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}(\emptyset,\{p,q\}) = \{\{p\},\{q\}\} \times \{\{p\},\{q\}\}$ gives us the extra information that we will always either derive p or q, which is information a deterministic approximator can simply not capture. Such a "choice" is not expressible within a single interval; hence, the deterministic ultimate approximation is $(\emptyset,\{p,q\})$. This example also illustrates the fact that, when applying the ultimate ndao-construction to (non-constant) deterministic operators O, $\mathcal{O}^{\mathcal{U}}$ might be a *non*-deterministic approximation operator.

However, one can still generalize the operator $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}}}$ to disjunctive logic programs. We first generalize the idea behind $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}},l}$ to an operator gathering the heads of rules that are true in every interpretation z in the interval $[x,y]$. Similarly, $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}},u}$ is generalized by gathering the heads of rules with bodies that are true in at least one interpretation in $[x,y]$:

$$\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y) = \bigcap_{x \subseteq z \subseteq y} HD_{\mathcal{P}}(z) \quad \text{and} \quad \mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y) = \bigcup_{x \subseteq z \subseteq y} HD_{\mathcal{P}}(z)\}.$$

The upper and lower immediate consequences operator are then straightforwardly defined, that is, by taking all interpretations that only contain atoms in $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y)$ and contain at least one member of every head $\Delta \in \mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y)$ (for $\dagger \in \{u,l\}$):

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

$$\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y) = \{z \subseteq \bigcup \mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y) \mid \forall \Delta \in \mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y) \neq \emptyset : z \cap \Delta \neq \emptyset\}.$$

Finally, the DMT-ndao is defined as: $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}(x,y) = \mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y) \times \mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y)$. We have

Proposition 2 (*Heyninck et al. 2022, Proposition 3*)
For any disjunctive logic program $\mathcal{P}$, $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}$ is an ndao that approximates $IC_{\mathcal{P}}$.

Notice that for a non-disjunctive program $\mathcal{P}$, $\bigcup \mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y) = \bigcup \mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},\dagger}(x,y) = \mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}},\dagger}(x,y)$ (for $\dagger \in \{u,l\}$), that is, the non-deterministic version reduces to the deterministic version when looking at non-disjunctive programs. Notice furthermore the operators $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y)$ and $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y)$ are only defined for consistent interpretations (x,y). We leave the extension of this operator to inconsistent interpretations for future work.

Example 4
Consider again the program $\mathcal{P} = \{p \vee q \leftarrow \neg q\}$ from Example 1. $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},l}$ behaves as follows:

- If $q \in y$ then $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y) = \emptyset$ and thus $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y) = \emptyset$.
- If $q \notin y$ then $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y) = \{\{p,q\}\}$ and $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},l}(x,y) = \{\{p\},\{q\},\{p,q\}\}$.

$\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},u}$ behaves as follows:

- If $q \in x$ then $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y) = \emptyset$ and thus $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y) = \emptyset$.
- If $q \notin x$ then $\mathcal{HD}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y) = \{\{p,q\}\}$ and thus $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},u}(x,y) = \{\{p\},\{q\},\{p,q\}\}$.

Thus, for example, $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}(\emptyset,\{p,q\}) = \{\emptyset\} \times \{\{p\},\{q\},\{p,q\}\}$ and $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}(\{p\},\{p\}) = \{\{p\},\{q\},\{p,q\}\} \times \{\{p\},\{q\},\{p,q\}\}$. We thus see that $(\{p\},\{p\})$ is a stable fixpoint of $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}$.

A slightly extended program $\mathcal{P} = \{q \leftarrow \neg q; p \vee q \leftarrow q\}$ shows some particular but unavoidable behavior of this operator. $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT},l}(\emptyset,\{q\}) = \{\emptyset\}$ as $HD_{\mathcal{P}}(\emptyset) = \{\{q\}\}$ and $HD_{\mathcal{P}}(\{q\}) = \{\{p,q\}\}$. Note that the lower bound for is *not* the stronger $\{p\}$. This would result in a loss of $\preceq_i^A$-monotonicity, as the lower bound $\{\{q\}\}$ for the less informative $(\emptyset,\{q\})$ would be $\preceq_L^S$-incomparable to the lower bound $\{\{p\},\{q\},\{p,q\}\}$ of the more informative $(\{q\},\{q\})$.

We have shown in this section that non-deterministic AFT admits an ultimate operator, thus providing a way to construct an ndao based on a non-deterministic operator. We have also shown that the ultimate ndao diverges from the ultimate operator for deterministic AFT, but that this deterministic ultimate operator can be generalized to disjunctive logic programs. Both operators will be used in Section 5 to define semantics for dlps with aggregates.

4 Semi-equilibrium semantics

To further extend the reach of non-deterministic AFT, we generalize yet another semantics for dlps, namely the *semi-equilibrium semantics* (Amendola *et al.* 2016). The semi-equilibrium semantics is a semantics for disjunctive logic programs that has been

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

studied for disjunctively normal logic programs. This semantics is a three-valued semantics that fulfills the following properties deemed desirable by Amendola *et al.* (2016): (1) every (total) answer set of $\mathcal{P}$ corresponds to a semi-equilibrium model; (2) if $\mathcal{P}$ has a (total) answer set, then all of its semi-equilibrium models are (total) answer sets; (3) if $\mathcal{P}$ has a classical model, then $\mathcal{P}$ has a semi-equilibrium model. We notice that these conditions can be seen as a view on approximation of the total stable interpretations alternative to the well-founded semantics. We do not aim to have the last word on which semantics is the most intuitive or desirable. Instead, we will show here that semi-equilibrium models can be represented algebraically and thus can be captured within AFT. This leaves the choice of exact semantics to the user once an ndao has been defined, and allows the use of the semi-equilibrium semantics for formalisms other than nlps, such as disjunctive logic programs with aggregates (see below) or conditional ADFs.

Semi-equilibrium models are based on the *logic of here-and-there* (Pearce 2006). An *HT-interpretation* is a pair (x, y) where $x \subseteq y$ (i.e., a consistent pair in AFT-terminology). Satisfaction of a formula ϕ, denoted $\models_{\mathsf{HT}}$, is defined recursively as follows:

- $(x, y) \models_{\mathsf{HT}} \alpha$ if $\alpha \in x$ for any $\alpha \in \mathcal{A}_{\mathcal{P}}$,
- $(x, y) \models_{\mathsf{HT}} \neg\phi$ if $(y, y)(\phi) \neq \mathsf{T}$, and $(x, y) \not\models_{\mathsf{HT}} \bot$,
- $(x, y) \models_{\mathsf{HT}} \phi \wedge [\vee]\psi$ if $(x, y) \models_{\mathsf{HT}} \phi$ and [or] $(x, y) \models_{\mathsf{HT}} \psi$,
- $(x, y) \models_{\mathsf{HT}} \phi \rightarrow \psi$ if (a) $(x, y) \not\models_{\mathsf{HT}} \phi$ or $(x, y) \models_{\mathsf{HT}} \psi$, and (b) $(y, y)(\neg\phi \vee \psi) = \mathsf{T}$.

The HT-models of $\mathcal{P}$ are defined as $\mathsf{HT}(\mathcal{P}) = \{(x, y) \mid \forall \psi \leftarrow \phi \in \mathcal{P} : (x, y) \models_{\mathsf{HT}} \phi \rightarrow \psi\}$.

Semi-equilibrium models are a special class of HT-models. They are obtained by performing two minimization steps on the set of HT-models of a program. The first step is obtained by minimizing w.r.t. $\leq_t$.[11] The second step is obtained by selecting the *maximal canonical models*. For this, the *gap* of an interpretation is defined as $gap(x, y) = y \setminus x$,[12] and, for any set of interpretations $\mathbf{X}$, the *maximally canonical interpretations* are $mc(\mathbf{X}) = \{(x, y) \in \mathbf{X} \mid \nexists (w, z) \in \mathbf{X} : gap(x, y) \supset gap(w, z)\}$. The semi-equilibrium models of $\mathcal{P}$ are then defined as: $\mathcal{SEQ}(\mathcal{P}) = mc(\min_{\leq_t}(\mathsf{HT}(\mathcal{P}))$.

Example 5

We illustrate these semantics with the program $\mathcal{P} = \{p \leftarrow \neg p, s \vee q \leftarrow \neg s, s \vee q \leftarrow \neg q\}$. Then $\mathsf{HT}(\mathcal{P}) = \{(x, y) \mid \{p\} \subseteq y \subseteq \{p, q, s\}, x \subseteq y, \{q, s\} \cap y \neq \emptyset\}$. Furthermore, $\min_{\leq_t}(\mathsf{HT}(\mathcal{P})) = \{(\emptyset, \{p, q, s\}), (\{q\}, \{q, p\}), (\{s\}, \{s, p\})\}$. As $gap(\emptyset, \{p, q, s\}) = \{p, q, s\}$ and $gap(\{q\}, \{q, p\}) = gap(\{s\}, \{s, p\}) = \{p\}$, $\mathcal{SEQ} = \{(\{q\}, \{q, p\}), (\{s\}, \{s, p\})\}$.

Before we capture the ideas behind this semantics algebraically, we look a bit deeper into the relationship between $\mathsf{HT}(\mathcal{P})$-models and the classical notion of three-valued models of a program (see Section 2.1). We first observe that HT-models of a program are a proper superset of the three-valued models of a program:

Proposition 3

Let a disjunctively normal logic program $\mathcal{P}$ and a consistent intepretation (x, y) be given. Then if (x, y) is a model of $\mathcal{P}$, it is an HT-model of $\mathcal{P}$. However, not every HT-model is a model of $\mathcal{P}$.

[11] Amendola *et al.* (2016) proceeds as follows. First, $\mathsf{HT}^{\kappa}(\mathcal{P}) = \{x \cup \{\mathsf{K}\alpha \mid \alpha \in y\}\}$ is constructed, and then the $\subseteq$-minimal sets in $\mathsf{HT}^{\kappa}(\mathcal{P})$ are selected. It is straightforward to see that this is equivalent to minimizing the original interpretations w.r.t. $\leq_t$.

[12] Again, Amendola *et al.* (2016) proceeds in a slightly more convoluted way by defining $gap(I) = \{\mathsf{K}\alpha \in I \mid \alpha \notin I\}$ for any $I \in \mathsf{HT}^{\kappa}(\mathcal{P})$.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

We now define the concept of a HT-pair algebraically, inspired by Truszczyński (2006):

Definition 3
Given an ndao $\mathcal{O}$ approximating a non-determinstic operator O, a pair (x, y) is a HT-*pair* (denoted $(x, y) \in \mathsf{HT}(\mathcal{O})$) if the following three conditions are satisfied: (1) $x \leq y$, (2) $O(y) \preceq_L^S y$, and (3) $\mathcal{O}_l(x, y) \preceq_L^S x$.

This simple definition faithfully transposes the ideas behind HT-models to an algebraic context. Indeed, applying it to $\mathcal{IC}_\mathcal{P}$ gives use exactly the HT-models of $\mathcal{P}$:

Proposition 4
Let some normal disjunctive logic program $\mathcal{P}$ be given. Then, $\mathsf{HT}(\mathcal{P}) = \mathsf{HT}(\mathcal{IC}_\mathcal{P})$.

We now show that exact $\leq_t$-minimal HT-models of $\mathcal{O}$ are stable interpretations of $\mathcal{O}$ in our algebraic setting. The opposite direction holds as well: total stable fixpoints are $\leq_t$-minimal HT-pairs of $\mathcal{O}$. In fact, *every* total fixpoint of $\mathcal{O}$ is a HT-pair of $\mathcal{O}$. We assume that $\mathcal{O}$ is *upward coherent*, that is, for every $x, y \in \mathcal{L}$, $\mathcal{O}_l(x, y) \preceq_L^S \mathcal{O}_u(x, y)$. In the appendix of the full version of this article (Heyninck and Bogaerts 2023), we provide more details on upward coherent operators. Notice that all ndaos in this paper are upward coherent.

Proposition 5
Given an upward coherent ndao $\mathcal{O}$, (1) if $(x, x) \in \mathcal{O}(x, x)$ then $(x, x) \in \mathsf{HT}(\mathcal{O})$; and (2) $(x, x) \in \min_{\leq_t}(\mathsf{HT}(\mathcal{O}))$ iff $(x, x) \in S(\mathcal{O})(x, x)$.

The second concept that we have to generalize to an algebraic setting is that of maximal canonical models. Recall that $gap(x, y)$ consists of the atoms which are neither true nor false, that is, it can be used as a measure of the informativeness or precision of a pair. For the algebraic generalization of this idea, it is useful to assume that the lattice under consideration admits a difference for every pair of elements.[13] In more detail, $z \in \mathcal{L}$ is the *difference* of y w.r.t. x if $z \sqcap x = \bot$ and $x \sqcup y = x \sqcup z$. If the difference is unique, we denote it by $x \oslash y$. As an example, note that any Boolean lattice admits a unique difference for every pair of elements. We can then define $\mathsf{mc}(\mathbf{X}) = \arg\min_{(x,y)\in\mathbf{X}}\{y \oslash x\}$. This allows us to algebraically formulate the semi-equilibrium models of an ndao $\mathcal{O}$ as:

$$\mathcal{SEQ}(\mathcal{O}) = \mathsf{mc}\left(\min_{\leq_t}(\mathsf{HT}(\mathcal{O}))\right)$$

The properties mentioned at the start of this section are preserved, and this definition generalizes the semi-equilibrium models for disjunctive logic programs by Amendola *et al.* (2016):

Proposition 6
Let an upward coherent ndao $\mathcal{O}$ over a finite lattice be given s.t. every pair of elements admits a unique difference. Then, $\mathcal{SEQ}(\mathcal{O}) \neq \emptyset$. Furthermore, if there is some $(x, x) \in \mathsf{mc}(\min_{\leq_t}(\mathsf{HT}(\mathcal{O})))$ then $\mathcal{SEQ}(\mathcal{O}) = \{(x, x) \in \mathcal{L}^2 \mid (x, x) \in S(\mathcal{O})(x, x)\}$.

[13] If a lattice does not admit a difference for some elements, one cannot characterize the semi-equilibrium semantics exactly but can still obtain an approximate characterization. We detail this in the appendix of the full version of this article (Heyninck and Bogaerts 2023).

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

Corollary 1
Let a disjunctively normal logic program $\mathcal{P}$ be given. Then, $\mathcal{SEQ}(\mathcal{IC}_{\mathcal{P}}) = \mathcal{SEQ}(\mathcal{P})$.

In this section, we have shown that semi-equilibrium models can be characterized algebraically. This means semi-equilibrium models can now be obtained for other ndaos (e.g., those from Section 5, as illustrated in the appendix of the full version of this paper (Heyninck and Bogaerts 2023)), thus greatly enlarging the reach of these semantics.

We end this section by making a short, informal comparison between the semi-equilibrium models and the well-founded state for ndaos (Heyninck *et al.* 2022). Both constructions have a similar goal: namely, approximate the (potentially non-existent) total stable interpretations. In the case of the semi-equilibrium models, the set of semi-equilibrium models coincides with the total stable interpretations if they exist, whereas the well-founded state approximates any stable interpretation (and thus in particular the total stable interpretations) but might not coincide with them. When it comes to existence, we have shown here that the semi-equilibrium models exist for any ndao, just like the well-founded state. Thus, the well-founded state and semi-equilibrium models seem to formalize two different notions of approximation. Which notion is most suitable is hard to decide *in abstracto* but will depend on the exact application context.

5 Application to dlps with aggregates

We apply non-deterministic AFT to disjunctive logic programs with aggregates by studying three ndaos: the ultimate, DMT, and the trivial operators. We show the latter two generalize the ultimate semantics (Pelov *et al.* 2007), respectively, the semantics by Gelfond and Zhang (2019).

5.1 Preliminaries on aggregates

We survey the necessary preliminaries on aggregates and the corresponding programs, restricting ourselves to propositional aggregates and leaving aggregates with variables for future work.

A set term S is a set of pairs of the form $[\bar{t} : Conj]$ with t a list of constants and $Conj$ a ground conjunction of standard atoms For example, $[1 : p; 2 : q; -1 : r]$ intuitively assigns 1 to p, 2 to q and -1 to r. An *aggregate function* is of the form $f(S)$ where S is a set term, and f is an *aggregate function symbol* (e.g., #`Sum`, #`Count` or #`Max`). An *aggregate atom* is an expression of the form $f(S) * w$ where $f(S)$ is an aggregate function, $* \in \{<, \leq, \geq, >, =\}$ and w is a numerical constant. We denote by $\mathsf{At}(f(S) * w)$ the atoms occuring in S.

A *disjunctively normal aggregate program* consists of rules of the form (where Δ is a set of propositional atoms, and $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m$ are aggregate or propositional atoms):

$$\bigvee \Delta \leftarrow \alpha_1, \ldots, \alpha_n, \neg\beta_1, \ldots, \neg\beta_m$$

An aggregate symbol is evaluated w.r.t. a set of atoms as follows. First, let $x(S)$ denote the multiset $[t_1 \mid \langle t_1, \ldots, t_n : Conj\rangle \in S$ and $Conj$ is true w.r.t. $x]$. $x(f(S))$ is then simply the result of the application of f on $x(S)$. If the multiset $x(S)$ is not in the domain of f, $x(f(s)) = \lightning$ where $\lightning$ is a fixed symbol not occuring in $\mathcal{P}$. An aggregate

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

atom $f(S) * w$ is true w.r.t. x (in symbols, $x(f * w) = \mathsf{T}$) if: (1) $x(f(S)) \neq \lightning$ and (2) $x(f(S)) * w$ holds; otherwise, $f(S) * w$ is false (in symbols, $x(f * w) = \mathsf{F}$). $\neg f(S) * w$ is true if: (1) $x(f(S)) \neq \lightning$ and (2) $x(f(S)) * w$ does not hold; otherwise, $\neg f(S) * w$ is false. Evaluating a conjunction of aggregate atoms is done as usual. We can now straightforwardly generalize the immediate consequence operator for disjunctive logic programs to disjunctive aggregate programs by generalizing $HD_{\mathcal{P}}$ to take into account aggregate formulas as described above: $HD_{\mathcal{P}}(x) = \{\Delta \mid \bigvee \Delta \leftarrow \phi \in \mathcal{P}, x(\phi) = \mathsf{T}\}$. $IC_{\mathcal{P}}$ from Definition 1 is then generalized straightforwardly by simply using the generalized $HD_{\mathcal{P}}$. Thus, the only difference with the immediate consequence operator for dlps is that the set of activated heads $HD_{\mathcal{P}}$ now takes into account the truth of aggregates as well.

The first semantics we consider is the one formulated by Gelfond and Zhang (2019) (defined there only for logic programs with aggregates occurring positively in the body of a rule):

Definition 4
Let a disjunctively normal aggregate logic program $\mathcal{P}$ s.t. for every $\bigvee \Delta \leftarrow \bigwedge_{i=1}^{n} \alpha_i \wedge \bigwedge_{j=1}^{m} \neg\beta_j \in \mathcal{P}$, β_j is a normal (i.e., non-aggregate) atom. Then the GZ-reduct of $\mathcal{P}$ w.r.t. x is defined by doing, for every $r = \bigvee \Delta \leftarrow \bigwedge_{i=1}^{n} \alpha_i \wedge \bigwedge_{j=1}^{m} \neg\beta_j \in \mathcal{P}$, the following: (1) if an aggregate atom α_i is false or undefined for some $i = 1, \ldots, n$, delete r; (2) otherwise, replace every aggregate atom $\alpha_i = f(S) * w$ by $\bigcup\{Conj \text{ occurs in S} \mid x(Conj) = \mathsf{T}\}$. We denote the GZ-reduct of $\mathcal{P}$ by $\mathcal{P}^x_{\mathsf{GZ}}$. Notice that this is a disjunctively normal logic program. A set of atoms $x \subseteq \mathcal{A}_{\mathcal{P}}$ is a GZ-*answer set of* $\mathcal{P}$ if (x, x) is an answer set of $\mathcal{P}^x_{\mathsf{GZ}}$.

Example 6
Consider the program $\mathcal{P} = \{p \leftarrow \#\mathtt{Sum}[1 : p, q] > 0; p \leftarrow \#\mathtt{Sum}[1 : q] > 0; q \leftarrow \#\mathtt{Sum}[1 : s] < 1\}$. We check whether $\{p, q\}$ is a GZ-answer set as follows:

1. The GZ-reduct is $\mathcal{P}^{\{p,q\}}_{\mathsf{GZ}} = \{p \leftarrow p, q; \quad p \leftarrow q; \quad q \leftarrow\}$. In more detail, as $\{p, q\}(\#\mathtt{Sum}[1 : p, q] > 0) = \mathsf{T}$, we replace $\#\mathtt{Sum}[1 : p, q] > 0$ in the first rule by the atoms in the condition of this aggregate atom verified by $\{p, q\}$, namely p and q. Similarly for the other rules.
2. As $\{p, q\}$ (or, to be formally more precise, $(\{p, q\}, \{p, q\})$) is a minimal model of $\frac{\mathcal{P}^{\{p,q\}}_{\mathsf{GZ}}}{(\{p,q\},\{p,q\})}$, we see $\{p, q\}$ is a GZ-answer set of $\mathcal{P}$.

We now move to the semantics by Denecker *et al.* (2002). They are defined only for non-disjunctive aggregate programs. They are defined on the basis of the ultimate (deterministic) approximator $\mathcal{IC}^{\mathsf{DMT}}_{\mathcal{P}}$ (Definition 2). In more detail, an interpretation (x, y) is $\mathsf{DMT}^{\mathsf{d}}$-*stable* if and only if $(x, y) \in S(\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}}}_{\mathcal{P}})(x, y)$, that is, $x \in lfp(\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}}}_{\mathcal{P}}(., y))$ and $y \in lfp(\mathcal{IC}^{\mathsf{DMT}^{\mathsf{d}}}_{\mathcal{P}}(x, .))$.

Example 7
Consider the program $\mathcal{P} = \{p \leftarrow \#\mathtt{Sum}[1 : p] > 0; \quad p \leftarrow \#\mathtt{Sum}[1 : p] < 1\}$. $(\{p\}, \{p\})$ is an $\mathsf{DMT}^{\mathsf{d}}$-stable model of $\mathcal{P}$, but the program has no GZ-stable models.

We first explain why $\{p\}$ is not a GZ-stable model. First, we construct $\mathcal{P}^{\{p\}}_{\mathsf{GZ}} = \{p \leftarrow p\}$. Since $\{p\}$ is not a stable model of $\mathcal{P}^{\{p\}}_{\mathsf{GZ}}$, we see that $\{p\}$ is not a GZ-stable model. Likewise,

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

since $\mathcal{P}^{\emptyset}_{\mathsf{GZ}} = \{p \leftarrow \emptyset\}$, we see that $\emptyset$ is not a stable model of $\mathcal{P}^{\emptyset}_{\mathsf{GZ}}$ and therefore not GZ-stable.

To see $\{p\}$ is a $\mathsf{DMT}^{\mathsf{d}}$-stable model, observe that $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}},l}(\emptyset,\{p\}) = \mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}},l}(\{p\},\{p\}) = \{p\}$. Thus, $lfp(\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}},l}(.,\{p\}) = \{p\}$, that is, $(\{p\},\{p\}) = S(\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}^{\mathsf{d}}})(\{p\},\{p\})$.

5.2 Non-deterministic approximation operators for disjunctive aggregate programs

We now proceed to define ndaos for disjunctive aggregate programs. The first ndao we consider generalizes the *trivial* operator (Pelov *et al.* 2007), which maps two-valued interpretations to their immediate consequences, whereas three-valued interpretations are mapped to the least precise pair $(\emptyset,\mathcal{A}_{\mathcal{P}})$ (or, in the non-deterministic case, $\{\emptyset\}\times\{\mathcal{A}_{\mathcal{P}}\}$). We also study the ndao $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}$ based on the deterministic ultimate approximation, and the ultimate ndao $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}$.

Definition 5
Given a disjunctively normal aggregate program $\mathcal{P}$ and a (consistent) interpretation (x,y), let

$$\mathcal{IC}_{\mathcal{P}}^{\mathsf{GZ}}(x,y) = \begin{cases} IC_{\mathcal{P}}(x)\times IC_{\mathcal{P}}(x) & \text{if } x=y \\ \{\emptyset\}\times\{\mathcal{A}_{\mathcal{P}}\} & \text{otherwise} \end{cases}$$

The ndaos $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}$ and $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}$ are defined exactly the same as in Section 3 (recall that $IC_{\mathcal{P}}(x)$ was generalized for aggregates in Section 5.1). We illustrate these semantics with an example:

Example 8
Let $\mathcal{P} = \{r \vee q \leftarrow \#\mathtt{Sum}[1:s] > 0; s \leftarrow \#\mathtt{Sum}[1:r,1:q] > 0\}$ be given.

We first look at $\mathcal{IC}_{\mathcal{P}}^{\mathsf{GZ}}$. As an example of a fixpoint, consider $(\{r,s\},\{r,s\})$. Notice first that $\#\mathtt{Sum}[1:r,1:q] > 0$ and $\#\mathtt{Sum}[1:r,1:q] > 0$ are true in $\{r,s\}$. Thus, $HD_{\mathcal{P}} = \{\{r,q\},\{s\}\}$ and $\mathcal{IC}_{\mathcal{P}}^{\mathsf{GZ}}(\{r,s\},\{r,s\}) = \{\{r,s\},\{q,s\},\{r,q,s\}\}\times\{\{r,s\},\{q,s\},\{r,q,s\}\}$.

We now look at the DMT-semantics. For this, we first calculate $HD_{\mathcal{P}}$ and $IC_{\mathcal{P}}$ for all members of $\wp(\{r,q,s\})$ (with $\Delta_1 = \{\{r\},\{q\},\{r,q\}\}$ and $\Delta_2 = \{\{s,r\},\{s,q\},\{s,r,q\}\}$):

x	$\emptyset$	$\{s\}$	$\{q\}$	$\{r\}$	$\{r,q\}$	$\{r,s\}$	$\{q,s\}$	$\{s,q,r\}$
$HD_{\mathcal{P}}(x)$	$\emptyset$	$\{\{r,q\}\}$	$\{\{s\}\}$	$\{\{s\}\}$	$\{\{s\}\}$	$\{\{r,q\},\{s\}\}$	$\{\{r,q\},\{s\}\}$	$\{\{r,q\},\{s\}\}$
$IC_{\mathcal{P}}(x)$	$\{\emptyset\}$	Δ_1	$\{\{s\}\}$	$\{\{s\}\}$	Δ_2	Δ_2	Δ_2	Δ_2

We then see that, for example, $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}(\{r,s\},\{r,s\}) = \{\{r,s\},\{q,s\},\{r,q,s\}\}\times\{\{r,s\},\{q,s\},\{r,q,s\}\}$, whereas $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}(\emptyset,\{r,s\}) = \{\emptyset\}\times\{\{r,s\},\{q,s\},\{r,q,s\}\}$.

We see that $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}(\{r,s\},\{r,s\}) = \{\{r,s\},\{q,s\},\{r,q,s\}\}\times\{\{r,s\},\{q,s\},\{r,q,s\}\}$, whereas $\mathcal{IC}_{\mathcal{P}}^{\mathcal{U}}(\emptyset,\{r,s\}) = \wp(\{r,s,q\})\times\wp(\{r,s,q\})$.

We now show that these operators are approximation operators with increasing orders of precision: $\mathcal{IC}_{\mathcal{P}}^{\mathsf{GZ}}$ is the least precise, $\mathcal{IC}_{\mathcal{P}}^{\mathsf{DMT}}$ holds a middle ground, and $\mathcal{IC}^{\mathcal{U}}$ is the most precise:

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

Proposition 7
Let some $\xi \in \{\mathsf{DMT}, \mathsf{GZ}, \mathcal{U}\}$ and a disjunctively normal aggregate logic program $\mathcal{P}$ be given. Then $\mathcal{IC}^{\xi}_{\mathcal{P}}(x,y)$ is an ndao approximating $IC_{\mathcal{P}}$. For any (x,y), $\mathcal{IC}^{\mathsf{GZ}}_{\mathcal{P}}(x,y) \preceq^{A}_{i} \mathcal{IC}^{\mathsf{DMT}}_{\mathcal{P}}(x,y) \preceq^{A}_{i} \mathcal{IC}^{\mathcal{U}}_{\mathcal{P}}(x,y)$.

The following properties follow from the general properties shown by Heyninck *et al.* (2022):

Proposition 8
Let some $\xi \in \{\mathsf{DMT}, \mathsf{GZ}, \mathcal{U}\}$ and a disjunctively normal aggregate logic program $\mathcal{P}$ be given. Then: (1) $S(\mathcal{IC}^{\epsilon}_{\mathcal{P}})(x,y)$ exists for any $x, y \subseteq \mathcal{A}_{\mathcal{P}}$, and (2) every stable fixpoint of $\mathcal{IC}^{\epsilon}_{\mathcal{P}}$ is a $\leq_t$-minimal fixpoint of $\mathcal{IC}^{\epsilon}_{\mathcal{P}}$.

The ndao $\mathcal{IC}^{\mathsf{GZ}}_{\mathcal{P}}$ only admits two-valued stable fixpoints, and these two-valued stable fixpoints generalize the GZ-semantics (Gelfond and Zhang 2019):

Proposition 9
If $(x,y) \in \min_{\leq_t}(\mathcal{IC}^{\mathsf{GZ}}_{\mathcal{P}}(x,y))$ then $x = y$. Let a disjunctively normal aggregate aggregate logic program $\mathcal{P}$ s.t. for every $\bigvee \Delta \leftarrow \bigwedge_{i=1}^{n} \alpha_i \wedge \bigwedge_{j=1}^{m} \neg\beta_j \in \mathcal{P}$, β_i is a normal atom be given. $(x,x) \in S(\mathcal{IC}^{\mathsf{GZ}}_{\mathcal{P}})(x,x)$ iff x is a GZ-answer set of $\mathcal{P}$.

We finally show that stable semantics based on $\mathcal{IC}^{\mathsf{DMT}}_{\mathcal{P}}$ generalize those for non-disjunctive logic programs with aggregates by Denecker *et al.* (2002).

Proposition 10
Let a non-disjunctive logic program $\mathcal{P}$ be given. Then, (x,y) is a stable model according to Denecker *et al.* (2002) iff $(x,y) \in S(\mathcal{IC}^{\mathsf{DMT}}_{\mathcal{P}})(x,y)$.

We have shown how semantics for disjunctive aggregate logic programs can be obtained using the framework of non-deterministic AFT, solving the open question (Alviano *et al.* 2023) of how operator-based semantics for aggregate programs can be generalized to disjunctive programs. This means AFT can be unleashed upon disjunctive aggregate programs, as demonstrated in this paper, as demonstrated in this section. Other semantics, such as the weakly supported semantics, the well-founded state semantics (Heyninck *et al.* 2022) and semi-equilibrium semantics (Section 4, as in the appendix of the full version of this article Heyninck and Bogaerts 2023) are obtained without any additional effort and while preserving desirable properties shown algebraically for ndaos. None of these semantics have, to the best of our knowledge, been investigated for dlps with aggregates. Other ndaos, left for future work, can likely be obtained straightforwardly on the basis of deterministic approximation operators for aggregate programs that we did not consider in this paper (e.g., the operator defined by Vanbesien *et al.* (2021) to characterize the semantics of Marek and Remmel (2004) or the bounded ultimate operator introduced by Pelov and Truszczynski (2004).

6 Conclusion, in view of related work

In this paper, we have made three contributions to the theory of non-deterministic AFT: (1) definition of the ultimate operator, (2) an algebraic generalization of the semi-equilibrium semantics, and (3) an application of non-deterministic AFT to dlps with ag-

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

gregates in the body. To the best of our knowledge, there are only a few other semantics that allow for disjunctive rules with aggregates. Among the best-studied is the semantics by Faber *et al.* (2004) (so-called FLP-semantics). As the semantics we propose generalize the operator-based semantics for aggregate programs without disjunction, the differences between the FLP-semantics and the semantics proposed here essentially generalize from the non-disjunctive case (see e.g., Alviano *et al.* 2023).

Among the avenues for future work are an in-depth analysis of the computational complexity of the semantics proposed here, the generalization of the constructions in Section 5 to other semantics (Vanbesien *et al.* 2021; Alviano *et al.* 2023), and defining ndaos for rules with choice constructs in the head (Marek *et al.* 2008), which can be seen as aggregates in the head.

References

Alcântara, J., Damásio, C. V. and Pereira, L. M. 2005. A well-founded semantics with disjunction. In *Proceedings of ICLP'05*. Springer, 341–355.

Alviano, M., Faber, W. and Gebser, M. 2023. Aggregate semantics for propositional answer set programs. *Theory and Practice of Logic Programming 23*, 1, 157–194.

Amendola, G., Eiter, T., Fink, M., Leone, N. and Moura, J. 2016. Semi-equilibrium models for paracoherent answer set programs. *Artificial Intelligence 234*, 219–271.

Bogaerts, B. 2015. *Groundedness in Logics with a Fixpoint Semantics*. Ph.D. thesis, Informatics Section, Department of Computer Science, Faculty of Engineering Science.

Bogaerts, B. 2019. Weighted abstract dialectical frameworks through the lens of approximation fixpoint theory. In *Proceedings of AAAI'19*, vol. 33, 2686–2693.

Denecker, M., Marek, V. and Truszczyński, M. 2000. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In *Logic-based Artificial Intelligence*. Engineering and Computer Science, vol. 597. Springer, 127–144.

Denecker, M., Marek, V. W. and Truszczynski, M. 2002. Ultimate approximations in nonmonotonic knowledge representation systems. In *Proceedings of KR'02*, 177–190.

Faber, W., Leone, N. and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proceedings of JELIA'04*. LNCS, vol. 3229. Springer, 200–212.

Fernández, J. A. and Minker, J. 1995. Bottom-up computation of perfect models for disjunctive theories. *The Journal of Logic Programming 25*, 1, 33–51.

Gelfond, M. and Zhang, Y. 2019. Vicious circle principle, aggregates, and formation of sets in asp based languages. *Artificial Intelligence 275*, 28–77.

Heyninck, J., Arieli, O. and Bogaerts, B. 2022. Non-deterministic approximation fixpoint theory and its application in disjunctive logic programming. arXiv preprint arXiv:2211.17262.

Heyninck, J. and Bogaerts, B. 2023. Non-deterministic approximation operators: Ultimate operators, semi-equilibrium semantics and aggregates (full version). arXiv preprint arXiv:2305.10846.

Marek, V. W., Niemelä, I. and Truszczynski, M. 2008. Logic programs with monotone abstract constraint atoms. *Theory and Practice of Logic Programming 8*, 2, 167–199.

Marek, V. W. and Remmel, J. B. 2004. Set constraints in logic programming. In *Logic Programming and Nonmonotonic Reasoning: 7th International Conference, LPNMR 2004*. Springer, 167–179.

Pearce, D. 2006. Equilibrium logic. *Annals of Mathematics and Artificial Intelligence 47*, 1, 3–41.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press

Pelov, N., Denecker, M. and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming 7*, 3, 301–353.

Pelov, N. and Truszczynski, M. 2004. Semantics of disjunctive programs with monotone aggregates: An operator-based approach. In *Proceedings of NMR'04*, 327–334.

Truszczyński, M. 2006. Strong and uniform equivalence of nonmonotonic theories–an algebraic approach. *Annals of Mathematics and Artificial Intelligence 48*, 3, 245–265.

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

Vanbesien, L., Bruynooghe, M. and Denecker, M. 2021. Analyzing semantics of aggregate answer set programming using approximation fixpoint theory. arXiv preprint arXiv:2104.14789.

https://doi.org/10.1017/S1471068423000236 Published online by Cambridge University Press