TPLP: Page 1–8. © The Author(s), 2021. Published by Cambridge University Press 2021
doi:10.1017/xxxxx

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

POLLACI SAMUELE
Vrije Universiteit Brussel, Belgium and Katholieke Universiteit Leuven, Belgium

KOSTOPOULOS BABIS
Harokopio University of Athens, Greece

DENECKER MARC
Katholieke Universiteit Leuven, Belgium

BOGAERTS BART
Katholieke Universiteit Leuven, Belgium and Vrije Universiteit Brussel, Belgium

submitted xx xx xxxx; revised xx xx xxxx; accepted xx xx xxxx

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 Charalambidis et al. (2018).

KEYWORDS: Approximation fixpoint theory, Higher-order definitions, Category theory.

1 Introduction

Approximation Fixpoint Theory (AFT) (Denecker et al. 2000) 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 (Denecker et al. 2011). Later, it has also been applied to a variety of other domains, including abstract argumentation (Strass 2013; Bogaerts 2019), active integrity constraints (Bogaerts and Cruz-Filipe 2018), stream reasoning (Antic 2020), integrity constraints for the semantic web (Bogaerts and Jakubowski 2021), and Datalog (Pollaci 2025).

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

* 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 Number: 16116).

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 (1976). 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 (Clark 1977), stable fixpoints correspond to (partial) stable models (Gelfond and Lifschitz 1988), the Kripke-Kleene fixpoint to the Kripke-Kleene model (Fitting 1985) and the well-founded fixpoint is the well-founded model (Van Gelder et al. 1991).

This paper is motivated by a need to apply AFT to *higher-order* logic programming that arose in several contexts (Dasseville et al. 2015; 2016; Charalambidis et al. 2018). 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 a brief example (for more detailed ones, we invite the reader to look at Examples 3, and 4). 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 whether $Q(S)$ is true, false, unknown, or inconsistent, for any given set S. 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, there is a need for a richer space of approximations where approximate objects can be applied to other approximate objects.

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 used by Charalambidis et al. (2018) 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 **C** satisfy a property that can be intuitively understood as follows: *if A and B are two objects of* **C***, then the set of morphisms from A to B is also an object of* **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 this 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 (Denecker et al. 2000), consistent AFT (Denecker et al. 2003), or other extensions (Charalambidis et al. 2018)). To formalize this, and unify different AFT accounts, 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 recent work, a stable semantics for higher-order logic programs was defined building on consistent AFT (Bogaerts et al. 2024). 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* (Denecker et al. 2004) is used; this has also been done in other extensions of

logic programming (Pelov et al. 2007; Dasseville et al. 2016). Bogaerts et al. (2024) also pointed out a rather counterintuitive behaviour of the well-founded semantics defined in the work of Charalambidis et al. (2018), namely that even for simple non-recursive programs, the well-founded model might not assume expected values (leaving all atoms unknown). It is important to mention, though, that this counterintuitive behaviour 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 Charalambidis and Rondogiannis (2023) 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 (2023) view the approximation spaces themselves as categories to provide a novel version of standard AFT.

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

1. We generalize the work of Charalambidis et al. (2018) to a category-theoretic setting. In doing so, we shed light on the general principles underlying their constructions for higher-order logic programing and make their construction applicable to arbitrary current and future non-monotonic reasoning formalism captured by AFT.
2. We improve the work of Charalambidis et al. (2018). In particular, we define a new approximator, which provides the expected well-founded semantics; and we study the concept of *exactness*, previously missing, allowing the use of the theory to define exact *stable models* instead of focusing purely on the well-founded model.

It is also worth remarking that the generality of the CT environment allows to cover various accounts of AFT, like the ones of Denecker et al. (2000), Denecker et al. (2003), and Charalambidis et al. (2018), and possibly others. Different versions of AFT are suitable for various situations and cater to specific applications. For instance, consistent AFT (Denecker et al. 2003) utilizes a three-valued logic and provides a rather intuitive and easily applicable notion of approximation. On the other hand, standard AFT (Denecker et al. 2000) employs a four-valued approach, with *inconsistent* elements, as presented earlier in this introduction. This can sometimes be of a more difficult use in applications, but shows several advantages from the formal, mathematical standpoint: having a full bilattice, composed of both consistent and inconsistent elements, provides symmetry and allows for duality results to be derived, simplifying the proofs of the fundamental theorems at the core of this version of AFT. It is hence valuable to obtain a framework that covers as many accounts of AFT as possible.

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 square bilattices form a Cartesian closed category that can be chosen as **Approx** for standard AFT. With a suitable choice of *App* and exact elements, depending on the application at hand, we obtain an approximation system that recovers the framework of standard AFT and extends it to higher-order objects. This section can be skipped by the reader interested uniquely in the AFT version of Charalambidis et al. (2018), which is addressed in the following section. In Section 5, we apply the novel categorical framework to Charalambidis et al. (2018). First, in Subsection 5.1, we show that the approximation spaces from Charalambidis

et al. (2018) form a Cartesian closed category. Second, in Subsection 5.2 we define an approximation system that enables us to reconstruct in a simple way (using the general principles outlined above) the semantic elements defined ad-hoc by Charalambidis et al. (2018). At the same time, the definition of such approximation system 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, this allows to determine when a model of a program is two-valued (see Example 2). We proceed with Subsection 5.3, where we present the new approximator that adjusts the behaviour of the well-founded semantics of Charalambidis et al. (2018) for programs with existential quantifiers in the body of rules. We close the subsection with two examples of logic programs in which we need to apply an approximate object on another approximate object. We conclude in Section 6.

2 Preliminaries

In this section, we provide a concise introduction to the formal concepts we utilize throughout the paper. We divide the content into two subsections. In the former (Section 2.1), we outline the core ideas at the foundation of AFT, and we present in more detail the parts of the work of Charalambidis et al. (2018) that we aim to modify in Section 5. In the second subsection (Section 2.2), we present the notions of Category Theory (CT) we need, with the definition of Cartesian closed category being the key concept. For further information on CT, we refer to the book by Riehl (2017).

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, when referring to a poset $\mathcal{P}$, we often use the notation for the underlying set P in place of the calligraphic one. 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 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 J such that for any subset $S \subseteq J$, $\bigsqcup S$ exists. A *complete lattice* is a poset L such that for any subset $S \subseteq L$, both $\bigsqcap S$ and $\bigsqcup S$ exist. A function $f \colon P_1 \to P_2$ between posets is *monotone* if for all $x, y \in P_1$ such that $x \leq_{P_1} y$, it holds that $f(x) \leq_{P_2} f(y)$. We refer to functions $O \colon C \to C$ with domain equal to the codomain as *operators*. An element $x \in C$ 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 (Denecker et al. 2000) 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, which is 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 (Fitting 2002), then the aforementioned four types of fixpoint correspond to the homonymous semantics of logic programing (Denecker et al. 2000; 2012).

The concepts presented so far are part of what we refer to as *standard AFT* (Denecker et al. 2000), i.e. the first account of AFT. Following this initial take, several other variants have been developed: consistent AFT (Denecker et al. 2003), non-deterministic AFT (Heyninck et al. 2024), or other extensions (Charalambidis et al. 2018). In particular, the latter already proposes a way to deal with higher-order logic programs via an extension of consistent AFT. However, as already highlighted by Bogaerts et al. (2024), the work of Charalambidis et al. (2018) had some hidden problematic features. They can be summarised as follows:

1. *The Approximator:* the well-founded semantics obtained via the approximator defined by Charalambidis et al. (2018) does not behave as expected when an existential quantifier occurs in the body of a rule. Take for instance the logic program with just the simple rule $\mathsf{p} \leftarrow \mathsf{R} \wedge {\sim}\,\mathsf{R}$, where R is a variable ranging over the booleans $\{\mathbf{f}, \mathbf{t}\}$. If we naively ground such program, we obtain $\mathsf{p} \leftarrow \mathbf{f} \wedge \mathbf{t}$ and $\mathsf{p} \leftarrow \mathbf{t} \wedge \mathbf{f}$, and p would clearly be evaluated as false. However, the approach adopted by Charalambidis et al. (2018) uses *approximated* elements. In particular, variables of type boolean range over $\{\mathbf{f}, \mathbf{t}, \mathbf{u}\}$. In more detail, for the logic program $\mathsf{p} \leftarrow \mathsf{R} \wedge {\sim}\,\mathsf{R}$, the approximator assignes to p the least upper bound of the body $\mathsf{R} \wedge {\sim}\,\mathsf{R}$, with R ranging over $\{\mathbf{f}, \mathbf{t}, \mathbf{u}\}$. Since such least upper bound is computed with respect to the *truth order* $\mathbf{f} \leq \mathbf{u} \leq \mathbf{t}$, the predicate p is assigned the value $\bigsqcup\{\mathbf{t} \wedge \mathbf{f}, \mathbf{f} \wedge \mathbf{t}, \mathbf{u} \wedge \mathbf{u}\} = \mathbf{u}$ under the well-founded semantics. This contradicts the more intuitive and standard two-valued approach via grounding, which assigns $\mathbf{f}$ to p. In a way, allowing the existentially quantified variable to vary over all approximated elements seems to unnecessarily increase (w.r.t. the truth order) the value of the defined predicate, when evaluated under the well-founded semantics.
2. *The Notion of Exactness:* the work of Charalambidis et al. (2018) lacks the notion of exactness for higher-order objects, which is fundamental in the context of AFT and rather non-trivial in the higher-order setting: exactness allows to recognize

whether an approximated object, i.e. a pair (x, y) in the bilattice, represents just one *real* element of the lattice, and, in particular, when a model is two-valued. In other words, having such concept makes it possible to study not just the well-funded models, but also the stable ones.

In Section 5, we will show how we can use our novel concepts in the framework of Charalambidis et al. (2018) to solve the issues listed above.

2.2 Category Theory

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.

Definition 1
A *category* $\mathbf{C}$ consists of

- a collection of *objects* $\mathrm{Ob}(\mathbf{C})$,
- a collection of *morphisms* $\mathrm{Mor}(\mathbf{C})$,
- for every morphism $f \in \mathrm{Mor}(\mathbf{C})$, an object $s(f)$ called the *source* (or *domain*) of f, and an object $t(f)$ called the *target* (or *codomain*) of f,
- for every object $X \in \mathrm{Ob}(\mathbf{C})$, a morphism id_X called the *identity morphism*,
- for every two morphisms $f, g \in \mathrm{Mor}(\mathbf{C})$ with $t(f) = s(g)$, a morphism $g \circ f$, called their *composite*,

such that

- for all $f, g \in \mathrm{Mor}(\mathbf{C})$ such that $t(f) = s(g)$, $s(g \circ f) = s(f)$
- for all $f, g \in \mathrm{Mor}(\mathbf{C})$ such that $t(f) = s(g)$, $t(g \circ f) = t(g)$,
- for all $X \in \mathrm{Ob}(\mathbf{C})$, $s(id_X) = t(id_X) = X$,
- for all $f, g, h \in \mathrm{Mor}(\mathbf{C})$ such that $t(f) = s(g)$ and $t(g) = s(h)$, $(h \circ g) \circ f = h \circ (g \circ f)$,
- for all $X, Y \in \mathrm{Ob}(\mathbf{C})$ and for all $f \in \mathrm{Mor}(\mathbf{C})$ such that $s(f) = X$ and $t(f) = Y$, $f \circ id_X = f$ and $id_Y \circ f = f$.

In this paper, objects will always be certain ordered sets, and morphisms will be monotone functions. In the same way as morphisms between objects encode relations within a category, a morphism of categories, called a *functor*, describes the relation between two categories.

Definition 2
Let $\mathbf{C}, \mathbf{D}$ be two categories. A *functor* $F\colon \mathbf{C} \to \mathbf{D}$ consists of a function $F_0\colon \mathrm{Ob}(\mathbf{C}) \to \mathrm{Ob}(\mathbf{D})$ between the classes of objects, and a function $F_1\colon \mathrm{Mor}(\mathbf{C}) \to \mathrm{Mor}(\mathbf{D})$, such that it respects target and source of morphisms, identity morphisms, and composition.

For each $x, y \in \mathrm{Ob}(\mathbf{C})$, we denote by $\hom_{\mathbf{C}}(x, y)$ the set of morphisms of $\mathbf{C}$ with domain x and codomain y.

Definition 3
A functor $F\colon \mathbf{C} \to \mathbf{D}$ is

- *full* if for each $X, Y \in \mathrm{Ob}(\mathbf{C})$, the map $F_1 \restriction_{\hom_{\mathbf{C}}(X,Y)} \colon \hom_{\mathbf{C}}(X,Y) \to \hom_{\mathbf{D}}(F_0(X), F_0(Y))$ is surjective,
- *faithful* if for each $X, Y \in \mathrm{Ob}(\mathbf{C})$, the map $F_1 \restriction_{\hom_{\mathbf{C}}(X,Y)} \colon \hom_{\mathbf{C}}(X,Y) \to \hom_{\mathbf{D}}(F_0(X), F_0(Y))$ is injective,
- *embedding* if F is faithful and F_0 is injective.

The domain of a full embedding $F \colon \mathbf{C} \to \mathbf{D}$ is called a *full subcategory* of the codomain (denoted as $\mathbf{C} \subseteq \mathbf{D}$).

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. Clearly, it also holds that $\mathbf{CLat} \subseteq \mathbf{CPO} \subseteq \mathbf{POSet}$ and $\mathbf{CLat} \subseteq \mathbf{CJSLat} \subseteq \mathbf{POSet}$.

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*.

Definition 4
$T \in \mathrm{Ob}(\mathbf{C})$ 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**.

Definition 5
Let $A_1, A_2 \in \mathrm{Ob}(\mathbf{C})$. A *product* of A_1 and A_2 is an object of $\mathbf{C}$, denoted by $A_1 \times A_2$, equipped with two morphisms $\pi_1 \colon A_1 \times A_2 \to A_1$ and $\pi_2 \colon A_1 \times A_2 \to A_2$, called *first*, and *second projection* respectively, such that for any $B \in \mathrm{Ob}(\mathbf{C})$ and any morphisms $f_1 \colon B \to A_1$ and $f_2 \colon B \to A_2$, there exists a unique morphism $f_1 \times f_2 \colon B \to A_1 \times A_2$ such that $\pi_1 \circ (f_1 \times f_2) = f_1$ and $\pi_2 \circ (f_1 \times f_2) = f_2$.

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.

Definition 6
Let $A_1, A_2 \in \mathrm{Ob}(\mathbf{C})$. An *exponential* of A_1 and A_2 is an object of $\mathbf{C}$, denoted by $A_2^{A_1}$, equipped with a morphism $ev \colon A_2^{A_1} \times A_1 \to A_2$, called the *evaluation*, such that for any $B \in \mathrm{Ob}(\mathbf{C})$ and any morphism $f \colon B \times A_1 \to A_2$, there exists a unique morphism $f' \colon B \to A_2^{A_1}$ such that $ev \circ (f' \times id) = f$.

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., $ev(f, x) = f(x)$.

Definition 7
A category $\mathbf{C}$ is *Cartesian closed* if it has a terminal object, and for each $A_1, A_2 \in \mathrm{Ob}(\mathbf{C})$, there exist $A_1 \times A_2 \in \mathrm{Ob}(\mathbf{C})$ and $A_2^{A_1} \in \mathrm{Ob}(\mathbf{C})$.

By our prior observations, 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 of a logic program. In order to include this, we need another notion from CT, namely a generalized version of the categorical product for (possibly infinite) families of objects.

Definition 8
Let $\{A_i\}_{i\in I}$ be a family of objects of a category $\mathbf{C}$ indexed by I. The *(generalized) product* of the family $\{A_i\}_{i\in I}$ is an object of $\mathbf{C}$, denoted by $\Pi_{i\in I}A_i$, equipped with morphisms $\pi_i\colon \Pi_{i\in I}A_i \to A_i$, called the *i-th projection*, such that for all $B \in \mathrm{Ob}(\mathbf{C})$ and for all families of morphisms $\{\varphi_i\colon B \to A_i\}_{i\in I}$ indexed by I, there exists a unique $\psi\colon B \to \Pi_{i\in I}A_i$ such that for all $i \in I$ it holds that $\varphi_i = \pi_i \circ \psi$.

We say that a category $\mathbf{C}$ *has generalized products* if 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. Clearly, if $\mathbf{C}$ is Cartesian closed and the index I is finite, this product always exists; but this is not always the case for infinite I. However, we will show in the remainder of this subsection that every full subcategory of **POSet** has generalized products (Proposition 2). We wish to warn the reader that the following category-theoretic notions are only meant to support the proofs of Propositions 2 and 3, and will not be used in the next sections of the paper.

The generalized product is a special case of a very common construction in CT, called the *limiting cone*, or simply *limit*. We proceed with the definitions leading to the concept of limit.

Definition 9
A *diagram* $X_\bullet$ in a category $\mathbf{C}$ is

1. a set $\{X_i\}_{i\in I}$ of objects of $\mathbf{C}$,
2. for every pair $(i,j) \in I \times I$, a set $\{f_\alpha\colon X_i \to X_j\}_{\alpha\in I_{i,j}}$ of morphisms,
3. for every $i \in I$ an element $\epsilon_i \in I_{i,i}$,
4. for each $(i,j,k) \in I \times I \times I$ a function $comp_{i,j,k}\colon I_{i,j} \times I_{j,k} \to I_{i,k}$ such that
   (a) *comp* is associative and unital with the f_{ϵ_i}'s being the neutral elements,
   (b) for every $i \in I$, $f_{\epsilon_i} = id_{X_i}$ is the identity morphism of X_i,
   (c) for every two composable morphisms $f_\alpha\colon X_i \to X_j$ and $f_\beta\colon X_j \to X_k$, it holds that $f_\beta \circ f_\alpha = f_{comp_{i,j,k}(\alpha,\beta)}$.

Definition 10
Let $X_\bullet = (\{f_\alpha\colon X_i \to X_j\}_{i,j\in I,\alpha\in I_{i,j}}, comp)$ be a diagram in $\mathbf{C}$. A *cone* over $X_\bullet$ is an object $X \in \mathrm{Ob}(\mathbf{C})$ together with, for each $i \in I$, a morphism $p_i\colon X \to X_i \in \mathrm{Mor}(\mathbf{C})$ such that for all $(i,j) \in I \times I$ and for all $\alpha \in I_{i,j}$, it holds that $f_\alpha \circ p_i = p_j$.
Moreover, the *limiting cone* or *limit* of $X_\bullet$ is, if it exists, the cone over $X_\bullet$ which is *universal* among all possible cones over $X_\bullet$.

Definition 11
A functor $F\colon \mathbf{C} \to \mathbf{D}$ *reflects all limits* if for all diagrams $X_\bullet$ in $\mathbf{C}$, and for all cones C over $X_\bullet$ such that $F(C)$ is a limiting cone of $F(X_\bullet)$, C is a limiting cone of $X_\bullet$.

Proposition 1
A full and faithful functor reflects all limits.

Proof
Lemma 3.3.5 in (Riehl 2017). □

We are finally able to prove that every full subcategory of **POSet** has geeralized products.

Proposition 2
If $\mathbf{C} \subseteq \mathbf{POSet}$, then $\mathbf{C}$ has generalized products.

Proof
By Definition 3, there exists a full embedding $F\colon \mathbf{C} \to \mathbf{POSet}$. By Proposition 1, F reflects all limits. Moreover, it is clear that **POSet** has generalized products. Since generalized products are a type of limit, we conclude that $\mathbf{C}$ has generalized products. □

In a full subcategory of **POSet** we can rewrite a generalized product as a poset of functions. Given two posets $\mathcal{P}_1, \mathcal{P}_2 \in \mathrm{Ob}(\mathbf{POSet})$, 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 3
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 $\mathbf{C}$.

Proof
By Proposition 2, $\Pi_{x \in X} Y \in \mathrm{Ob}(\mathbf{C})$. Moreover, there exists a bijection $\varphi\colon (X \to Y) \to \Pi_{x \in X} Y$, defined by $\varphi(f) = (f(a))_{a \in X}$, with inverse $\varphi^{-1}\colon \Pi_{x \in X} Y \to (X \to Y)$, defined by, for all $a \in X$, $\varphi^{-1}((y_x)_{x \in X})(a) = y_a$. Because of the definition of pointwise order and product order, it is immediate to show that both φ and φ^{-1} preserve the orders, i.e., they are monotone. □

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 5 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 12) 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 2). 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 12). 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 12). Lastly, we add one more condition on exact elements to accommodate several existing versions of AFT. In consistent AFT (Denecker et al. 2003), 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 12). 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 4 and 5).

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., $f^{-1}(b) = \{a \mid f(a) = b\} \subseteq A$. Recall that given two posets $\mathcal{P}_1, \mathcal{P}_2 \in \mathrm{Ob}(\mathbf{POSet})$, we denote by $(\mathcal{P}_1 \to \mathcal{P}_2) \in \mathrm{Ob}(\mathbf{POSet})$ the poset of (possibly non-monotone) functions from $\mathcal{P}_1$ to $\mathcal{P}_2$ ordered with the pointwise order.

Definition 12
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 $App(E_{\tau_1\to\tau_2}) = (E_{\tau_1} \to App(E_{\tau_2}))$,
   (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 $\mathfrak{p}^0_\tau(\bigsqcap(\mathfrak{p}^0_\tau)^{-1}(e)) = e$.

Notice that, by Proposition 3, the object $(E_{\tau_1} \to App(E_{\tau_2}))$ in Item 2b of Definition 12 is indeed an object of the approximation category **Approx**. Morover, again by Proposition 3, 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.

Finally, 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 13
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 for all $e_1 \in E_{\tau_1}$, $e(e_1) \in App(E_{\tau_2})$ 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.

The reader may wonder why there are two cases for a morphism type $\tau_1 \to \tau_2$ in Definition 13, depending whether E_{τ_1} is in $S_\mathbb{T}$ or not, i.e. whether we approximate the elements of the semantics of τ_1 or not. Recall that an exact element in the approximation space is meant to represent one and only one element of the semantics of the same type. Intuitively, for a morphism type $\tau_1 \to \tau_2$, a function in $App(E_{\tau_1\to\tau_2})$ is exact when the image of any exact element is exact. This is indeed sufficient and we do not need to consider the image of non-exact elements of the domain: we will prove in Proposition 5 that there is an exact element in the approximation space for each element of the semantics of the corresponding type. Now, considering *exact elements of the domain* (and looking at their image) makes sense only when the domain is an approximation space, i.e. when $E_{\tau_1} \in S_\mathbb{T}$; in the other case, when $E_{\tau_1} \notin S_\mathbb{T}$ we can directly consider the elements of the semantics, as they do not get approximated.

For $\tau \in \mathbb{T}$, we denote by $\mathcal{E}_\tau$ the subset of $App(E_\tau)$ of exact elements of type τ. The following proposition shows that the condition 3b of Definition 12 holds for any $E_\tau \in S_\mathbb{T}$.

Proposition 4
Either for all $E_\tau \in S_\mathbb{T}$ it holds that $App(E_\tau) \in \mathrm{Ob}(\mathbf{CJSLat})$, or for all $E_\tau \in S_\mathbb{T}$, for all $b \in App(E_\tau)$, and for all $e \in \mathcal{E}_\tau$, if $e \leq_{App(E_\tau)} b$, then $b \in \mathcal{E}_\tau$.

Proof
Suppose there exists $E_{\tau'} \in S_\mathbb{T}$ such that $App(E_{\tau'}) \notin \mathrm{Ob}(\mathbf{CJSLat})$. We have to show that for all $E_\tau \in S_\mathbb{T}$, for all $b \in App(E_\tau)$, and for all $e \in \mathcal{E}_\tau$, if $e \leq_{App(E_\tau)} b$, then $b \in \mathcal{E}_\tau$. We proceed by induction on τ.

Let τ be a base type. Since we assumed that $App(E_{\tau'}) \notin \mathrm{Ob}(\mathbf{CJSLat})$ for some $E_{\tau'} \in S_\mathbb{T}$, and **CJSLat** is Cartesian closed, and has generalized products by Proposition 2, then there must exists a $\sigma \in \mathcal{B}$ such that $App(E_\sigma) \notin \mathrm{Ob}(\mathbf{CJSLat})$. Thus, by condition 3b of Definition 12 we can conclude the base step of the induction.

Now let $\tau = \Pi_{i\in I}\tau_i$ and suppose the proposition hold for E_{τ_i} for all $i \in I$. Let $(b_i) \in App(E_\tau)$ such that $e := (e_i) \leq_{App(E_\tau)} (b_i)$. By the definition of the product order, Definition 13, and the induction hypothesis, we get that $b_i \in \mathcal{E}_{\tau_i}$ for all $i \in I$, i.e., $(b_i) \in \mathcal{E}_\tau$, as desired.

Let $\tau = \tau_1 \to \tau_2$ with $E_{\tau_1} \notin S_\mathbb{T}$, and suppose the proposition hold for E_{τ_2}. By Proposition 3, it holds that $App(E_\tau) \cong App(\Pi_{i\in E_{\tau_1}} E_{\tau_2})$, thus, we can reduce to the previous case.

Let $\tau = \tau_1 \to \tau_2$ with $E_{\tau_1} \in S_\mathbb{T}$, and suppose the proposition hold for E_{τ_1} and E_{τ_2}. Let $f \in App(E_\tau)$ such that $e \leq_{App(E_\tau)} f$. For f to be exact, it must send exact elements to exact elements. Let $a \in \mathcal{E}_{\tau_1}$. By the definition of the order on morphisms, and Defintion

13, it holds that $f(a) \geq_{App(E_{\tau_2})} e(a) \in \mathcal{E}_{\tau_2}$. By induction hypothesis, it follows that $f(a) \in \mathcal{E}_{\tau_2}$. Hence, $f \in \mathcal{E}_\tau$, as desired. □

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 $f \in \mathcal{E}_\tau$, and for all $e \in E_{\tau_1}$, $\mathfrak{p}_\tau(f)(e) := \mathfrak{p}_{\tau_2}(f(e))$.
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 $d \in \mathfrak{p}^{-1}_{\tau_1}(e)$, i.e., $\mathfrak{p}_{\tau_1}(d) = e$.

Recall that, intuitively, the function $\mathfrak{p}_\tau$ sends an exact element of type τ to the element it represents in the semantics of τ.

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 12.

Proposition 5
Let $E_\tau \in S_\mathbb{T}$, $e_1, e_2 \in \mathcal{E}_\tau$, and $e \in E_\tau$. The following statements hold:

1. $\mathfrak{p}_\tau$ is well-defined.
2. $\mathfrak{p}_\tau$ is surjective.
3. if $e_1 \leq_{App(E_\tau)} e_2$, then $\mathfrak{p}_\tau(e_1) = \mathfrak{p}_\tau(e_2)$.
4. there exists $\bigsqcap \mathfrak{p}^{-1}_\tau(e) \in \mathcal{E}_\tau$ and $\mathfrak{p}_\tau(\bigsqcap \mathfrak{p}^{-1}_\tau(e)) = e$.

Proof
We proceed by induction on τ. Let $\tau \in \mathcal{B}_\mathbb{T}$. Then $\mathfrak{p}_\tau = \mathfrak{p}^0_\tau$ and the Items 1, 2, 3, and 4 hold by definition of $\mathfrak{p}^0_\tau$.

Now suppose $\tau = \Pi_{i\in I}\tau_i$, and assume that Items 1, 2, 3, and 4 hold for τ_i for all $i \in I$. Since $\mathfrak{p}_{\tau_i}$ is well-defined and surjective by hypothesis for all $i \in I$, it is clear by definition that also $\mathfrak{p}_\tau$ is well-defined and surjective. Let $(a_i), (b_i) \in \mathcal{E}_\tau$ such that $(a_i) \leq_{App(E_\tau)} (b_i)$. By the product order on $App(E_\tau)$ we have $a_i \leq_{App(E_{\tau_i})} b_i$ for all $i \in I$. By definition 13, $a_i, b_i \in \mathcal{E}_{\tau_i}$ for all $i \in I$. Hence, by hypothesis it follows that $\mathfrak{p}_{\tau_i}(a_i) = \mathfrak{p}_{\tau_i}(b_i)$ for all $i \in I$. By definition of $\mathfrak{p}_\tau$, we get that $\mathfrak{p}_\tau((a_i)) = \mathfrak{p}_\tau((b_i))$. Thus, Item 3 hold. Now let $(a_i) \in E_\tau$. By the definition of $\mathfrak{p}_\tau$, it is easy to see that $\mathfrak{p}^{-1}_\tau((a_i)) = \Pi_{i\in I}\mathfrak{p}^{-1}_{\tau_i}(a_i)$. Hence, by induction hypothesis for Item 4, we have that $\bigsqcap(\mathfrak{p}^{-1}_\tau((a_i))) = \bigsqcap(\Pi_{i\in I}\mathfrak{p}^{-1}_{\tau_i}(a_i)) = \Pi_{i\in I}\bigsqcap\mathfrak{p}^{-1}_{\tau_i}(a_i) \in \Pi_{i\in I}\mathfrak{p}^{-1}_{\tau_i}(a_i) = \mathfrak{p}^{-1}_\tau((a_i))$, where the second equality holds because of the definition of the product order on $App(E_\tau)$. Thus, also Item 4 hold for $\mathfrak{p}_\tau$.

Suppose $\tau = \tau_1 \to \tau_2$ with $E_{\tau_1} \notin S_\mathbb{T}$, and assume that Items 1, 2, 3, and 4 hold for and τ_2. By Proposition 3 and Definition 12, we have $E_{\tau_1\to\tau_2} = (E_{\tau_1} \to E_{\tau_2}) \cong \Pi_{i\in E_{\tau_1}} E_{\tau_2}$ and $App(E_{\tau_1\to\tau_2}) \cong App(\Pi_{i\in E_{\tau_1}} E_{\tau_2})$. It is easy to see that we can reduce to the previous case with $I := E_{\tau_1}$.

Finally, suppose $\tau = \tau_1 \to \tau_2$ with $E_{\tau_1} \in S_\mathbb{T}$, and assume that Items 1, 2, 3, and 4 hold for τ_1 and τ_2. We first show that $\mathfrak{p}_\tau$ is well-defined, i.e., for all $f \in \mathcal{E}_\tau$ there exists unique $g \in E_\tau = E_{\tau_2}^{E_{\tau_1}}$ such that $\mathfrak{p}_\tau(f) = g$. Let $f \in \mathcal{E}_\tau$. First notice that,

since $\mathfrak{p}_{\tau_1}$ is surjective by hypothesis, for all $e \in E_{\tau_1}$ there exists $d \in \mathcal{E}_{\tau_1}$ such that $\mathfrak{p}_{\tau_1}(d) = e$. Moreover, since $\mathfrak{p}_{\tau_2}$ is well-defined by hypothesis, for all $e \in E_{\tau_1}$ and $d \in \mathfrak{p}_{\tau_1}^{-1}(e)$, we get an element $\mathfrak{p}_{\tau_2}(f(d)) \in E_{\tau_2}$. It remains to show that for all $e \in E_{\tau_1}$, if $d_1, d_2 \in \mathfrak{p}_{\tau_1}^{-1}(e) \subseteq \mathcal{E}_{\tau_1}$, then $\mathfrak{p}_{\tau_2}(f(d_1)) = \mathfrak{p}_{\tau_2}(f(d_2))$. Let $e \in E_{\tau_1}$ and $d_1, d_2 \in \mathfrak{p}_{\tau_1}^{-1}(e)$. By induction hypothesis for Item 4, there exists $d_3 \in \mathfrak{p}_{\tau_1}^{-1}(e)$ such that $d_3 \leq_{App(E_{\tau_1})} d_1$, $d_3 \leq_{App(E_{\tau_1})} d_2$. Since $f \in \mathcal{E}_\tau = \mathcal{E}_{\tau_2}^{\mathcal{E}_{\tau_1}}$ is a morphism of cpo's, it is monotone. Hence, it holds that $f(d_3) \leq_{App(E_{\tau_2})} f(d_1)$, $f(d_3) \leq_{App(E_{\tau_2})} f(d_2)$. By induction hypothesis for Item 3, it follows that $\mathfrak{p}_{\tau_2}(f(d_1)) = \mathfrak{p}_{\tau_2}(f(d_2))$. Thus, $\mathfrak{p}_\tau$ is well-defined.

We now show that $\mathfrak{p}_\tau$ is surjective. Let $g \in E_\tau = E_{\tau_2}^{E_{\tau_1}}$. By the induction hypothesis on τ_2 for Item 4, for each $e \in E_{\tau_1}$, we can define an element $d_e := \bigsqcap \mathfrak{p}_{\tau_2}^{-1}(g(e)) \in \mathfrak{p}_{\tau_2}^{-1}(g(e))$. By Proposition 4, for each $a \in App(E_{\tau_1}) \setminus \mathcal{E}_{\tau_1}$ such that there exists (at least one) $b \in \mathcal{E}_{\tau_1}$ with $b \leq_{App(E_{\tau_1})} a$, we can define

$$c_a := \bigsqcup \{ d_{\mathfrak{p}_{\tau_1}(b)} \mid b \in \mathcal{E}_{\tau_1} \text{ such that } b \leq_{App(E_{\tau_1})} a \}.$$

We define $f \colon App(E_{\tau_1}) \to App(E_{\tau_2})$ for all $a \in App(E_{\tau_1})$ as follows:

$$f(a) := \begin{cases} d_{\mathfrak{p}_{\tau_1}(a)} & \text{if } a \in \mathcal{E}_{\tau_1}, \\ c_a & \text{if } a \notin \mathcal{E}_{\tau_1} \text{ and exists } b \in \mathcal{E}_{\tau_1} \text{ such that } b \leq_{App(E_{\tau_1})} a, \\ \bot_{App(E_{\tau_2})} & \text{otherwise.} \end{cases} \tag{1}$$

In the following we show that f is monotone. Let $a_1, a_2 \in App(E_{\tau_1})$ such that $a_1 \leq_{App(E_{\tau_1})} a_2$. If $a_1 \notin \mathcal{E}_{\tau_1}$ and for all $b \in \mathcal{E}_{\tau_1}$ is not the case that $b \leq_{App(E_{\tau_1})} a$, then clearly $f(a_1) = \bot_{App(E_{\tau_2})} \leq_{App(E_{\tau_2})} f(a_2)$. If $a_1, a_2 \in \mathcal{E}_{\tau_1}$, then $f(a_1) = f(a_2)$ by the induction hypothesis for Item 3. If a_1 is exact but a_2 is not, then clearly $f(a_1) \leq_{App(E_{\tau_2})} f(a_2)$. If both $a_1, a_2 \notin \mathcal{E}_{\tau_1}$ and they are greater than some exact $b \in \mathcal{E}_{\tau_1}$, then $\{b \in \mathcal{E}_{\tau_1} \mid b \leq_{App(E_{\tau_1})} a_1\} \subseteq \{b \in \mathcal{E}_{\tau_1} \mid b \leq_{App(E_{\tau_1})} a_2\}$. Hence, $f(a_1) = c_{a_1} \leq_{App(E_{\tau_2})} c_{a_2} = f(a_2)$, as desired. It follows that $f \in App(E_\tau)$. Moreover, it is clear that f sends exact elements to exact elements, i.e., $f \in \mathcal{E}_\tau$. For all $e \in E_{\tau_1}$, $\mathfrak{p}_\tau(f)(e) = \mathfrak{p}_{\tau_2}(f(c)) = \mathfrak{p}_{\tau_2}(d_{\mathfrak{p}_{\tau_1}(c)}) = \mathfrak{p}_{\tau_2}(d_e) = g(e)$, where c is some element in the preimage $\mathfrak{p}_{\tau_1}^{-1}(e)$. Thus, $\mathfrak{p}_\tau(f) = g$, as desired.

We proceed to show that Item 3 holds for τ. Let $f_1, f_2 \in \mathcal{E}_\tau$ such that $f_1 \leq_{App(E_\tau)} f_2$, and let $e \in E_{\tau_1}$. We have already shown in Item 1 for τ that $\mathfrak{p}_\tau$ is well defined. In particular, $\mathfrak{p}_\tau(g)(e) = \mathfrak{p}_{\tau_2}(g(d_1)) = \mathfrak{p}_{\tau_2}(g(d_2))$ for all $g \in \mathcal{E}_\tau$ and $d_1, d_2 \in \mathfrak{p}_{\tau_1}^{-1}(e)$. By the definition of the order on morphisms, $f_1(d) \leq_{App(E_{\tau_2})} f_2(d)$ for all $d \in \mathcal{E}_{\tau_1}$. By the induction hypothesis for Item 3, it holds that $\mathfrak{p}_\tau(f_1)(e) = \mathfrak{p}_{\tau_2}(f_1(d)) = \mathfrak{p}_{\tau_2}(f_2(d)) = \mathfrak{p}_\tau(f_2)(e)$. Hence, $\mathfrak{p}_\tau(f_1) = \mathfrak{p}_\tau(f_2)$, as desired.

Finally, we show that Item 4 holds for τ. Let $g \in E_\tau$. We can construct a morphism $f \in \mathcal{E}_\tau$ using the same technique as in (1). By the proof of Item 2, we already have $\mathfrak{p}_\tau(f) = g$. It remains to show that $f = \bigsqcap \mathfrak{p}_\tau^{-1}(g)$. Let $h \in \mathfrak{p}_\tau^{-1}(g)$. First notice that since $\mathfrak{p}_\tau(h) = \mathfrak{p}_\tau(f) = g$, it holds that for all $e \in E_{\tau_1}$, $\mathfrak{p}_{\tau_2}(f(l)) = \mathfrak{p}_{\tau_2}(h(l)) = g(e)$ for all $l \in \mathfrak{p}_{\tau_1}^{-1}(e)$. In particular, for all $a \in \mathcal{E}_{\tau_1}$, it holds that $f(a), h(a) \in \mathfrak{p}_{\tau_2}^{-1}(g(\mathfrak{p}_{\tau_1}(a)))$. Hence, for all $a \in \mathcal{E}_{\tau_1}$, we have that $f(a) = d_{\mathfrak{p}_{\tau_1}(a)} = \bigsqcap(\mathfrak{p}_{\tau_2}^{-1}(g(\mathfrak{p}_{\tau_1}(a)))) \leq_{App(E_{\tau_2})} h(a)$. Now let $a \notin \mathcal{E}_{\tau_1}$ such that there exists $b \in \mathcal{E}_{\tau_1}$ such that $b \leq_{App(E_{\tau_1})} a$. Since we have already shown that $f(c) = d_{\mathfrak{p}_{\tau_1}(c)} \leq_{App(E_{\tau_2})} h(c)$ for all $c \in \mathcal{E}_{\tau_1}$, it is easy to see that $f(a) = c_a \leq_{App(E_{\tau_2})} h(a)$. For all the other cases of $a \in App(E_{\tau_1})$ it is obvious that $f(a) \leq_{App(E_{\tau_2})} h(a)$. Hence, f is a lower bound of $\mathfrak{p}_\tau^{-1}(g)$. Since $f \in \mathfrak{p}_\tau^{-1}(g)$, we get $f = \bigsqcap \mathfrak{p}_\tau^{-1}(g)$, as desired. $\square$

In most applications of AFT, for approximation spaces of base types, there exists a unique exact element representing an object of a semantics, and Items 3 and 4 of Proposition 5 are trivially verified. 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 $h(\mathbf{f}) = h(\mathbf{t}) = \mathbf{t}$.

We conclude this section with the definition of *consistent* elements.

Definition 14
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 standard AFT

In this section, we show how our new framework extends the standard AFT setting to higher-order definitions.

The main building block of an approximation system is the category **Approx**, containing all the desired approximation spaces. Hence, we start by showing that the approximation spaces used in standard AFT, i.e. the square bilattices, form a Cartesian closed category.

First, recall that a *square bilattice* is a poset of the form $\langle L \times L, \leq_p\rangle$, where $\langle L, \leq_L\rangle$ is a complete lattice and $\leq_p$ is the *precision order*, i.e. $(x_1, y_1) \leq_p (x_2, y_2)$ iff $x_1 \leq_L x_2$ and $y_2 \leq_L y_1$. If we view these objects from a category-theoretic perspective, we can write $\langle L \times L, \leq_p\rangle = \mathcal{L} \times \mathcal{L}^{op}$ where $\mathcal{L} := \langle L, \leq_L\rangle \in \mathrm{Ob}(\mathbf{CLat})$. Hence, we can define the category **BiLat** of square bilattices as follows:

$$\mathrm{Ob}(\mathbf{BiLat}) := \{\mathcal{L} \times \mathcal{L}^{op} \mid \mathcal{L} \in \mathbf{CLat}\}$$
$$\mathrm{Mor}(\mathbf{BiLat}) := \{f\colon \mathcal{L}_1 \to \mathcal{L}_2 \mid \mathcal{L}_1, \mathcal{L}_2 \in \mathrm{Ob}(\mathbf{BiLat}) \wedge f \text{ monotone}\}$$

We will denote an element $\mathcal{L} \times \mathcal{L}^{op}$ of **BiLat** by $\overline{\mathcal{L}}$.

Lemma 1
The category **BiLat** is a full subcategory of **CLat**.

Proof
Clearly, if $\mathcal{L} \in \mathbf{CLat}$, then $\mathcal{L}^{op} \in \mathbf{CLat}$. Since **CLat** is Cartesian closed, for all $\mathcal{L} \in \mathbf{CLat}$, we have that $\mathcal{L} \times \mathcal{L}^{op} \in \mathbf{CLat}$. We conclude by the definition of **BiLat**. □

By Lemma 1, proving that **BiLat** is Cartesian closed reduces to show that the following isomorphisms of complete lattices hold for all $\overline{\mathcal{L}_1}, \overline{\mathcal{L}_2} \in \mathbf{BiLat}$:

1. $\mathcal{T} \cong \overline{\mathcal{T}}$, where $\mathcal{T}$ is the terminal object of **CLat**,
2. $\overline{\mathcal{L}_1} \times \overline{\mathcal{L}_2} \cong \overline{(\mathcal{L}_1 \times \mathcal{L}_2)}$,
3. $\overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}} \cong \overline{\mathcal{L}_2^{\mathcal{L}_1}}$.

While the first two isomorphisms are rather straightforward, the latter deserves some attention. Consider a morphism of square bilattices f from $\overline{\mathcal{L}_1}$ to $\overline{\mathcal{L}_2}$. Since $\overline{\mathcal{L}_2} = \mathcal{L}_2 \times {\mathcal{L}_2}^{op}$, we can write f as a pair (f_1, f_2) of morphisms of complete lattices, where $f_1 \colon \overline{\mathcal{L}_1} \to \mathcal{L}_2$ and $f_2 \colon \overline{\mathcal{L}_1} \to {\mathcal{L}_2}^{op}$. It follows easily that $\overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}} \cong \mathcal{L}_2^{\overline{\mathcal{L}_1}} \times ({\mathcal{L}_2}^{op})^{\overline{\mathcal{L}_1}}$. Then, the isomoprhism $\varphi \colon \overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}} \to \overline{\mathcal{L}_2^{\mathcal{L}_1}}$, is realised by mapping $f \hat{=} (f_1, f_2)$ to a new pair $\varphi(f) := (f_1, f_2') \in \mathcal{L}_2^{\overline{\mathcal{L}_1}} \times (\mathcal{L}_2^{\overline{\mathcal{L}_1}})^{op} = \overline{\mathcal{L}_2^{\mathcal{L}_1}}$, where the second component is defined by $f_2'(x, y) := f_2(y, x)$. Notice that, since f_2 is a monotone function from $\overline{\mathcal{L}_1}$ to ${\mathcal{L}_2}^{op}$, f_2' is indeed a monotone function from $\overline{\mathcal{L}_1}$ to $\mathcal{L}_2$. Further details regarding the isomorphisms listed above are contained in the proof of Theorem 1.

Theorem 1
The category **BiLat** is Cartesian closed.

Proof
Since **CLat** is Cartesian closed, and **BiLat** is a full-subcategory of **CLat** (Lemma 1), it is sufficient to show that the terminal object of **CLat** is an object of **BiLat** and that for all $\overline{\mathcal{L}_1}, \overline{\mathcal{L}_2} \in \mathbf{BiLat}$, the product $\overline{\mathcal{L}_1} \times \overline{\mathcal{L}_2}$ and the exponential $\overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}}$, computed in the category **CLat**, are also objects of **BiLat**.

- *Terminal object.* There is an obvious isomorphism from the terminal object $\mathcal{T}$ of **CLat**, i.e. the lattice with just one element and trivial order, and the object $\mathcal{T} \times \mathcal{T}^{op} \in \mathbf{BiLat}$.
- *Product.* Let $\overline{\mathcal{L}_1}, \overline{\mathcal{L}_2} \in \mathbf{BiLat}$. By Cartesian closedness of **CLat**, $\overline{\mathcal{L}_1} \times \overline{\mathcal{L}_2}$ is an object of **BiLat**. We define a function $\varphi \colon \overline{\mathcal{L}_1} \times \overline{\mathcal{L}_2} \to \overline{\mathcal{L}_1 \times \mathcal{L}_2}$ by sending an element $((a_1, b_1), (a_2, b_2))$ to $((a_1, a_2), (b_1, b_2))$. Clearly, φ is bijective. Moreover, by the definition of the product order, the following double-implications hold for all $a_1, b_1, x_1, y_1 \in \mathcal{L}_1$, and for all $b_1, b_2, x_2, y_2 \in \mathcal{L}_2$

$$
\begin{aligned}
((a_1, b_1), (a_2, b_2)) \leq_{\overline{\mathcal{L}_1} \times \overline{\mathcal{L}_2}} ((x_1, y_1), (x_2, y_2)) & \\
\iff (a_1, b_1) \leq_{\overline{\mathcal{L}_1}} (x_1, y_1) \wedge (a_2, b_2) \leq_{\overline{\mathcal{L}_2}} (x_2, y_2) & \\
\iff a_1 \leq_{\mathcal{L}_1} x_1 \wedge y_1 \leq_{\mathcal{L}_1} b_1 \wedge a_2 \leq_{\mathcal{L}_2} x_2 \wedge y_2 \leq_{\mathcal{L}_2} b_2 & \\
\iff (a_1, a_2) \leq_{\mathcal{L}_1 \times \mathcal{L}_2} (x_1, x_2) \wedge (y_1, y_2) \leq_{\mathcal{L}_1 \times \mathcal{L}_2} (b_1, b_2) & \\
\iff ((a_1, a_2), (b_1, b_2)) \leq_{\overline{\mathcal{L}_1 \times \mathcal{L}_2}} ((x_1, x_2), (y_1, y_2)). &
\end{aligned}
$$

Hence, φ and its inverse are monotone functions, i.e. morphisms. It follows that $\overline{\mathcal{L}_1} \times \overline{\mathcal{L}_2} \cong \overline{\mathcal{L}_1 \times \mathcal{L}_2} \in \mathbf{BiLat}$, as desired.

- *Exponential.* Let $\overline{\mathcal{L}_1}, \overline{\mathcal{L}_2} \in \mathbf{BiLat}$. By Cartesian closedness of **CLat**, $\overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}}$ is an object of **BiLat**. Let $\delta\colon \overline{\mathcal{L}_1} \to \overline{\mathcal{L}_1}$ be the function sending (x, y) to (y, x). We define a function $\psi\colon \overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}} \to \overline{\mathcal{L}_1^{\mathcal{L}_2}}$ by sending a morphism $f := (f_1, f_2)$ to $(f_1, f_2 \circ \delta)$, where $f_1\colon \overline{\mathcal{L}_1} \to \mathcal{L}_2$ and $f_2\colon \overline{\mathcal{L}_1} \to {\mathcal{L}_2}^{op}$ are the components of f. Since f_2 is an antimonotone function from $\overline{\mathcal{L}_1}$ to $\mathcal{L}_2$, it is easy to check that $f_2 \circ \delta$ is a monotone function from $\overline{\mathcal{L}_1}$ to $\mathcal{L}_2$, as desired. Clearly, φ is bijective. Moreover, by the definition of the pointwise order, the following double-implications hold for all $f_1, g_1 \in \mathcal{L}_2^{\overline{\mathcal{L}_1}}$, and for all $f_2, g_2 \in ({\mathcal{L}_2}^{op})^{\overline{\mathcal{L}_1}}$

$$
\begin{aligned}
(f_1, f_2) &\leq_{\overline{\mathcal{L}_2^{\mathcal{L}_1}}} (g_1, g_2) \\
&\iff \forall (x, y) \in \overline{\mathcal{L}_1}, (f_1(x, y), f_2(x, y)) \leq_{\overline{\mathcal{L}_2}} (g_1(x, y), g_2(x, y)) \\
&\iff \forall (x, y) \in \overline{\mathcal{L}_1}, f_1(x, y) \leq_{\mathcal{L}_2} g_1(x, y) \wedge g_2(x, y) \leq_{\mathcal{L}_2} f_2(x, y) \\
&\iff \forall (x, y) \in \overline{\mathcal{L}_1}, f_1(x, y) \leq_{\mathcal{L}_2} g_1(x, y) \wedge g_2(y, x) \leq_{\mathcal{L}_2} f_2(y, x) \\
&\iff f_1 \leq_{\mathcal{L}_2^{\overline{\mathcal{L}_1}}} g_1 \wedge g_2 \circ \delta \leq_{\mathcal{L}_2^{\overline{\mathcal{L}_1}}} f_2 \circ \delta \\
&\iff (f_1, f_2 \circ \delta) \leq_{\overline{\mathcal{L}_1^{\mathcal{L}_2}}} (g_1, g_2 \circ \delta).
\end{aligned}
$$

Hence, ψ and its inverse are monotone functions, i.e. morphisms. It follows that $\overline{\mathcal{L}_2}^{\overline{\mathcal{L}_1}} \cong \overline{\mathcal{L}_1^{\mathcal{L}_2}} \in \mathbf{BiLat}$, as desired. □

It is interesting to observe that the approximators used in standard AFT, i.e., the *symmetric* approximators, when viewed in their square bilattice approximator space, correspond to pairs of equal functions, i.e., the classic definition of exact pair (Denecker et al. 2000). Similarly, a *gracefully degrading* approximator $A = (A_1, A_2)\colon \overline{\mathcal{L}} \to \overline{\mathcal{L}}$, i.e., such that $A_1(x, y) \leq_{\overline{\mathcal{L}}} A_2(y, x)$ for all $(x, y) \in \overline{\mathcal{L}}$ (Denecker and Vennekens 2007), when viewed in $\overline{\mathcal{L}^{\overline{\mathcal{L}}}}$ is a pair $\varphi(A) = (A_1, A_2')$ with $A_1 \leq_{\mathcal{L}^{\overline{\mathcal{L}}}} A_2'$, i.e., a consistent pair according to the classic definition of AFT.

Thanks to Theorem 1 and **BiLat** $\subseteq$ **CPO**, we can fix **BiLat** as our approximation category. This can be done for *any* application in which we want to use standard AFT techniques. Nevertheless, depending on the application at hand, the approximation system may differ. Let us show how to define an approximation system given a language based on a type hierarchy $\mathbb{H}$. Let $S_{\mathbb{T}}$ be the set of the semantics of types of $\mathbb{H}$ we want to approximate, and assume that such semantics are complete lattices, as is usually the case in logic programming. Then, we can inductively define a mapping $App\colon S_{\mathbb{T}} \to \mathrm{Ob}(\mathbf{BiLat})$ by setting, for all $\tau \in \mathcal{B}_{\mathbb{T}}$, $App(E_\tau) := \overline{E_\tau}$, and proceed using the conditions in Definition 12. Notice that the base case of the induction is nothing more than what is usually done in standard AFT: from a complete lattice $\langle L, \leq_L \rangle$ we obtain the square bilattice $\langle L^2, \leq_p \rangle$. The remaining steps are naturally provided by following the Cartesian closed structure of **BiLat**.

For each base type $\tau \in \mathcal{B}_{\mathbb{T}}$, the exact elements of $App(E_\tau)$ are defined as in standard AFT: $(x, y) \in App(E_\tau)$ is exact if $x = y$, i.e., $\mathcal{E}_\tau = \{(x, x) \mid x \in E_\tau\}$. Notice that, since **BiLat** $\subseteq$ **CLat** $\subseteq$ **CJSLat**, the condition 3b in Defintion 12 is satisfied. Finally, for each base type τ, we define $\mathfrak{p}_\tau^0\colon \mathcal{E}_\tau \to E_\tau$ by sending (x, x) to x. Both conditions 4b and 4c in

Definition 12 hold since $(\mathfrak{p}^0_\tau)^{-1}(x) = \{(x, x)\}$. Hence, we have obtained an approximation system $(\mathbf{BiLat}, App, \{\mathcal{E}_\tau\}_{\tau\in\mathcal{B}}, \{\mathfrak{p}^0_\tau\}_{\tau\in\mathcal{B}})$ for $S_\mathbb{T}$.

In standard AFT, we are ultimately interested in the approximation space of interpretations. Given a vocabulary V, $S_\mathbb{T}$ can be easily chosen to contain the semantics of the types of the symbols in V and the space of interpretations for V, i.e., the complete lattice $\Pi_{s\in V'} E_{t(s)}$, where $t(s)$ is the type of the symbol s. It follows that the approximation space of interpretations is $App(\Pi_{s\in V'} E_{t(s)}) = \Pi_{s\in V'} App(E_{t(s)}) \in \mathbf{BiLat}$. Clearly, if we restrict to a vocabulary with only symbols of base type, then we retrieve the usual framework of standard AFT.

5 Revised Extended Consistent AFT

Charalambidis et al. (2018) developed an extension of consistent AFT (Denecker et al. 2003) to generalize the well-founded semantics for classical logic programs to one for programs with higher-order predicates. As already pointed out in Section 2.1, this generalization bears some issues.

In this section, we examine in detail the work of Charalambidis et al. (2018) under the lenses of our novel categorical framework. First, in Subsection 5.1, we present their extension of consistent AFT with their version of approximation spaces, and we prove that this new class of mathematical objects forms a Cartesian closed category. Then, in Subsection 5.2, we briefly recall the types and semantics used by Charalambidis et al. (2018), and we define an approximation system for it. Thanks to the inductive nature of Cartesian closed categories, from the tuple defining the approximation system, we can effortlessly retrieve the entire, complex hierarchy built by Charalambidis et al. (2018). From the definition of the approximation system, we already obtain a concept of exactness for higher-order objects, which was previously missing in Charalambidis et al. (2018). Finally, in Subsection 5.3, we present our solution to the problem encountered in the work of Charalambidis et al. (2018) concerning the semantics of logic programs with existential quantifiers. In particular, we propose a new approximator which provides the expected well-founded semantics. We conclude the subsection with two examples of logic programs in which we need to apply an approximate object on another approximate object.

5.1 The Approximation Category for Extended Consistent AFT

In consistent AFT (Denecker et al. 2003), an approximation space is the consistent part of a square bilattice, i.e., given a bilattice $\overline{\mathcal{L}} = \langle L \times L, \leq_p \rangle$, only the subset $\{(x, y) \mid x \leq_L y\} \subseteq \overline{\mathcal{L}}$ of consistent elements is taken into account. Charalambidis et al. (2018) extended consistent AFT to a new class of approximation spaces: the sets 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*, where L may differ from U.

In order for the machinery of consistent AFT to work over these new spaces, Charalambidis et al. (2018) added some conditions to restrain the possible choices for L and U.

Definition 15

An *approximation tuple* is a tuple $(L, U \leq)$, where L and U are sets, and $\leq$ is a partial order on $L \cup U$ such that the following conditions hold:

1. $\langle L \cup U, \leq \rangle$ has a top element $\top$ and a bottom element $\bot$,
2. $\top, \bot \in L \cap U$,
3. $\langle L, \leq \rangle$ and $\langle U, \leq \rangle$ are complete lattices,
4. *Interlattice Least Upper Bound Property (ILP)*: for all $b \in U$ and for all $S \subseteq L$ such that for all $x \in S$, $x \leq b$, we have $\bigsqcup_L S \leq b$,
5. *Interlattice Greatest Lower Bound Property (IGP)*: for all $a \in L$ and for all $S \subseteq U$ such that for all $x \in S, a \leq x$, we have $a \leq \bigsqcap_U S$.

Definition 16
Let $(L, U, \leq)$ be an approximation tuple. The *approximation space* (associated to $(L, U, \leq)$) is the poset $\langle L \otimes U, \leq_p \rangle$, where $L \otimes U := \{(x, y) \mid x \in L, y \in U, x \leq y\}$, and $\leq_p$ is the partial order defined for all $(x_1, y_1), (x_2, y_2) \in L \otimes U$ by: $(x_1, y_1) \leq_p (x_2, y_2)$ iff $x_1 \leq x_2$ and $y_2 \leq y_1$. We call $\leq_p$ the *precision order* on $L \otimes U$.

In the remainder of this subsection, we prove that the new class of approximation spaces defined in Definition 16 forms a Cartesian closed full subcategory of **CPO** (Theorem 2). First, we define a new category **LUcons**, with objects the approximation spaces just introduced, as follows:

$$\begin{aligned} \mathrm{Ob}(\mathbf{LUcons}) &:= \{\langle L \otimes U, \leq_p \rangle \mid (L, U, \leq) \text{ is an approximation tuple}\} \\ \mathrm{Mor}(\mathbf{LUcons}) &:= \{f \colon A \to B \mid A, B \in \mathrm{Ob}(\mathbf{LUcons}) \wedge f \text{ monotone}\}. \end{aligned}$$

Theorem 2
The category **LUcons** is a Cartesian closed full subcategory of **CPO**.

We split the proof of Theorem 2 into smaller results: first we show that **LUcons** is a full subcategory of **CPO**, then we prove it is Cartesian closed.

Proposition 6
Let $L \otimes U \in \mathrm{Ob}(\mathbf{LUcons})$. Then $L \otimes U$ is a cpo.

Proof
Let $L \otimes U \in \mathrm{Ob}(\mathbf{LUcons})$, and $S \subseteq L \otimes U$ a chain. We denote by $p_1 \colon L \otimes U \to L$ the function of sets sending (x, y) to x, and by $p_2 \colon L \otimes U \to U$ the function sending (x, y) to y. Clearly, $p_1(S)$ and $p_2(S)$ are chains in $\langle L, \leq \rangle$ and $\langle U, \leq \rangle$, respectively. Since $\langle L, \leq \rangle$ and $\langle U, \leq \rangle$ are lattices, there exist $\bigsqcup_L p_1(S) =: x \in p_1(S)$ and $\bigsqcap_U p_2(S) =: y \in p_2(S)$. We now show that $(x, y) \in L \otimes U$, i.e., $x \leq y$. Let $r \in p_1(S) \subseteq L$ and $q \in p_2(S) \subseteq U$. Then, there exist $p \in p_1(S)$ and $s \in p_2(S)$ such that $(r, s), (p, q) \in S$. Since S is a chain, we either have $(r, s) \leq_p (p, q)$ or $(p, q) \leq_p (r, s)$. In both cases, $s \leq q$. By the arbitrarity of q and the IGP, $s \leq y$. By the arbitrarity of s and the ILP, we have $x \leq y$, as desired. Clearly $(x, y) = \bigsqcup_{L \otimes U} S \in L \otimes U$, so it remains to show that $(x, y) \in S$. Since $x \in p_1(S)$ and $y \in p_2(S)$, there exist $x' \in p_1(S)$ and $y' \in p_2(S)$ such that $(x, y'), (x', y) \in S$. By the definitions of x and y, we must have $x' \leq x$ and $y' \geq y$. Suppose $x' < x$ and $y' > y$. Then (x, y') and (x, y') cannot be ordered, which condradicts S being a chain. Hence, either $x = x'$ or $y = y'$. In any case, $(x, y) \in S$, as desired. $\square$

Corollary 1
LUcons is a full subcategory of **CPO**.

Proof
Clear from Proposition 6 and the definition of Mor(**LUcons**). □

Proposition 7
LUcons is a Cartesian closed category.

Proof
Since **CPO** is Cartesian closed, and **LUcons** is a full-subcategory of **CPO** (Corollary 1), it is sufficient to show that the terminal object of **CPO** is an object of **LUcons** and that for all $\mathcal{A}, \mathcal{B} \in \mathbf{LUcons}$, the product $\mathcal{A} \times \mathcal{B}$ and the exponential $\mathcal{B}^{\mathcal{A}}$, computed in the category **CPO**, are also objects of **LUcons**.

Let $\mathcal{A}, \mathcal{B} \in \mathbf{LUcons}$. Let $(L_{\mathcal{A}}, U_{\mathcal{A}}, \leq_{\mathcal{A}})$ and $(L_{\mathcal{B}}, U_{\mathcal{B}}, \leq_{\mathcal{B}})$ be the approximation tuples of the approximation spaces $\mathcal{A}, \mathcal{B} \in \mathbf{LUcons}$. We denote the orders on $\mathcal{A}$ and $\mathcal{B}$ as $\leq_{p,\mathcal{A}}$ and $\leq_{p,\mathcal{B}}$, respectively.

- *Terminal object.* Let $\mathcal{T} = \langle \{*\}, \leq \rangle$ be the cpo with one element, i.e., the terminal object of **CPO**. Clearly, $(\{*\}, \{*\}, \leq)$ is an approximation tuple, and $\mathcal{T} \cong \langle \{*\} \otimes \{*\}, \leq \rangle$ in **CPO**. Hence, $\mathcal{T} \in \mathrm{Ob}(\mathbf{LUcons})$.
- *Product.* This follows from Proposition 2 and Corollary 1.
- *Exponential.* We have to show that $\mathcal{B}^{\mathcal{A}} \in \mathbf{CPO}$ is isomorphic (in **CPO**) to some $\mathcal{C} = L_{\mathcal{C}} \otimes U_{\mathcal{C}} \in \mathbf{LUcons}$. Let $L_{\mathcal{C}} := \hom_{\mathbf{CPO}}(\mathcal{A}, \langle L_{\mathcal{B}}, \leq_{\mathcal{B}} \rangle)$, $U_{\mathcal{C}} := \hom_{\mathbf{CPO}}(\mathcal{A}, \langle U_{\mathcal{B}}, \geq_{\mathcal{B}} \rangle)$, and $\leq_{\mathcal{C}}$ be the restriction onto $L_{\mathcal{C}} \cup U_{\mathcal{C}}$ of the pointwise extension of $\leq_{\mathcal{B}}$, namely for all $f, g \in L_{\mathcal{C}} \cup U_{\mathcal{C}}$,

$$f \leq_{\mathcal{C}} g \iff \forall x \in \mathcal{A}, f(x) \leq_{\mathcal{B}} g(x)$$

We first show that $L_{\mathcal{C}} \otimes U_{\mathcal{C}}$ with the precision order $\leq_{p,\mathcal{C}}$ induced by $\leq_{\mathcal{C}}$ is an object of **LUcons**. In other words, we show that $(L_{\mathcal{C}}, U_{\mathcal{C}}, \leq_{\mathcal{C}})$ is an approximation tuple.

1. The morphisms $\bot_{\mathcal{C}} \colon x \mapsto \bot_{\mathcal{B}}$ and $\top_{\mathcal{C}} \colon x \mapsto \top_{\mathcal{B}}$ are the bottom and top element of $L_{\mathcal{C}} \cup U_{\mathcal{C}}$, respectively.
2. Clearly, $\bot_{\mathcal{C}}, \top_{\mathcal{C}} \in L_{\mathcal{C}} \cap U_{\mathcal{C}}$.
3. Since $\langle L_{\mathcal{B}}, \leq_{\mathcal{B}} \rangle$ and $\langle U_{\mathcal{B}}, \leq_{\mathcal{B}} \rangle$ are complete lattices by definition of approximation space, and $\leq_{\mathcal{C}}$ is the pointwise extension of $\leq_{\mathcal{B}}$, it is straightforward to see that $\langle L_{\mathcal{C}}, \leq_{\mathcal{C}} \rangle$ and $\langle U_{\mathcal{C}}, \leq_{\mathcal{C}} \rangle$ are also complete lattices.
4. Let $g \in U_{\mathcal{C}}$, and let $S \subseteq L_{\mathcal{C}}$ such that for all $f \in S$, $f \leq_{\mathcal{C}} g$, i.e., for all $x \in \mathcal{A}$ we have $f(x) \leq_{\mathcal{B}} g(x)$. We have to show that $\bigsqcup_{L_{\mathcal{C}}} S \leq_{\mathcal{C}} g$. Since $g \in U_{\mathcal{C}}$ and $f \in L_{\mathcal{C}}$, for all $x \in \mathcal{A}$ we have $g(x) \in U_{\mathcal{B}}$ and $S_x := \{f(x) \mid f \in S\} \subseteq L_{\mathcal{B}}$. By using the ILP on $\mathcal{B}$, we get that $\bigsqcup_{L_{\mathcal{B}}} S_x \leq_{\mathcal{B}} g(x)$, for all $x \in \mathcal{A}$. It is not difficult to see that $\bigsqcup_{L_{\mathcal{C}}} S(x) = \bigsqcup_{L_{\mathcal{B}}} S_x$, for all $x \in \mathcal{A}$. Hence, $\bigsqcup_{L_{\mathcal{C}}} S \leq_{\mathcal{C}} g$, as desired.
5. Analogous to Item 4.

Hence, $\mathcal{C} \in \mathbf{LUcons}$. It remains to show that $\mathcal{C}$ is isomorphic to $\mathcal{B}^{\mathcal{A}}$ in **CPO**. Notice that there is an obvious isomorphism of sets

$$\begin{aligned} \mu\colon \hom_{\mathbf{CPO}}(\mathcal{A}, L_{\mathcal{B}} \times U_{\mathcal{B}}) &\to L_{\mathcal{C}} \times U_{\mathcal{C}} \\ f &\mapsto (f_1, f_2), \end{aligned}$$

where f_1, f_2 are the two components of f. By the definitions of the orders (notice the inversion of the order $\leq_{\mathcal{B}}$ on $U_{\mathcal{B}}$ in $U_{\mathcal{C}}$), it is easy to check that μ and μ^{-1} are both well-defined, i.e., they send a monotone function to a pair of monotone functions, and a pair of monotone functions to a monotone function, respectively. Now, let $f \in \hom_{\mathbf{CPO}}(\mathcal{A}, L_{\mathcal{B}} \times U_{\mathcal{B}})$. Then

$$\begin{aligned} f \in \hom_{\mathbf{CPO}}(\mathcal{A}, \mathcal{B}) &\iff \forall x \in \mathcal{A}, f(x) \in \mathcal{B} \\ &\iff \forall x \in \mathcal{A}, \mu_1(f)(x) = f_1(x) \leq_{\mathcal{B}} f_2(x) = \mu_2(f)(x) \\ &\iff \mu_1(f) \leq_{\mathcal{C}} \mu_2(f) \\ &\iff \mu(f) \in L_{\mathcal{C}} \otimes U_{\mathcal{C}}. \end{aligned}$$

Analogously, if $(g, h) \in L_{\mathcal{C}} \otimes U_{\mathcal{C}}$, then $\mu^{-1}(f, g) \in \hom_{\mathbf{CPO}}(\mathcal{A}, \mathcal{B})$. Hence, by restricting domain and codomain of μ, we get another isomorphism of sets $\nu\colon \hom_{\mathbf{CPO}}(\mathcal{A}, \mathcal{B}) \to L_{\mathcal{C}} \otimes U_{\mathcal{C}}$. It remains to show that ν and ν^{-1} are monotone. Let $f, g \in \hom_{\mathbf{CPO}}(\mathcal{A}, \mathcal{B})$ such that $f \leq_{\mathcal{B}^{\mathcal{A}}} g$, i.e., for all $x \in \mathcal{A}$, $f(x) \leq_{p,\mathcal{B}} g(x)$. By the defintion of the precision order, this means that $\nu_1(f)(x) \leq_{\mathcal{B}} \nu_1(g)(x) \leq_{\mathcal{B}} \nu_2(g)(x) \leq_{\mathcal{B}} \nu_2(f)(x)$ for all $x \in \mathcal{A}$. Hence, $\nu_1(f) \leq_{\mathcal{C}} \nu_1(g)$ and $\nu_2(g) \leq_{\mathcal{C}} \nu_2(f)$. It follows by definition that $\nu(f) \leq_{p,\mathcal{C}} \nu(g)$, as desired. The analogous result holds for ν^{-1} and can be shown similarly. Therefore, the corresponding morphism $\nu'\colon \mathcal{B}^{\mathcal{A}} \to \mathcal{C}$ between cpo's is an isomorphism. By Corollary 1, $\mathcal{B}^{\mathcal{A}} \in \mathbf{LUcons}$, as desired. □

5.2 An Approximation System

Thanks to Theorem 2, we can take **LUcons** as the approximation category for any application in which we wish to apply the version of AFT of Charalambidis et al. (2018). Depending on the specific language and semantics at hand, demanded by the application, we would define a different approximation system with **LUcons**. In this subsection, we present the approximation system for the language $\mathcal{HOL}$ and the semantics used in (Charalambidis et al. 2018) to tackle higher-order logic programs.

The language $\mathcal{HOL}$ 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**. For instance, the semantics of type $o \to o$ is simply the poset of functions from E_o to itself, i.e., $E_{o \to o} := (E_o \to E_o)$.

Since the ultimate goal of this application is studying the well-founded semantics of higher order logic programs via AFT, we are interested in the approximation space of Herbrand interpretations. Since Herbrand interpretations fix the value assigned to symbols

of functional types, we only need the approximation spaces for the semantics E_π, for all predicate types π. In other terms, we can focus on the smallest subset S of Ob(**POSet**) containing E_π for all π, and closed under generalized product.

Now the definition of a suitable approximation system for S 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, and a program P over V, the approximation space of Herbrand interpretations of P is $\mathcal{H}_\mathsf{P} := App\left(\Pi_{s\in V} E_{t(s)}\right) = \Pi_{s\in V} App(E_{t(s)}) \in$ Ob(**Approx**), where $t(s)$ is the type of the symbol s.

This greatly simplifies the construction of Charalambidis et al. (2018). 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, by changing the base types and their semantics, this approximation system can be readily adapted to suit other applications.

In conclusion, it is important to stress that we now have a clear concept of exactness: for the base type o the exact elements are $\mathcal{E}_o = \{(\mathbf{t},\mathbf{t}),(\mathbf{f},\mathbf{f})\}$, and for higher-order types, we follow Definition 13. The work of Charalambidis et al. (2018) 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. 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 a predicate of type $o \to o$ and R is a variable of type o. The space of interpretations for p is simply $App(E_{o\to o}) = App(E_o)^{App(E_o)}$, i.e., all the monotone functions from $App(E_o) = \{(\mathbf{f},\mathbf{t}),(\mathbf{f},\mathbf{f}),(\mathbf{t},\mathbf{t})\}$ to itself, as defined above. By the semantics of Charalambidis et al. (2018), 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 (Denecker et al. 2000), 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 is easy to see 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.

5.3 A New Approximator

As presented at the end of Section 2.1, the approximator of Charalambidis et al. (2018) does not provide the expected well-founded semantics for logic programs when there is an existential quantifier in the body of a rule. In this subsection, we propose a new approximator that solves such issue. We achieve this by restricting the set of elements over which certain variables can range. In particular, variables that are arguments of a predicate being defined, i.e., in the head of a rule, can range over all the elements of the approximation spaces of the corresponding types: we want to define the approximation of a higher order predicate also when the argument is an approximation. On the contrary,

variables that appear exclusively in the body of a rule do not need to be approximated. In other words, a variable of type τ that is not argument of any predicate being defined, will be forced to range only over the set $\mathcal{E}_\tau$ of exact elements.

Before stating the new definition for the approximator, we briefly recall the full syntax of $\mathcal{HOL}$. We slightly modify the one presented in (Charalambidis et al. 2018) to make it less heavy.

The *alphabet* of $\mathcal{HOL}$ consists of the following: predicate variables/constants of every predicate type π; individual variables/constants of type ι; the equality constant $\approx$ of type $\iota \to \iota \to o$ for comparing individuals of type ι; the conjunction constant $\wedge$ of type $o \to o \to o$; the rule operator constant $\leftarrow$ of type $o \to o \to o$; and the negation constant $\sim$ of type $o \to o$.

Every predicate variable/constant and every individual variable/constant is a *term*; if E_1 is a term of type $\rho \to \pi$ and E_2 a term of type ρ then $(\mathsf{E}_1\ \mathsf{E}_2)$ is a term of type π. Every term is also an *expression*; if E is a term of type o then $(\sim\mathsf{E})$ is an expression of type o; if E_1 and E_2 are terms of type ι, then $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is an expression of type o.

A *rule* of $\mathcal{HOL}$ is a formula $\mathsf{p}\ \mathsf{R}_1 \cdots \mathsf{R}_n \leftarrow \mathsf{E}_1 \wedge \ldots \wedge \mathsf{E}_m$, where p is a predicate constant of type $\rho_1 \to \cdots \to \rho_n \to o$, $\mathsf{R}_1, \ldots, \mathsf{R}_n$ are distinct variables of types $\rho_1, \ldots, \rho_n$ respectively and the E_i are expressions of type o. The term $\mathsf{p}\ \mathsf{R}_1 \cdots \mathsf{R}_n$ is the *head* of the rule and $\mathsf{E}_1 \wedge \ldots \wedge \mathsf{E}_m$ is the *body* of the rule. For the sake of simplicity, we often write $\mathsf{E}_1, \ldots, \mathsf{E}_m$, in place of $\mathsf{E}_1 \wedge \ldots \wedge \mathsf{E}_m$. A *program* P of $\mathcal{HOL}$ is a finite set of rules. A *state* s of a program P is a function that assigns to each variable R of type ρ, an element of $App(E_\rho)$, if $\rho \neq \iota$, or an element of $E_\iota = D$, if $\rho = \iota$. We denote by $s[\mathsf{R}_1/d_1, \ldots, \mathsf{R}_n/d_n]$ a state that assigns to each R_i the corresponding value d_i, and coincides with s on the other variables.

Finally, we provide the definition for the three-valued semantics of Charalambidis et al. (2018) adapted to the above, slightly-modified definitions.

Definition 17
Let P be a program, $\mathcal{I}$ an interpretation of P, and s a state. The *three-valued semantics* of expressions and bodies is defined as follows:

1. $[\![\mathsf{c}]\!]_s(\mathcal{I}) = \mathsf{c}$, for every individual constant c,
2. $[\![\mathsf{p}]\!]_s(\mathcal{I}) = \mathcal{I}(\mathsf{p})$, for every predicate constant p,
3. $[\![\mathsf{R}]\!]_s(\mathcal{I}) = s(\mathsf{R})$, for every variable R,
4. $[\![(\mathsf{E}_1\ \mathsf{E}_2)]\!]_s(\mathcal{I}) = [\![\mathsf{E}_1]\!]_s(\mathcal{I})\ ([\![\mathsf{E}_2]\!]_s(\mathcal{I}))$,
5. $[\![(\mathsf{E}_1 \bigwedge \mathsf{E}_2)]\!]_s(\mathcal{I}) = \bigsqcap_{\leq_t} \{[\![\mathsf{E}_1]\!]_s(\mathcal{I}), [\![\mathsf{E}_2]\!]_s(\mathcal{I})\}$,
6. $[\![(\sim \mathsf{E})]\!]_s(\mathcal{I}) = ([\![\mathsf{E}]\!]_s(\mathcal{I}))^{-1}$, with $(\mathbf{t},\mathbf{t})^{-1} = (\mathbf{f},\mathbf{f})$, $(\mathbf{f},\mathbf{f})^{-1} = (\mathbf{t},\mathbf{t})$ and $(\mathbf{f},\mathbf{t})^{-1} = (\mathbf{f},\mathbf{t})$,
7. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(\mathcal{I}) = \begin{cases} (\mathbf{t},\mathbf{t}), & \text{if } [\![\mathsf{E}_1]\!]_s(\mathcal{I}) = [\![\mathsf{E}_2]\!]_s(\mathcal{I}) \\ (\mathbf{f},\mathbf{f}), & \text{otherwise} \end{cases}$.

where $\leq_t$ is the *truth order* defined by $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{t}$.

As already explained, we want to restrict a variable appearing only in the body of a rule to range over the elements of $\mathcal{E}_\tau$, with τ being the type of the variable. We call a state s *exact* if for all variables R of type $\tau \neq \iota$, it holds that $s(\mathsf{R}) \in \mathcal{E}_\tau$. We denote by $\mathcal{S}$ the set of exact states.

We have now all the elements to introduce the new approximator, i.e. the *three-valued immediate consequence operator*. For the sake of simplicity, in the following we define $App(E_\iota) := E_\iota = D$, even though $E_\iota \notin S$ and has no associated approximation space.

Definition 18
Let P be a program. The *three-valued immediate consequence operator* $\Psi_\mathsf{P} : \mathcal{H}_\mathsf{P} \to \mathcal{H}_\mathsf{P}$ is defined for every predicate constant $\mathsf{p} : \rho_1 \to \cdots \to \rho_n \to o$ in P, and for all $d_1 \in App(E_{\rho_1}), \ldots, d_n \in App(E_{\rho_n})$, as: $\Psi_\mathsf{P}(\mathcal{I})(\mathsf{p})\ d_1 \cdots d_n = \bigsqcup_{\leq_t} \{[\![\mathsf{E}]\!]_{s[\mathsf{R}_1/d_1,\ldots,\mathsf{R}_n/d_n]}(\mathcal{I}) \mid s \in \mathcal{S}$ and $(\mathsf{p}\ \mathsf{R}_1 \cdots \mathsf{R}_n \leftarrow \mathsf{E})$ in $\mathsf{P}\}$.

With Definition 18, we solve the issue linked to existential quantifiers. Let us review the example presented in Subsection 2.1. We considered a program P with just one rule $\mathsf{p} \leftarrow \mathsf{R} \wedge \sim \mathsf{R}$, where p is a predicate constant of type o, and R is a variable of type o. Observe that in this case the space of Herbrand interpretations is just $\mathcal{H}_\mathsf{P} := App(E_o) \in \mathrm{Ob}(\mathbf{Approx})$, as we are only interested in the interpretation of the predicate p. For all interpretations $\mathcal{I} \in \mathcal{H}_P$, we have

$$\begin{aligned}\Psi_\mathsf{P}(\mathcal{I})(\mathsf{p}) &= \bigsqcup_{\leq_t} \{[\![\mathsf{E}]\!]_s(\mathcal{I}) \mid s \in \mathcal{S} \text{ and } (\mathsf{p} \leftarrow \mathsf{E}) \text{ in } \mathsf{P}\} = \\ &= \bigsqcup_{\leq_t} \{[\![\mathsf{R} \wedge \sim \mathsf{R}]\!]_s(\mathcal{I}) \mid s \in \mathcal{S}\} = \bigsqcup_{\leq_t} \{s(\mathsf{R}) \wedge s(\mathsf{R})^{-1} \mid s \in \mathcal{S}\} = \\ &= \bigsqcup_{\leq_t} \{(\mathbf{f},\mathbf{f}) \wedge (\mathbf{t},\mathbf{t}), (\mathbf{t},\mathbf{t}) \wedge (\mathbf{f},\mathbf{f})\} = (\mathbf{f},\mathbf{f}).\end{aligned} \tag{2}$$

Notice that for this specific program P, both the approximator Ψ_P and the old version from (Charalambidis et al. 2018) do not depend on the interpretation $\mathcal{I}$, but only on the states. While the approximator of Charalambidis et al. (2018) considered any possible state, even the one sending R to $(\mathbf{f},\mathbf{t})$, Ψ_P takes into account only exact states, i.e. R can only be sent to an element of $\mathcal{E}_o = \{(\mathbf{f},\mathbf{f}), (\mathbf{t},\mathbf{t})\}$. This limitation removes the formula $(\mathbf{f},\mathbf{t}) \wedge (\mathbf{f},\mathbf{t}) = (\mathbf{f},\mathbf{t})$ from the least upper bound computation in (2), which was the one causing the evaluation of p being $(\mathbf{f},\mathbf{t})$ for the approximator of Charalambidis et al. (2018).

Since the approximator Ψ_P does not depend on the interpretation, it is immediate to see that the *well-founded* operator $\mathcal{S}_{\Psi_\mathsf{P}}$ coincides with the approximator Ψ_P for all $(I,J) \in \mathcal{H}_\mathsf{P}$:

$$\mathcal{S}_{\Psi_\mathsf{P}}(I,J) = \big(\mathrm{lfp}(\Psi_\mathsf{P}(\cdot,J)_1), \mathrm{lfp}(\Psi_\mathsf{P}(I,\cdot)_2)\big) = \big(\Psi_\mathsf{P}(\cdot,J)_1, \Psi_\mathsf{P}(I,\cdot)_2\big) = \Psi_\mathsf{P}(I,J).$$

It follows that $\mathcal{S}_{\Psi_\mathsf{P}}$ does not depend on the interpretation either. Thus, the least fixpoint of $\mathcal{S}_{\Psi_\mathsf{P}}$, which corresponds to the well-founded model of P, is just the interpretation sending p to $(\mathbf{f},\mathbf{f})$, resulting in a sensible account for the well-founded sementics. Moreover, observe that this interpretation is also exact by our new definition, and it corresponds to the unique exact stable model of the program P.

In the remainder of this section, we present two examples that highlight the importance of enabling the application of approximate objects to approximate objects.

Example 3

Consider an undirected graph given by a predicate `node`: $\iota \to o$, containing all the nodes of the graph, and a predicate `edge`: $\iota \to \iota \to o$ defining the edge relation, which we assume to be symmetric. Some nodes of the graph are *marked*. We call a set of nodes S a *covering* if for every marked node n there exists a node in S with an edge to n. Now, suppose that a Player can modify the set of marked nodes by swapping a marked node with a neighbouring, unmarked node. The goal of the Player is reached when the set of marked nodes is a covering. At that point, the game is over and the Player cannot swap nodes anymore.

The key predicates of our example are contained in Listing 1. We have defined them in terms of a time parameter `T` of type ι, assuming that the Player can only do one swap at a time. In particular, here are the signatures and meanings of the main predicates of Listing 1: `swap`: $\iota \to \iota \to \iota \to o$ indicates whether at a certain time, two nodes are swapped; `marked`: $\iota \to \iota \to o$ represents the set of marked nodes at a specific time (Lines 6 to 8); `covering`: $\iota \to (\iota \to o) \to o$ tells whether at a certain time a set of nodes is a covering (Lines 14, and 15); and `gameOver`: ι expresses whether the game is over at a certain time (Line 18).

Listing 1. *Graph Game.*

```
1  % We define predicates to add/remove nodes to/from marked set.
2  add T X  ← swap T X Y
3  remove T Y ← swap T X Y
4
5  % We define the set of marked nodes based on the marked nodes at
       the previous time point and the last swap.
6  marked T X ← succ T' T, ~(gameOver T), marked T' X,
       ~(remove T' X).
7  marked T X ← succ T' T, ~(gameOver T), add T' X.
8  marked T X ← succ T' T, gameOver T', marked T' X.
9
10 % We define what a covering of a set of nodes is.
11 nextTo S X ← S Y, edge Y X.
12 nonsubset S Q ← S X, ~(Q X).
13 subset S Q ←  ~(nonsubset S Q).
14 ncovering T S ← marked T X, ~(nextTo S X).
15 covering T S ← subset S node, ~(ncovering T S).
16
17 % We define when the game ends.
18 gameOver T ← covering T (marked T).
```

In the formalization of Charalambidis et al. (2018), natural numbers are not taken into account. Thus, we regard the time variable `T` as an individual variable of type ι and we limit our example to only three time points, expressed by the individual constants `a`, `b`, and `c`, related by the successor relation `succ`: $\iota \to \iota \to o$, as expressed in Listing 2. In the same Listing, we also instantiate the nodes and edges of the graph that we chose for this example, and the initial set of marked nodes, i.e. `marked a`.

Listing 2. *Instantiation of time points, graph's nodes and edges, and initial set of marked nodes.*

```
1  % Time points a, b, and c: b successor of a, and c of b.
2  time a.
3  time b.
4  time c.
5  succ b a.
6  succ c b.
```

```
7          % Nodes.
8          node x.
9          node y.
10         node z.
11         node u.
12         node v.
13         % Edges.
14         edge x y.
15         edge x z.
16         edge x u.
17         edge z v.
18         edge u v.
19         % Marked nodes at the start of the game (time point a).
20         marked a y.
21         marked a u.
```

We are only missing the swaps the Player makes at each of the three time points. For example, we could have those listed in Listing 3.

Listing 3. *Swaps.*

```
1        % Nodes x, y, z, u, and v, and swaps at the time points a, b, and
             ↪ c.
2        swap a v u.
3        swap b x y.
4        swap c z x.
```

By joining Listings 1, 2, and 3, we obtain a program P encoding a specific run of the game.

Using the machinery of AFT, we can easily find the well-founded, the stable, the Kripke-Kleene, and the supported models of P. To obtain the well-founded operator, we compute the least fixpoint of the well-founded operator of the approximator contained in Definition 18, i.e., the least fixpoint of $\mathcal{S}_{\Psi_\mathsf{P}} : (x, y) \mapsto (S_{\Psi_\mathsf{P}}(y), S_{\Psi_\mathsf{P}}(x))$, where $S_{\Psi_\mathsf{P}} : x \mapsto \mathrm{lfp}(\Psi_{\mathsf{P}_1}(\cdot, x))$ is the stable operator[1]. Since the well-founded operator is monotone, to find its least fixpoint it is sufficient to repeatedely apply the operator starting from the bottom element of its domain, namely the interpretation sending every predicate constant to the bottom element of the respective approximation space. Notice that during the first iterative applications of the well-founded operator, the predicates `marked`, `covering`, and `gameOver` are being defined only for the first time points, i.e., they are partially defined. In other words, the three-valued interpretations that we obtain from the first computations leading to the well-founded fixpoint, send the aforementioned predicates to approximate objects of the respective approximation spaces. Since `marked`, `covering`, and `gameOver` are all defined by mutual induction, we are forced to apply an approximate object on another approximate object. In particular, in Line 18 of Listing 1, for the definition of `gameOver`, the predicate `covering` is applied on `marked`. Only when the fixpoint is reached, all the predicates being defined will be exact, i.e. two-valued.

In Listing 3, we have provided a specific set of swaps the Player makes. We can obtain a more general setup by using choice rules to define the predicate `swap`, as we do in Listing 4.

Listing 4. *Choice Rules.*

[1] The well-founded, and the stable operator have been briefly introduced in Section 2.1 of the Preliminaries.

```
% Choice rules: the user can swap one edge with a neighbour at
    ↪ each time.
swap T X Y ← node X, node Y, time T, ∼(nswap T X Y).
nswap T X Y ← node X, node Y, time T, ∼(swap T X Y).
% First element must be in the marked set, second cannot
← swap T X Y, ∼(marked T X).
← swap T X Y, marked T Y.
% At most one swap at a time.
← swap T X Y, swap T X' Y', ∼(X' ≈ X).
← swap T X Y, swap T X' Y', ∼(Y' ≈ Y).
% Can only swap neighbors.
← swap T X Y, ∼(edge X Y).
```

By joining Listings 1, 2, and 4, we obtain another program P', and we can again compute the models of interest via AFT. In particular, now each stable model corresponds to a possible run of the game with starting set of marked nodes `marked a`. Notice that, because of the choice rules in Listing 4, the well-founded model of P' leaves most of the predicates undefined.

Example 4

Let us consider a manufacturing company that aims at growing and diversifying its production. We represent raw materials with individual constants of type ι, and finished products with predicate constants of type $\iota \to o$, such that if `P` is any finished product, and `M` is any raw material, then `P M` is true if and only if `M` is necessary to craft `P`. We denote by `materials`: $\iota \to o$ the predicate corresponding to the set of all raw materials, and by `products`: $(\iota \to o) \to o$ the predicate corresponding to the set of all finished products.

We want to define a predicate `production`: $\iota \to \iota \to o$ (Lines 24 and 26 of Listing 5) that indicates which raw materials the company has to acquire for production at a certain time: `production T M` is true if and only if the company acquires the material `M` at time `T`. As in Example 3, we regard the time variable `T` as an individual variable of type ι and we limit our program in Listing 5 to only three time points, related by the successor relation (Lines 2 to 6). We fix the initial set `production a`: $\iota \to o$ of materials the company starts with. At each time point, the company decides which new materials to acquire: we encode the information about these potential new ingredients with the predicate `candidates`: $\iota \to (\iota \to o) \to o$ (Lines 18 and 20), which takes as argument a time point, i.e., an individual variable, and a set of materials, i.e., a predicate of type $\iota \to o$. The selection of new materials the company takes into consideration obeys a few criteria:

1. *Maximize profit*: products necessitating more raw materials to be crafted require more expertise and more capital to invest, but they provide more profit. Hence, as time progresses, the company aims at products more and more complex: at time `T`, a set `P` of raw materials is a candidate if it can be covered by sets corresponding to some finished products of complexity `T` (Lines 14, 16, and 18). We assume a constant predicate `complexity`: $(\iota \to o) \to \iota \to o$ indicating the complexity of a product is given.
2. *Cautiousness*: producing items using only new materials may be risky and time consuming, as the manifacturing team has to acquire novel knowledge, and new suppliers for the raw materials need to be found. Hence, the newly accepted products are required to share at

least one raw material with a product in production at the prevous time point (Lines 8, 14, 16, and 18).

3. *Efficient growth*: as time passes and the company produces more complex items, older, simpler products can be put out of production. This is done gradually: if in `production T` there is still some material that is not needed to craft any product of size `T` or `succ T`, then such material will not be in production at the following time point; otherwise, all materials that are not needed to craft any product of size `succ T` are dropped out of production at time `succ T` (Lines 20). In other words, all the products with the lowest complexity are dropped.

Finally, `production` is just the union of all the candidates (Line 24). If there are no candidates at a certain time point, the production does not vary at the next time point, and the company ends its expansion (Lines 22, and 26).

Listing 5. *The growth of the manifacturing company.*

```
1  % a, b, and c are time points, b is the successor of a, and c of
       ↪ b.
2  time a.
3  time b.
4  time c.
5  succ b a.
6  succ c b.
7  % intersect Q R is true if the intersection between Q and R is
       ↪ non-empty.
8  intersects Q R ← Q X, R X.
9  % subset Q R is true (nonsubset Q R is true) if R is (is not) a
       ↪ subset of Q.
10 nonsubset Q R ← Q X, ~(R X).
11 subset Q R ←  ~(nonsubset Q R).
12
13 % There exists a product P made of T raw materials, all
       ↪ belonging to S, some belonging to production T', and
       ↪ including M.
14 existsubprod T S M ← products P, complexity P T, subset P S,
       ↪ P M, intersects P (production T'), succ T T'.
15 % S cannot be covered by products of size T if there exists a
       ↪ material M in S that is never part of a subproduct of S.
16 notcovered T S ← S M,  ~(existsubprod T S M), time T.
17 % S is a candidate at time T if it can be covered and it is a
       ↪ set of materials (maximize profit, and cautiousness).
18 candidates T S ← ~notcovered T S, time T, subset S materials.
19 % S is a candidate if it is a product of size T', in production
       ↪ at time T', and if in production T' there were still raw
       ↪ materials only needed for products of size different than
       ↪  T' (efficient growth).
20 candidates T S ← ~(candidates T' (production T')), products P,
       ↪ subset P (production T'), complexity P T', succ T T'.
21 % There exists a candidate at time T.
22 existcandidate T ← candidates T S.
23 % A material is needed for production at time T if it is in one
       ↪ of the candidate sets at time T.
24 production T M ← candidates T S, S M.
25 % If there are no candidates at time T, then the production
       ↪ remains the same.
26 production T M ← ~(existcandidate T), production T' M,
       ↪ succ T T'.
```

Note that the symbols `Q`, `R`, and `S`, highlighted in red, are predicate variables of type $\iota \to o$ that are used in a higher-order style in the rules in Lines 8, 10, 16, and 24.

Similarly to what happens for Example 3, since the predicates `candidates` and `production` are defined by mutual induction, they will be fully defined only at the end of the least fixpoint construction. However, in order to define `candidates` we need to apply it to `production` (Line 20). In particular, before reaching the fixpoint, we will need to apply an approximate object, namely `candidates`, on another approximate object, i.e., `production`.

6 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 use of CT not only provides a solid, formal mathematical foundation to work with, but also allows to reduce the complexity of proofs. 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 standard AFT and the extension of consistent AFT of Charalambidis et al. (2018). Moreover, concerning the latter version of AFT mentioned, we have resolved its problematic features. In particular, we provided a novel approximator which behaves properly, even on programs with existential quantifiers in the body of rules, and we defined the concept of exactness, previously missing in the work of Charalambidis et al. (2018), allowing to consider exact stable models.

As far as future work and developments are concerned, it is paramount to notice that by systematically extending present (and possibly future) versions of AFT to the higher-order setting, this paper further enriches the vast body of algebraic results on AFT. In particular, this allows us to make all the already-existing formal results regarding AFT readily available in a higher-order AFT context. This includes, but it is not limited to, stratification results (Vennekens et al. 2006; Bogaerts and Cruz-Filipe 2021), grounded fixpoints (Bogaerts et al. 2015), well-founded induction (Denecker and Vennekens 2007), and strong equivalence (Truszczynski 2006). Moreover, in light of this newly established bridge to the higher-order environment, one could explore the possibilities within the application fields where AFT previously succeeded, such as abstract argumentation (Strass 2013; Bogaerts 2019), active integrity constraints (Bogaerts and Cruz-Filipe 2018), stream reasoning (Antic 2020), integrity constraints for the semantic web (Bogaerts and Jakubowski 2021), and Datalog (Pollaci 2025).

Lastly, it may be of interest to research how the developed higher-order semantics and language $\mathcal{HOL}$ presented in Section 5 relate to Hilog and Prolog with meta-predicates (Chen et al. 1993). In particular, $\mathcal{HOL}$ and Prolog show two rather different natures: while Prolog is procedural and intensional, the language $\mathcal{HOL}$ provides a declarative and extensional approach. This is indeed not trivial to obtain for the higher-order setting, as it was also pointed out by Rondogiannis and Symeonidou (2018).

Competing Interests

The authors declare none.

References

Antic, C. 2020. Fixed point semantics for stream reasoning. *Artif. Intell.*, *288*, 103370.

Bogaerts, B. Weighted abstract dialectical frameworks through the lens of approximation fixpoint theory. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019* 2019, pp. 2686–2693. AAAI Press.

Bogaerts, B., Charalambidis, A., Chatziagapis, G., Kostopoulos, B., Pollaci, S., and Rondogiannis, P. 2024. The stable model semantics for higher-order logic programming. *Theory Pract. Log. Program.*, *24*, 4, 737–754.

Bogaerts, B. and Cruz-Filipe, L. 2018. Fixpoint semantics for active integrity constraints. *Artif. Intell.*, *255*, 43–70.

Bogaerts, B. and Cruz-Filipe, L. 2021. Stratification in approximation fixpoint theory and its application to active integrity constraints. *ACM Trans. Comput. Log.*, *22*, 1, 6:1–6:19.

Bogaerts, B. and Jakubowski, M. Fixpoint semantics for recursive SHACL. In Formisano, A., Liu, Y. A., Bogaerts, B., Brik, A., Dahl, V., Dodaro, C., Fodor, P., Pozzato, G. L., Vennekens, J., and Zhou, N., editors, *Proceedings 37th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021* 2021, volume 345 of *EPTCS*, pp. 41–47.

Bogaerts, B., Vennekens, J., and Denecker, M. 2015. Grounded fixpoints and their applications in knowledge representation. *Artif. Intell.*, *224*, 51–71.

Charalambidis, A. and Rondogiannis, P. Categorical approximation fixpoint theory. In Gaggl, S. A., Martinez, M. V., and Ortiz, M., editors, *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings* 2023, volume 14281 of *Lecture Notes in Computer Science*, pp. 515–530. Springer.

Charalambidis, A., Rondogiannis, P., and Symeonidou, I. 2018. Approximation fixpoint theory and the well-founded semantics of higher-order logic programs. *Theory Pract. Log. Program.*, *18*, 3-4, 421–437.

Chen, W., Kifer, M., and Warren, D. S. 1993. HILOG: A foundation for higher-order logic programming. *J. Log. Program.*, *15*, 3, 187–230.

Clark, K. L. Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977* 1977, Advances in Data Base Theory, pp. 293–322, New York. Plemum Press.

Dasseville, I., van der Hallen, M., Bogaerts, B., Janssens, G., and Denecker, M. A compositional typed higher-order logic with definitions. In Carro, M., King, A., Saeedloei, N., and Vos, M. D., editors, *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA* 2016, volume 52 of *OASIcs*, pp. 14:1–14:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Dasseville, I., van der Hallen, M., Janssens, G., and Denecker, M. 2015. Semantics of templates in a compositional framework for building logics. *Theory Pract. Log. Program.*, *15*, 4-5, 681–695.

Denecker, M., Bruynooghe, M., and Vennekens, J. Approximation fixpoint theory and the semantics of logic and answers set programs. In Erdem, E., Lee, J., Lierler, Y., and Pearce, D., editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz* 2012, volume 7265 of *Lecture Notes in Computer Science*, pp. 178–194. Springer.

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

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

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

Denecker, M., Marek, V. W., and Truszczynski, M. 2004. Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Inf. Comput.*, *192*, 1, 84–121.

Denecker, M. and Vennekens, J. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In Baral, C., Brewka, G., and Schlipf, J. S., editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings* 2007, volume 4483 of *Lecture Notes in Computer Science*, pp. 84–96. Springer.

Fitting, M. 1985. A Kripke-Kleene semantics for logic programs. *J. Log. Program.*, *2*, 4, 295–312.

Fitting, M. 2002. Fixpoint semantics for logic programming a survey. *Theor. Comput. Sci.*, *278*, 1-2, 25–51.

Gelfond, M. and Lifschitz, V. The stable model semantics for logic programming. In Kowalski, R. A. and Bowen, K. A., editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)* 1988, pp. 1070–1080. MIT Press.

Heyninck, J., Arieli, O., and Bogaerts, B. 2024. Non-deterministic approximation fixpoint theory and its application in disjunctive logic programming. *Artif. Intell.*, *331*, 104110.

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

Pollaci, S. 2025. Fixpoint semantics for datalogmtl with negation.

Riehl, E. 2017. *Category theory in context*. Aurora: Dover modern math originals. Dover Publications.

Rondogiannis, P. and Symeonidou, I. 2018. Extensional semantics for higher-order logic programs with negation. *Log. Methods Comput. Sci.*, *14*, 2.

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

Truszczynski, M. 2006. Strong and uniform equivalence of nonmonotonic theories - an algebraic approach. *Ann. Math. Artif. Intell.*, *48*, 3-4, 245–265.

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

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

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