A Category-Theoretic Perspective on Higher-Order Approximation Fixpoint Theory

Samuele Pollaci[1,2 [0000−0002−2914−787X]], Babis Kostopoulos[3 [0009−0000−8734−4540]], Marc Denecker[2 [0000−0002−0422−7339]], and Bart Bogaerts[1,2 [0000−0003−3460−4251]]

[1] Vrije Universiteit Brussel, Brussels, Belgium
[2] Katholieke Universiteit Leuven, Leuven, Belgium
[3] Harokopio University of Athens, Athens, Greece
{samuele.pollaci,bart.bogaerts}@vub.be, kostbabis@hua.gr, marc.denecker@kuleuven.be

Abstract. Approximation Fixpoint Theory (AFT) is an algebraic framework designed to study the semantics of non-monotonic logics. Despite its success, AFT is not readily applicable to higher-order definitions. To solve such an issue, we devise a formal mathematical framework employing concepts drawn from Category Theory. In particular, we make use of the notion of Cartesian closed category to inductively construct higher-order approximation spaces while preserving the structures necessary for the correct application of AFT. We show that this novel theoretical approach extends standard AFT to a higher-order environment, and generalizes the AFT setting of [7].

Keywords: Approximation fixpoint theory · Higher-order definitions · Category theory.

1 Introduction

Approximation Fixpoint Theory (AFT) [12] is an algebraic framework designed to study the semantics of non-monotonic logics. It was originally designed for characterizing the semantics of logic programming, autoepistemic logic, and default logic, and to resolve longstanding problems on the relation between these formalisms [13]. Later, it has also been applied to a variety of other domains, including abstract argumentation [2, 21], active integrity constraints [4], stream reasoning [1], and integrity constraints for the semantic web [5].

The core ideas of AFT are relatively simple: we are interested in fixpoints of an operator O on a given lattice $\langle L, \leq \rangle$. For monotonic operators, Tarski's theory guarantees the existence of a least fixpoint, which is of interest in many applications. For non-monotonic operators, the existence of fixpoints is not guaranteed; and even if fixpoints exist, it is not clear which would be "good" fixpoints. AFT generalizes Tarki's theory for monotonic operators by making use of a so-called *approximating operator*; this is an operator $A : L^2 \to L^2$, that operates on L^2, and that is monotonic with respect to the precision order $\leq_p$ (defined by $(x, y) \leq_p (u, v)$ if $x \leq u$ and $v \leq y$)). The intuition is that elements

of L^2 approximate elements of L: $(x, y) \in L^2$ approximates z if $x \leq z \leq y$, i.e., when $x \leq y$, the tuple (x, y) can be thought of as an interval in L. Given such an approximator, AFT defines several types of fixpoints (supported fixpoints, a Kripke-Kleene fixpoint, stable fixpoints, a well-founded fixpoint) of interest.

In several fields of non-monotonic reasoning, it is relatively straightforward to define an approximating operator and it turns out that the different types of fixpoints then correspond to existing semantics. In this way, AFT clarifies on the one hand how different semantics in a single domain relate, and on the other hand what the relation is between different (non-monotonic) logics.

Let us illustrate the application of AFT to standard, first-order, logic programming. In this setting, the lattice L is the lattice of interpretations with the truth order $I \leq J$ if $P^I \subseteq P^J$ for each predicate P. The operator is the immediate consequence operator T_P, as defined in the seminal work of van Emden and Kowalski [22]. Given a logic program (i.e., a set of rules), this operator has the property that q holds in $T_P(I)$ if and only if there is a rule $q \leftarrow \varphi$ in P such that φ is true in I. In this setting, pairs (I, J) are seen as *four-valued* interpretations: I represents what is *true* and J what is *possible*. A fact is then *true* (resp. *false*) if it is true (resp. false) in both I and J, *unknown* if it is true in J but not true in I and *inconsistent* if it is true in I but not in J. The approximating operator Ψ_P is, in this case, nothing else than Fitting's (2002) four-valued immediate consequence operator, which uses Kleene's truth tables to evaluate the body of each rule in a four-valued interpretation. For this approximator, the fixpoints defined by AFT correspond to the major semantics of logic programming: supported fixpoints are models of Clark's completion [8], stable fixpoints correspond to (partial) stable models [18], the Kripke-Kleene fixpoint to the Kripke-Kleene model [16] and the well-founded fixpoint is the well-founded model [23].

This paper is motivated by a need to apply AFT to *higher-order* logic programming that arose in several contexts [7, 9, 10]. An important issue that arises in this context is that using pairs of interpretations no longer allows for an obvious way to evaluate formulas in an approximation. Let us illustrate this with an example. Consider a logic program in which a first-order predicate p and a second-order predicate Q are defined. Now assume that in the body of a rule, the atom $Q(p)$ occurs. A tuple (I, J) of interpretations in this case tells us for any given set S if $Q(S)$ is true, false, unknown, or inconsistent. However, the interpretation of p via (I, J) is not a set, but a partially defined set, making it hard to evaluate expressions of the form $Q(p)$. In other words, an approximation of the interpretation of Q has to take as argument not only sets, i.e., *exact* elements, but also partially defined sets, i.e., *approximate* elements, like the interpretation of p in this example. Thus, a richer space of approximations where approximate objects can be applied to other approximate objects is needed.

The above example and considerations suggest that spaces of approximations of higher-order objects should be defined inductively from lower-order ones, following the type hierarchy: we start by assigning a *base approximation space* to each type at the bottom of the hierarchy, and then, for each composite type $\tau_1 \to \tau_2$, we define its approximation space as *a certain class of functions* from

the approximation space for τ_1 to the approximation space for τ_2, and so on. This method was heavily inspired by the approach of Charalambidis et al. [7] to obtain a generalization of the well-founded semantics for higher-order logic programs with negation. Notice that there are two major points in the construction above which are yet not defined: the base approximation spaces, and the class of functions we consider. The main question of this paper is how to define them in a generic way that works in all applications of AFT.

We want to apply the same AFT techniques on approximation spaces at any hierarchy level, i.e., on base approximation spaces and the aforementioned sets of functions, which should thus have the same algebraic structure. In Category Theory (CT), the notion of *Cartesian closed category* captures this behavior. A category consists of a collection of *objects* and a collection of *morphisms*, i.e., relations between objects. For example, we can define the *category of square bilattices* as the one having square bilattices as objects, and monotone functions as morphisms. The objects of a Cartesian closed category $\mathbf{C}$ satisfy a property that can be intuitively understood as follows: *if A and B are two objects of $\mathbf{C}$, then the set of morphisms from A to B is also an object of $\mathbf{C}$*. Hence, if the base approximation spaces are objects of a Cartesian closed category, then the category contains the full hierarchy of spaces we are aiming for. We call such a Cartesian closed category an *approximation category* and denote it by **Approx**.

In the category-theoretic framework, the questions on the nature of the base approximation spaces and the class of functions reduce to defining the objects and the morphisms of **Approx**. Clearly, this depends on the application we want to use AFT for. Different applications imply different higher-order languages, with different types, and possibly different versions of AFT (standard AFT [12], consistent AFT [14], or other extensions [7]). To formalize this, we develop the notion of an *approximation system*. Once a language and the semantics of its types are fixed, we can choose an approximation system that consists, among other things, of a Cartesian closed category **Approx**, equipped with a function *App* associating the semantics of a type to an approximation space in **Approx**. The approximation system also determines which elements of the approximation spaces are *exact*, i.e., which elements approximate exactly one element of the semantics of a type, and, for every type, it provides a projection from the exact elements to the objects they represent in the corresponding semantics. This is non-trivial for higher-order approximation spaces, and it is indeed fundamental to obtain a sensible account for AFT for higher-order definitions.

In short, the main contributions of our paper are as follows:

1. We generalize the work of Charalambidis et al. [7] to a category-theoretic setting. In doing so, we shed light on the general principles underlying their constructions for higher-order logic programimng and make their construction applicable to arbitrary current and future non-monotonic reasoning formalism captured by AFT.
2. We extend the work of Charalambidis et al. [7] by studying *exactness* and in this way also allow using the theory to define exact *stable models* instead of focusing purely on the well-founded model.

In recent work, a stable semantics for higher-order logic programs was defined building on consistent AFT [3]. In that work, the approach taken to evaluate an expression of the form $Q(p)$, instead of applying an approximate interpretation for Q to an approximate interpretation for p, is to apply the approximate interpretation for Q to *all* exact interpretations for p that are still possible, and returning the least precise approximation of all the results. What this means in effect is that some sort of *ultimate construction* [15] is used; this has also been done in other extensions of logic programming [9, 19]. Bogaerts et al. [3] also discovered a flaw in the work of Charalambidis et al. [7], namely that even for simple non-recursive programs, the well-founded model might not behave as expected (leaving all atoms unknown). It is important to mention, though, that this aberration is caused solely by the treatment of (existentially) quantified variables and not by the algebraic theory (which is the focus of the current paper). Finally, it is interesting to mention that another paper [6] joined CT and AFT, albeit with different perspective and aim: while we treat the whole set of possible approximation spaces as a category to apply any account of AFT to higher-order definitions, Charalambidis and Rondogiannis [6] view the approximation spaces themselves as categories to provide a novel version of standard AFT.

The rest of this paper is structured as follows. In Section 2, we provide an overview of the fundamental concepts from AFT and CT that we use. Section 3 presents the novel definitions of approximation system, with the category **Approx**, and of exact elements of an approximation space. In Section 4, we show that the approximation spaces from [7] form a Cartesian closed category: the objects are the chain-complete posets of the form $L \otimes U$, where, intuitively, L represents a set of lower bounds and U a set of upper bounds. Our approach is not just able to reconstruct in a simple way (using the general principles outlined above) the semantic elements defined ad-hoc by Charalambidis et al. [7], but it also resolves a question that was left open in that work. Namely, what we get now is a clear definition for exact higher-order elements, and, in particular, it allows to determine when a model of a program is two-valued (see Example 2). We conclude in Section 5.

Because of space constraints, all the proofs, some additional definitions and propositions are provided in the extended version of this paper [20].

2 Preliminaries

In this section, we provide a concise introduction to the formal concepts we utilize throughout the paper. First, we outline the core ideas at the foundation of AFT. Then, we present the notions of CT we need, with the definition of Cartesian closed category being the key concept.

2.1 Approximation Fixpoint Theory

AFT generalizes Tarki's theory to non-monotonic operators, with the initial goal of studying the semantics of non-monotonic logics. As such, AFT heavily relies on the following notions from order theory.

A *partially ordered set* (poset) $\mathcal{P}$ is a set equipped with a partial order, i.e., a reflexive, antisymmetric, transitive relation. We denote a poset by $\mathcal{P} = \langle P, \leq_P \rangle$, where P is the underlying set, and $\leq_P$ the partial order. By abuse of notation, we sometimes use the calligraphic poset notation $\mathcal{P}$ in place of the notation for its underlying set P, and vice versa. We denote by $\mathcal{P}^{op}$ the poset with the same underlying set as $\mathcal{P}$ but opposite order, i.e., $\mathcal{P}^{op} = \langle P, \geq_P \rangle$. Given a subset $S \subseteq P$, a lower bound l of S is the *greatest lower bound of* S, denoted by $\bigsqcap S$, if it is greater than any other lower bound of S. Analogously, an upper bound u of S is the *least upper bound of* S, denoted by $\bigsqcup S$, if it is lower than any other upper bound of S. A *chain complete poset* (cpo) is a poset $\mathcal{C}$ such that for every chain $S \subseteq C$, i.e., a totally ordered subset, $\bigsqcup S$ exists. A *complete join semilattice* is a poset $\mathcal{J}$ such that for any subset $S \subseteq J$, $\bigsqcup S$ exists. A *complete lattice* is a poset $\mathcal{L}$ such that for any subset $S \subseteq L$, both $\bigsqcap S$ and $\bigsqcup S$ exist. A function $f \colon \mathcal{P}_1 \to \mathcal{P}_2$ between posets is *monotone* if for all $x, y \in \mathcal{P}_1$ such that $x \leq_{P_1} y$, it holds that $f(x) \leq_{P_2} f(y)$. We refer to functions $O \colon \mathcal{C} \to \mathcal{C}$ with domain equal to the codomain as *operators*. An element $x \in \mathcal{C}$ is a *fixpoint* of O if $O(x) = x$. By Tarski's least fixpoint theorem, every monotone operator O on a cpo has a least fixpoint, denoted $\mathrm{lfp}(O)$. To use a similar principle for operators stemming from non-monotonic logics, standard AFT [12] considers, for each complete lattice $\mathcal{L}$, its associated square bilattice $\langle L^2, \leq_p \rangle$, where $\leq_p$ is the *precision order* on the Cartesian product L^2, i.e., $(x_1, y_1) \leq_p (x_2, y_2)$ iff $x_1 \leq_L x_2$ and $y_2 \leq_L y_1$. A square bilattice $\langle L^2, \leq_p \rangle$ can be viewed as an approximation of L: an element $(x, y) \in L^2$ such that $x \leq_L y$ "approximates" all the values $z \in L$ such that $x \leq_L z \leq_L y$. Such pairs (x, y) with $x \leq_L y$ are called *consistent*. Pairs of the form $(x, x) \in L^2$ are called *exact*, since they approximate only one element of L.

An *approximator* $A \colon L^2 \to L^2$ is a monotone operator that is *symmetric*, i.e., for all $(x, y) \in L^2$ it holds that $A_1(x, y) = A_2(y, x)$, where $A_1, A_2 \colon L^2 \to L$ are the components of A, i.e. $A(x, y) = (A_1(x, y), A_2(x, y))$. An approximator $A \colon L^2 \to L^2$ *approximates* an operator $O \colon L \to L$ if for all $x \in L$, $A(x, x) = (O(x), O(x))$. Since A is by definition monotone, by Tarski's theorem A has a least fixpoint, called the *Kripke-Kleene* fixpoint. Moreover, given an approximator A, there are three other operators which deserve our attention, together with their fixpoints: the operator approximated by A, $O_A \colon x \in L \mapsto A_1(x, x) \in L$ whose fixpoints are called *supported*; the *stable operator* $S_A \colon x \in L \mapsto \mathrm{lfp}(A_1(\cdot, x)) \in L$ with the *stable* fixpoints (where $A_1(\cdot, x) \colon y \in L \mapsto A_1(y, x) \in L$); and the *well-founded operator* $\mathcal{S}_A \colon (x, y) \in L^2 \mapsto (S_A(y), S_A(x)) \in L^2$, whose least fixpoint is referred to as the *well-founded* fixpoint. If A is the four-valued immediate consequence operator [17], then the aforementioned four types of fixpoint correspond to the homonymous semantics of logic programing [11, 12].

2.2 Cartesian Closed Categories

Category Theory (CT) studies mathematical structures and the relations between them, through the notion of a *category*. Intuitively, a category $\mathbf{C}$ consists of a collection $\mathrm{Ob}(\mathbf{C})$ of *objects* and a collection $\mathrm{Mor}(\mathbf{C})$ of relations, called *morphisms*, between objects, satisfying some basic properties: every morphism f has a *domain*

$s(f)$ and a *codomain* $t(f)$, morphisms can be composed, and so on[20]. In this paper, objects will always be certain ordered sets, and morphisms will be monotone functions. We say that $\mathbf{D}$ is a *full subcategory* of $\mathbf{C}$, denoted as $\mathbf{D} \subseteq \mathbf{C}$, if $\mathrm{Ob}(\mathbf{D}) \subseteq \mathrm{Ob}(\mathbf{C})$, and $\mathrm{Mor}(\mathbf{D}) = \{f \mid f \in \mathrm{Mor}(\mathbf{C}) \text{ such that } t(f), s(f) \in \mathrm{Ob}(\mathbf{D})\}$.

It is easy to see that we can define a category **POSet** with objects the posets, and as morphisms the monotone functions between posets. We denote by **CPO**, **CJSLat**, and **CLat** the full subcategories of **POSet** with objects the cpo's, the complete join semilattices, and the complete lattices, respectively.

We are interested in inductively building *approximation spaces* for higher-order concepts starting from *base* ones. To be able to perform this construction, we need the approximation spaces to belong to a *Cartesian closed* category, i.e., a category $\mathbf{C}$ with a *terminal object*, *products*, and *exponentials*. Let us briefly explain what these concepts are (for the formal definitions we refer to [20]). An object $T \in \mathrm{Ob}(\mathbf{C})$ is *terminal* if for each $A \in \mathrm{Ob}(\mathbf{C})$ there exists a unique morphism $f\colon A \to T$. For instance, the poset with one element and trivial order is the terminal object of **POSet**, **CPO**, **CJSLat**, and **CLat**. Given $A_1, A_2 \in \mathrm{Ob}(\mathbf{C})$, the *product* of A_1 and A_2 is an object $A_1 \times A_2 \in \mathrm{Ob}(\mathbf{C})$ equipped with two *projection* morphisms $\pi_1\colon A_1 \times A_2 \to A_1$ and $\pi_2\colon A_1 \times A_2 \to A_2$ satisfying a universal property. In **POSet**, and analogously for **CPO**, **CJSLat**, and **CLat**, the product of two objects $\mathcal{P}_1$ and $\mathcal{P}_2$ is the Cartesian product of P_1 and P_2 equipped with the *product order*, i.e., $(x_1, y_1) \leq (x_2, y_2)$ if and only if $x_1 \leq_{P_1} x_2$ and $y_1 \leq_{P_2} y_2$. The projections π_1 and π_2 are given by the usual Cartesian projections. Finally, given $A_1, A_2 \in \mathrm{Ob}(\mathbf{C})$, the *exponential* of A_1 and A_2 is an object $A_2^{A_1} \in \mathrm{Ob}(\mathbf{C})$ equipped with a morphism $ev\colon A_2^{A_1} \times A_1 \to A_2$, called the *evaluation*, satisfying a universal property. In **POSet**, and analogously for **CPO**, **CJSLat**, and **CLat**, the exponential $\mathcal{P}_2^{\mathcal{P}_1}$ is the set of monotone functions from P_1 to P_2 equipped with the *pointwise order* (induced by $\leq_{P_2}$), i.e., $f_1 \leq_{pt} f_2$ if and only if for all $x \in P_1$, $f_1(x) \leq_{P_2} f_2(x)$. The evaluation ev is given by the usual function evaluation, i.e., $ev(f, x) = f(x)$. Clearly, it follows that **POSet**, **CPO**, **CJSLat**, and **CLat** are all Cartesian closed.

In the context of AFT, we are often interested in the space of interpretations over a possibly infinite vocabulary. To include this, we need a category in which a product over an arbitrary indexed family of objects is again an object of the category. We say that such a category $\mathbf{C}$ *has generalized products*, i.e., for all families $\{A_i\}_{i \in I}$ of objects of $\mathbf{C}$ the product $\Pi_{i \in I} A_i \in \mathrm{Ob}(\mathbf{C})$ exists [20].

Proposition 1. *If* $\mathbf{C} \subseteq$ **POSet**, *then* $\mathbf{C}$ *has generalized products.*

Given $\mathcal{P}_1, \mathcal{P}_2 \in \mathrm{Ob}(\mathbf{POSet})$, we denote by $(\mathcal{P}_1 \to \mathcal{P}_2) \in \mathrm{Ob}(\mathbf{POSet})$ the poset of functions from $\mathcal{P}_1$ to $\mathcal{P}_2$ ordered with the pointwise order. In particular, notice that $(\mathcal{P}_1 \to \mathcal{P}_2)$ may contain non-monotone functions.

Proposition 2. *Let* $\mathbf{C}$ *be a full subcategory of* **POSet**, $X \in \mathrm{Ob}(\mathbf{POSet})$, *and* $Y \in \mathrm{Ob}(\mathbf{C})$. *Then there exists an isomorphism* $(X \to Y) \cong \Pi_{x \in X} Y$ *in* $\mathbf{C}$.

3 The Approximation System

In this section, we introduce the notions of *approximation category* and of *approximation system*, which constitute the core of the theoretical framework for AFT we developed.

Let $\mathcal{L}$ be a higher-order language based on a hierarchy of types $\mathbb{H}$ comprising of *base types* τ, and two kinds of composite types: *product types* $\Pi_{i\in I}\tau_i$, and *morphism types* $\tau_1 \to \tau_2$. For instance, a base type could be the boolean type o or the type ι of individuals, whereas in the composite types we may find the type $\iota \to o$, which is the type of unary first-order predicates. We denote by $\mathcal{B}_{\mathbb{H}}$ the set of base types. For the sake of simplicity, we omit the subscript of $\mathcal{B}$ when it is clear from the context of use. We associate to each type τ of $\mathcal{B}_{\mathbb{H}}$, an object $E_\tau \in \mathrm{Ob}(\mathbf{POSet})$, and we define inductively for all $\{\tau_i\}_{i\in I} \subseteq \mathbb{H}$, $E_{\Pi_{i\in I}\tau_i} = \Pi_{i\in I}E_{\tau_i}$, and for all $\tau_1, \tau_2 \in \mathbb{H}$, $E_{\tau_1\to\tau_2} = (E_{\tau_1} \to E_{\tau_2})$. The object E_τ is called the *semantics of* τ. For example, if the semantics of the boolean type o is chosen to be $E_o := \{\mathbf{f}, \mathbf{t}\}$ with the standard truth ordering, then the semantics for type $o \to o$ is the poset of functions from E_o to E_o.

In many applications of AFT, we are ultimately interested in the space of interpretations, which associate to each symbol of a vocabulary, an element of the semantics of the type of such symbol. It follows that an interpretation can be seen as a tuple of elements of different semantics. In more detail, given a vocabulary V, we can consider the product type $\tau = \Pi_{s\in V}t(s)$, where $t(s)$ is the type of the symbol s. Then, the space of interpretations for the vocabulary V coincides with the semantics $E_\tau = \Pi_{s\in V}E_{t(s)}$.

We have so far defined the semantics of all the base types and the composite ones constructed from them. Notice that, it is often not necessary to define the spaces of approximations for all such semantics E_τ, which are infinitely many. Because of the nature of our formalism, we can easily restrict the set of types we take into account: we can fix a subset $\mathbb{T} \subseteq \mathbb{H}$ of types, and focus our attention onto the set $S_{\mathbb{T}}$ defined as follows:

- for all $\tau \in \mathbb{T}$, $E_\tau \in S_{\mathbb{T}}$,
- if $E_{\tau_1\to\tau_2} \in S_{\mathbb{T}}$, then $E_{\tau_2} \in S_{\mathbb{T}}$,
- if $E_{\Pi_{i\in I}\tau_i} \in S_{\mathbb{T}}$, then $E_{\tau_i} \in S_{\mathbb{T}}$ for all $i \in I$.

We will dive deeper into this matter in Section 4 where we present applications of our framework. We denote by $\mathcal{B}_{\mathbb{T}}$ the set of base types of $\mathbb{H}$ belonging to $\mathbb{T}$.

The notion of *approximation system* (Definition 1) together with what follows in this section, provide a general framework in which the techniques of AFT can be applied on higher-order languages. Before stating the, rather lengthy, definition of an approximation system, we provide an intuitive understanding of its components.

For each $E_\tau \in S_{\mathbb{T}}$, we shall consider a corresponding space $App(E_\tau)$, called an *approximation space*, whose elements approximate the elements of E_τ. Hence, we define a Cartesian closed full subcategory of **CPO**, denoted by **Approx**, and a map $App\colon S_{\mathbb{T}} \to \mathrm{Ob}(\mathbf{Approx})$ encoding such correspondence. The fact that $\mathbf{Approx} \subseteq \mathbf{CPO}$ allows us to apply the Knaster-Tarski theorem on the

approximation spaces, and guarantees the existence of generalized products (Proposition 1). Notice that, even though we fixed a mapping App between the set $S_{\mathbb{T}}$ and the objects of **Approx**, there is, so far, no relation between the elements of E_τ and those of $App(E_\tau)$. The approximation space $App(E_\tau)$ is meant to approximate the elements of E_τ. In particular, we want the order $\leq_{App(E_\tau)}$ on $App(E_\tau)$, which we call a *precision order*, to encode the approximating nature of $App(E_\tau)$ for E_τ: intuitively, $a \leq_{App(E_\tau)} b$ if a is *less precise* than b, i.e., if an element $e \in E_\tau$ is approximated by b, then e is also approximated by a. In the context of AFT, of particular interest are the elements of $App(E_\tau)$ which approximate just one element, called the *exact* elements. Thus, in the definition of approximation system that we are about to give, for every base type $\tau \in \mathcal{B}_{\mathbb{T}}$, we fix a set $\mathcal{E}_\tau$ of exact elements of $App(E_\tau)$, and a function $\mathfrak{p}^0_\tau \colon \mathcal{E}_\tau \to E_\tau$, which associates each exact element to the unique element of E_τ it represents.

To obtain a sensible framework, it is fundamental to carefully define the sets of exact elements and a projection that associates each exact element to the object it represents. Hence, we impose conditions on the possible choices of the sets $\mathcal{E}_\tau$ and the functions $\mathfrak{p}^0_\tau$, for $\tau \in \mathcal{B}_{\mathbb{T}}$. Since an exact element of $App(E_\tau)$ approximates a single element of the semantics E_τ, if both a and b are exact and one is more precise than the other, then they should represent the same element, i.e. $\mathfrak{p}^0_\tau(a) = \mathfrak{p}^0_\tau(b)$ (Item 4b in Definition 1). This requirement also hints at a very important fact: the definition of approximation system allows for the existence of multiple exact elements of $App(E_\tau)$ representing the *same* element of E_τ. Because of this possible multitude of exact representatives, we want to have, for each element $e \in E_\tau$, a natural choice for a representative in the approximation space $App(E_\tau)$. This is why, for each element $e \in E_\tau$, we require that the greatest lower bound of all the exact elements representing e exists, is exact, and represents e (Item 4c in Definition 1). Lastly, we add one more condition on exact elements to accommodate several existing versions of AFT. In consistent AFT [14], exact elements are maximal, while in standard AFT, this is not the case, and there are elements beyond exact ones. We require that either the exact elements are maximal, or we can take arbitrary joins in the approximation spaces (Item 3b in Definition 1). This last condition will later allow for a generalization of both $\mathcal{E}_\tau$ and $\mathfrak{p}^0_\tau$ to any type τ of $\mathbb{H}$, satisfying properties analogous to the ones required for the base types counterparts (Propositions 3 and 4).

We are now ready to state the definition of an approximation system. We write $f^{-1}(b)$ for the *preimage* of an element $b \in B$ via a function $f \colon A \to B$, i.e., $f^{-1}(b) = \{a \mid f(a) = b\} \subseteq A$.

Definition 1. *A tuple* $(\mathbf{Approx}, App, \{\mathcal{E}_\tau\}_{\tau\in\mathcal{B}}, \{\mathfrak{p}^0_\tau\}_{\tau\in\mathcal{B}})$ *is an* approximation system (for $S_{\mathbb{T}}$) *if*

1. **Approx** *is a Cartesian closed full subcategory of* **CPO**, *called the* approximation category. *The objects of* **Approx** *are called* approximation spaces.
2. $App \colon S_{\mathbb{T}} \to \mathrm{Ob}(\mathbf{Approx})$ *is a function such that for all* $E_\tau \in S_{\mathbb{T}}$
   (a) *if* $\tau = \Pi_{i\in I}\tau_i$ *is a product type, then* $App(E_\tau) = \Pi_{i\in I} App(E_i)$,
   (b) *if* $\tau = \tau_1 \to \tau_2$ *and* $E_{\tau_1} \notin S_{\mathbb{T}}$, *then* $App(E_{\tau_1\to\tau_2}) = App(\Pi_{i\in E_{\tau_1}} E_{\tau_2})$,

   (c) *if* $\tau = \tau_1 \to \tau_2$ *and* $E_{\tau_1} \in S_{\mathbb{T}}$, *then* $App(E_{\tau_1 \to \tau_2}) = App(E_{\tau_2})^{App(E_{\tau_1})}$.
3. $\{\mathcal{E}_\tau\}_{\tau \in \mathcal{B}}$ *is a family of sets such that the following hold:*
   (a) *for each base type* $\tau \in \mathcal{B}$, $\mathcal{E}_\tau \subseteq App(E_\tau)$,
   (b) *either* $App(E_\tau) \in \mathrm{Ob}(\mathbf{CJSLat})$ *for all* $\tau \in \mathcal{B}$, *or for all* $\tau \in \mathcal{B}$, *if* $a \in \mathcal{E}_\tau$ *and* $b \in App(E_\tau)$ *such that* $a \leq_{App(E_\tau)} b$, *then also* $b \in \mathcal{E}_\tau$.
4. $\{\mathfrak{p}^0_\tau\}_{\tau \in \mathcal{B}}$ *is a family of surjective functions such that for each base type* $\tau \in \mathcal{B}$:
   (a) $\mathfrak{p}^0_\tau \colon \mathcal{E}_\tau \to E_\tau$,
   (b) *for all* $a, b \in \mathcal{E}_\tau$, *if* $a \leq_{App(E_\tau)} b$, *then* $\mathfrak{p}^0_\tau(a) = \mathfrak{p}^0_\tau(b)$,
   (c) *for all* $e \in E_\tau$, *there exists* $\bigsqcap((\mathfrak{p}^0_\tau)^{-1}(e)) \in \mathcal{E}_\tau$ *and* $\mathfrak{p}^0_\tau(\bigsqcap(\mathfrak{p}^0_\tau)^{-1}(e)) = e$.

Observe that, by Proposition 2, it holds that $E_{\tau_1 \to \tau_2} = (E_{\tau_1} \to E_{\tau_2}) \cong \Pi_{i \in E_{\tau_1}} E_{\tau_2} = E_{\Pi_{i \in E_{\tau_1}} \tau_2}$. However, in Item 2c of the above definition, we have $App(E_{\tau_1 \to \tau_2}) = App(E_{\tau_2})^{App(E_{\tau_1})} \ncong \Pi_{i \in E_{\tau_1}} App(E_{\tau_2}) = App(E_{\Pi_{i \in E_{\tau_1}} \tau_2})$. Hence, while the map App, in a way, respects the structure given by the type hierarchy $\mathbb{H}$, it does not commute with isomorphisms of posets. Moreover, it is important to notice that, while the approximation system depends on the application at hand, i.e., on the language, the semantics, and so on, the approximation category depends only on the version of AFT.

We now fix an approximation system $\mathcal{S} = (\mathbf{Approx}, App, \{\mathcal{E}_\tau\}_{\tau \in \mathcal{B}}, \{\mathfrak{p}^0_\tau\}_{\tau \in \mathcal{B}})$ for $S_{\mathbb{T}}$, and extend the notion of exactness to all approximation spaces.

Definition 2. *Let* $E_\tau \in S_{\mathbb{T}}$. *An element* $e \in App(E_\tau)$ *is* exact *if one of the following conditions holds:*

1. $\tau \in \mathcal{B}_{\mathbb{T}}$ *and* $e \in \mathcal{E}_\tau$,
2. $\tau = \Pi_{i \in I} \tau_i$ *and for each* $i \in I$, *the* i*-th component* $\pi_i(e)$ *of* e *is exact,*
3. $\tau = \tau_1 \to \tau_2$, $E_{\tau_1} \notin S_{\mathbb{T}}$, *and every component of* e *is exact,*
4. $\tau = \tau_1 \to \tau_2$, $E_{\tau_1} \in S_{\mathbb{T}}$, *and for all* $e_1 \in App(E_{\tau_1})$ *exact,* $e(e_1) \in App(E_{\tau_2})$ *is exact.*

For $\tau \in \mathbb{T}$, we denote by $\mathcal{E}_\tau$ the subset of $App(E_\tau)$ of exact elements of type τ. Intuitively, the reason for Item 4 in the definition above is that it allows us to project exact elements of type $\tau_1 \to \tau_2$ to functions in $E_{\tau_1 \to \tau_2}$. For each $E_\tau \in S_{\mathbb{T}}$, we denote by $\mathcal{E}_\tau$ the set of exact elements of $App(E)$. The following proposition shows that the condition 3b of Definition 1 holds for any $E_\tau \in S_{\mathbb{T}}$.

Proposition 3. *Let* $E_\tau \in S_{\mathbb{T}}$. *Either for all* $\sigma \in \mathcal{B}$ *it holds that* $App(E_\sigma) \in \mathrm{Ob}(\mathbf{CJSLat})$, *or for all* $b \in App(E_\tau)$ *and* $e \in \mathcal{E}_\tau$, *if* $e \leq_{App(E_\tau)} b$, *then* $b \in \mathcal{E}_\tau$.

Now that we have defined the exact elements for any semantics in $S_{\mathbb{T}}$, we extend the family $\{\mathfrak{p}^0_\tau\}_{\tau \in \mathcal{B}}$ to have a map for each $E_\tau \in S_{\mathbb{T}}$. We can do this inductively, by defining a new family of functions $\{\mathfrak{p}_\tau \colon \mathcal{E}_\tau \to E_\tau\}_{E_\tau \in S_{\mathbb{T}}}$ as follows:

1. if $\tau \in \mathcal{B}$, then $\mathfrak{p}_\tau := \mathfrak{p}^0_\tau$,
2. if $\tau = \Pi_{i \in I} \tau_i$, then for all $(e_i)_{i \in I} \in \mathcal{E}_\tau$, $\mathfrak{p}_\tau((e_i)_{i \in I}) := (\mathfrak{p}_{\tau_i}(e_i))_{i \in I}$,
3. if $\tau = \tau_1 \to \tau_2$, and $E_{\tau_1} \notin S_{\mathbb{T}}$, then for all $(e_i)_{i \in E_{\tau_1}} \in \mathcal{E}_\tau$, for all $x \in E_{\tau_1}$, $\mathfrak{p}_\tau((e_i)_{i \in E_{\tau_1}})(x) := \mathfrak{p}_{\tau_2}(e_x)$,

4. if $\tau = \tau_1 \to \tau_2$, and $E_{\tau_1} \in S_{\mathbb{T}}$, then for all $f \in \mathcal{E}_\tau$, and for all $e \in E_{\tau_1}$, $\mathfrak{p}_\tau(f)(e) := \mathfrak{p}_{\tau_2}(f(d))$, where $d \in \mathfrak{p}_{\tau_1}^{-1}(e)$, i.e., $\mathfrak{p}_{\tau_1}(d) = e$.

In the following proposition, we prove that for each $E_\tau \in S_{\mathbb{T}}$, the function $\mathfrak{p}_\tau$ is well-defined, surjective, and satisfies properties analogous to 4b and 4c of Definition 1.

Proposition 4. *Let $E_\tau \in S_{\mathbb{T}}$, $e_1, e_2 \in \mathcal{E}_\tau$, and $e \in E_\tau$. Then, $\mathfrak{p}_\tau$ is well-defined and surjective, there exists $\bigsqcap \mathfrak{p}_\tau^{-1}(e) \in \mathcal{E}_\tau$, and $\mathfrak{p}_\tau(\bigsqcap \mathfrak{p}_\tau^{-1}(e)) = e$. Moreover, if $e_1 \leq_{App(E_\tau)} e_2$, then $\mathfrak{p}_\tau(e_1) = \mathfrak{p}_\tau(e_2)$.*

In most applications of AFT, for approximation spaces of base types, there exists a unique exact element representing an object of a semantics. However, for higher-order approximation spaces, this is not always the case, as we illustrate in the following example.

Example 1. Let o be the Boolean type, with semantics $E_o := \langle \{\mathbf{f}, \mathbf{t}\}, \leq_t \rangle$, where $\leq_t$ is the standard truth order. In standard AFT, we would define the approximation space for E_o to be the bilattice $App(E_o) := \langle E_o \times E_o, \leq_p \rangle$, with $\leq_p$ the precision order. Then, the semantics for $o \to o$ is the poset of functions from E_o to E_o, and the approximation space for it is the exponential, i.e., the set of monotone functions from $App(E_o)$ to itself, ordered pointwise. Clearly, we can set the exact elements of $App(E_o)$ to be $(\mathbf{f}, \mathbf{f})$ and $(\mathbf{t}, \mathbf{t})$, and $\mathfrak{p}_o$ to send them to $\mathbf{f}$ and $\mathbf{t}$, respectively. Now consider the following two functions: $f, g \colon App(E_o) \to App(E_o)$ defined by $f(\mathbf{f}, \mathbf{t}) = (\mathbf{f}, \mathbf{t})$, $g(\mathbf{f}, \mathbf{t}) = f(\mathbf{f}, \mathbf{f}) = g(\mathbf{f}, \mathbf{f}) = f(\mathbf{t}, \mathbf{t}) = g(\mathbf{t}, \mathbf{t}) = (\mathbf{t}, \mathbf{t})$, and $f(\mathbf{t}, \mathbf{f}) = g(\mathbf{t}, \mathbf{f}) = (\mathbf{t}, \mathbf{f})$. Clearly, both f and g send exacts to exacts, thus, they are exact. Moreover, even though $f \neq g$, it is easy to see that $\mathfrak{p}_{o\to o}(f) = \mathfrak{p}_{o\to o}(g) = h \colon E_o \to E_o$, where $h(\mathbf{f}) = h(\mathbf{t}) = \mathbf{t}$.

Definition 3. *Let $E_\tau \in S_{\mathbb{T}}$. An element $c \in App(E_\tau)$ is* consistent *if there exists $e \in \mathcal{E}_\tau$ such that $c \leq_{App(E_\tau)} e$.*

Notice that a function of the family $\{\mathfrak{p}_\tau \colon \mathcal{E}_\tau \to E_\tau\}_{E_\tau \in S_{\mathbb{T}}}$ not only determines which element of the semantics an exact element represents, but it also helps understanding what a consistent element is approximating: if $c \in App(E_\tau)$ is consistent and $c \leq_{App(E_\tau)} e$ for some exact e, then c approximates $\mathfrak{p}_\tau(e)$. Clearly, consistent elements may approximate more than one element of a semantics.

4 An Approximation System for Extended Consistent AFT

In this section, we present the application of our framework for the extension of consistent AFT developed by Charalambidis et al. [7]. They considered a new class of approximation spaces, of the form $L \otimes U := \{(x, y) \mid x \in L, y \in U, x \leq y\}$, comprising the consistent elements of the cartesian product between a set L of *lower bounds* and a set U of *upper bounds* (Appendix D in [20]). We can define a new category **LUcons** as the full subcategory of **POSet** with objects the approximation spaces just introduced.

Theorem 1. LUcons *is a Cartesian closed full subcategory of* **CPO**.

By Theorem 1, the category **LUcons** can be taken as approximation category when using the extension of AFT of [7]. Let us now present the approximation system for the language $\mathcal{HOL}$ and the semantics used in [7]. $\mathcal{HOL}$ is based on a type hierarchy $\mathbb{H}$ with base types o, the boolean type, and ι, the type of individuals. The composite types are morphism types obtained from o and ι. In particular, the types are divided into *functional types* $\sigma := \iota \mid \iota \to \sigma$, *predicate types* $\pi := o \mid \rho \to \pi$, and *parameter types* $\rho := \iota \mid \pi$. The semantics of the base types are defined as usual: $E_o := \{\mathbf{t}, \mathbf{f}\}$ with the truth order $\mathbf{f} \leq_t \mathbf{t}$, and $E_\iota = D$ with the trivial order ($d_1 \leq d_2$ iff $d_1 = d_2$), where D is some fixed domain for individuals. The semantics for composite types are defined following the Cartesian closed structure of **POSet**.

The ultimate goal of this application is studying the approximation space of Herbrand interpretations, which fix the value assigned to symbols of functional types. Thus, we only need the approximation spaces for the semantics E_π, for all predicate types π: we can take $S_\mathbb{T}$ to be the smallest subset of Ob(**POSet**) containing E_π for all π, and closed under generalized product.

Now the definition of a suitable approximation system is very straightforward: we just have to define the approximation space $App(E_o)$, the set of exact elements $\mathcal{E}_o$, and the projection $\mathfrak{p}_o$. All the other elements are defined inductively following the Cartesian closed structure of **LUcons**. We define: $App(E_o) := E_o \otimes E_o = \langle\{(\mathbf{t},\mathbf{t}),(\mathbf{f},\mathbf{t}),(\mathbf{f},\mathbf{f})\}, \leq_p\rangle$; $\mathcal{E}_o = \{(\mathbf{t},\mathbf{t}),(\mathbf{f},\mathbf{f})\}$; and $\mathfrak{p}_o(\mathbf{t},\mathbf{t}) := \mathbf{t}$ and $\mathfrak{p}_o(\mathbf{f},\mathbf{f}) := \mathbf{f}$. Finally, given a vocabulary V for $\mathcal{HOL}$ containing symbols of predicate type, the approximation space of Herbrand interpretations is $App\left(\Pi_{s\in V} E_{t(s)}\right) = \Pi_{s\in V} App(E_{t(s)}) \in \text{Ob}(\mathbf{Approx})$, where $t(s)$ is the type of the symbol s.

This greatly simplifies the construction of [7]. In particular, notice that the pairs of *monotonte-antimonotone* and *antimonotone-monotone* functions they defined are *precisely* the elements of the exponential objects of **LUcons**. Moreover, Charalambidis et al. [7] lacked a notion of exactness, making it impossible to determine whether a model is actually two-valued; they discussed this question in their future work section. In our framework, this open question is now resolved, in Definition 2; let us illustrate on their example accompanying the discussion.

Example 2. Let P be a program with the single rule $p(R) \leftarrow R$, where p is of type $o \to o$ and R is a parameter variable of type o. The space of interpretations for p is simply $App(E_{o\to o}) = App(E_o)^{App(E_o)}$, as defined above. By the semantics of [7], the meaning of this program is given by the interpretation (I, J), where $I(p)(\mathbf{t},\mathbf{t}) = J(p)(\mathbf{t},\mathbf{t}) = J(p)(\mathbf{f},\mathbf{t}) = \mathbf{t}$, and $I(p)(\mathbf{f},\mathbf{f}) = J(p)(\mathbf{f},\mathbf{f}) = I(p)(\mathbf{f},\mathbf{t}) = \mathbf{f}$. Since $I \neq J$, (I, J) is not exact according to the classical definition of AFT [12], even though we would expect to find a 2-valued model, i.e., the one assigning to p the identity function over $\{\mathbf{f},\mathbf{t}\}$. Nevertheless, according to our definition, (I, J) is indeed exact: it sends exacts of E_o to exacts of E_o. Furthermore, by the approximation system we defined in this section, it holds that (I, J) represents $\mathfrak{p}_{o\to o}(I, J) = \mathcal{I} \in E_{o\to o} = (E_o \to E_o)$, where $\mathcal{I}(\mathbf{t}) = \mathbf{t}$ and $\mathcal{I}(\mathbf{f}) = \mathbf{f}$, as desired.

5 Conclusion

We introduced a novel theoretical framework that provides a mathematical foundation for using the machinery of AFT on higher-order objects. In particular, we defined *approximation categories* and *approximation systems*: they employ the notion of Cartesian closedness to inductively construct a hierarchy of approximation spaces for each semantics of the types of a given (higher-order) language. This approach solves the issue of applying approximate objects onto approximate objects and ensures that the approximation spaces have the same mathematical structure at any order of the hierarchy, enabling the application of the same AFT techniques at all levels. Moreover, we defined exact elements of a higher-order approximation space, together with a projection function. This is a non-trivial definition and it is fundamental to obtain a sensible AFT framework, i.e., a framework in which we can determine when an object, and in particular a model, is two-valued, and retrieve the elements that are being approximated.

Despite seeming complicated at first, the chosen approach, not only provides a solid, formal mathematical foundation to work with but also allows to reduce proof complexity. The inductive nature and generality of the definition of an approximation system make it extremely easy to adapt the framework to different languages, types, and semantics, as we only have to modify the base elements of the induction. Such generality enables extending different existing versions of AFT while capturing their common underlying characteristics, as we have shown for the extension of consistent AFT of Charalambidis et al. [7].

Acknowledgments. This study was funded by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G0B2221N, and grant V426524N), and by the European Union – NextGenerationEU under the National Recovery and Resilience Plan "Greece 2.0" (H.F.R.I. "Basic research Financing (Horizontal support of all Sciences)", project 16116).

Bibliography

[1] Antic, C.: Fixed point semantics for stream reasoning. Artif. Intell. **288**, 103370 (2020)
[2] Bogaerts, B.: Weighted abstract dialectical frameworks through the lens of approximation fixpoint theory. In: AAAI, pp. 2686–2693, AAAI Press (2019)
[3] Bogaerts, B., Charalambidis, A., Chatziagapis, G., Kostopoulos, B., Pollaci, S., Rondogiannis, P.: The stable model semantics for higher-order logic programming. ICLP 2024 (2024), (to appear)
[4] Bogaerts, B., Cruz-Filipe, L.: Fixpoint semantics for active integrity constraints. Artif. Intell. **255**, 43–70 (2018)
[5] Bogaerts, B., Jakubowski, M.: Fixpoint semantics for recursive SHACL. In: ICLP Technical Communications, EPTCS, vol. 345, pp. 41–47 (2021)
[6] Charalambidis, A., Rondogiannis, P.: Categorical approximation fixpoint theory. In: Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings, Lecture Notes in Computer Science, vol. 14281, pp. 515–530, Springer (2023)

[7] Charalambidis, A., Rondogiannis, P., Symeonidou, I.: Approximation fixpoint theory and the well-founded semantics of higher-order logic programs. Theory Pract. Log. Program. **18**(3-4), 421–437 (2018)
[8] Clark, K.L.: Negation as failure. In: Logic and Data Bases, pp. 293–322, Advances in Data Base Theory, Plemum Press, New York (1977)
[9] Dasseville, I., van der Hallen, M., Bogaerts, B., Janssens, G., Denecker, M.: A compositional typed higher-order logic with definitions. In: ICLP (Technical Communications), OASIcs, vol. 52, pp. 14:1–14:13, Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
[10] Dasseville, I., van der Hallen, M., Janssens, G., Denecker, M.: Semantics of templates in a compositional framework for building logics. Theory Pract. Log. Program. **15**(4-5), 681–695 (2015)
[11] Denecker, M., Bruynooghe, M., Vennekens, J.: Approximation fixpoint theory and the semantics of logic and answers set programs. In: Correct Reasoning, LNCS, vol. 7265, pp. 178–194, Springer (2012)
[12] Denecker, M., Marek, V., Truszczyński, M.: Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In: Logic-Based Artificial Intelligence, vol. 597, pp. 127–144, Springer US (2000)
[13] Denecker, M., Marek, V., Truszczyński, M.: Reiter's default logic is a logic of autoepistemic reasoning and a good one, too. In: Brewka, G., Marek, V., Truszczyński, M. (eds.) Nonmonotonic Reasoning – Essays Celebrating Its 30th Anniversary, pp. 111–144, College Publications (2011), URL http://arxiv.org/abs/1108.3278
[14] Denecker, M., Marek, V.W., Truszczynski, M.: Uniform semantic treatment of default and autoepistemic logics. Artif. Intell. **143**(1), 79–122 (2003)
[15] Denecker, M., Marek, V.W., Truszczynski, M.: Ultimate approximation and its application in nonmonotonic knowledge representation systems. Inf. Comput. **192**(1), 84–121 (2004)
[16] Fitting, M.: A kripke-kleene semantics for logic programs. J. Log. Program. **2**(4), 295–312 (1985)
[17] Fitting, M.: Fixpoint semantics for logic programming a survey. Theor. Comput. Sci. **278**(1-2), 25–51 (2002)
[18] Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, pp. 1070–1080, MIT Press (1988)
[19] Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. Theory Pract. Log. Program. **7**(3), 301–353 (2007)
[20] Pollaci, S., Kostopoulos, B., Denecker, M., Bogaerts, B.: A category-theoretic perspective on higher-order approximation fixpoint theory (extended version) (2024), URL https://arxiv.org/abs/2408.11712
[21] Strass, H.: Approximating operators and semantics for abstract dialectical frameworks. Artif. Intell. **205**, 39–70 (2013)
[22] van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. J. ACM **23**(4), 733–742 (1976)
[23] Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM **38**(3), 620–650 (1991)