The Category of Approximation Spaces*

Samuele Pollaci[1,2]

[1] Vrije Universiteit Brussel, Brussels, Belgium
samuele.pollaci@vub.be
[2] KU Leuven, Leuven, Belgium
samuele.pollaci@kuleuven.be

Approximation Fixpoint Theory (AFT) [5] is an algebraic framework designed to study the semantics of non-monotonic logics, like logic programming, autoepistemic logic, and default logic, and to resolve longstanding problems on the relation between these formalisms [6]. The core ideas of AFT are relatively simple: we are interested in fixpoints of an operator on a given lattice $\langle L, \leq \rangle$. For monotonic operators, Tarski's theory guarantees the existence of a least fixpoint. AFT generalizes Tarki's theory to non-monotonic operators by making use of a so-called *approximating operator*: an operator $A : L^2 \to L^2$ 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 are used to approximate elements of L: the tuple $(x, y) \in L^2$ is said to approximate z if $x \leq z \leq y$. Given such an approximator, AFT defines several types of fixpoints (supported fixpoints, a Kripke-Kleene fixpoint, stable fixpoints, and a well-founded fixpoint) of interest.

Let us illustrate the application of AFT to standard, first-order, logic programming. In this setting, the lattice L is the lattice of interpretations, ordered by the truth order $I \leq J$ if $P^I \subseteq P^J$ for each predicate P. The operator at hand is the immediate consequence operator T_P of a logic program P [9]. In this setting, pairs (I, J) can be seen as *four-valued* interpretations: a fact q is true if it is *true* in both I and J, *false* if it is 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 more than Fitting's four-valued immediate consequence operator [7].

This research is motivated by a need to apply AFT to *higher-order* logic programming that arose in several contexts [3, 2, 8]. An important issue in this context is that using pairs of interpretations no longer allows for an obvious way to evaluate formulas in an approximation. Let us illustrate this with an example. Consider a logic program in which a first-order predicate p and a second-order predicate Q are defined. Now assume that in the body of a rule, the atom $Q(p)$ occurs. A tuple (I, J) of interpretations in this case tells us for any given set S if $Q(S)$ is true, false, unknown, or inconsistent. However, the interpretation of p in such an interpretation (I, J) is not a set, but a partially defined set, making it hard to evaluate expressions of the form $Q(p)$. To deal with definitions of higher-order objects, approximate interpretations should take into account the application of approximate objects to approximate objects. This suggests 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. The main question is how to define the base approximation spaces and the class of functions in a generic way that works in all applications of AFT.

Clearly, we want to be able to apply the same AFT techniques at any level of the hierarchy, i.e. all approximation spaces should share the same algebraic structure. In Category Theory (CT), there already exists a notion that captures this behavior: the concept of *Cartesian closed*

*This research has been conducted under the supervision of Bart Bogaerts (Vrije Universiteit Brussel, Brussels, Belgium), and Marc Denecker (KU Leuven, Leuven, Belgium).

category (ccc). The objects of a ccc **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**. It follows that, if the base approximation spaces are objects of a ccc, then the category also contains the full hierarchy of approximation spaces we are aiming for. We will call such a ccc an *approximation category* and denote it by **Approx**. Clearly, the definition of **Approx** 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 [5], consistent AFT [4], or other extensions [1]). To formalize this, we develop the notion of an *approximation system*. Once a language and the semantics of its types are fixed, we can choose an approximation system that consists, among other things, of a ccc **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. Thanks to the generality of this formalization, there are several viable choices for an approximation system. For instance, we show that the bilattices form a ccc with the monotone functions as morphisms. With a suitable choice of *App* and exact elements we obtain an approximation system that recovers the framework of standard AFT and extends it to higher-order objects. Furthermore, we have shown that the approximation spaces from [1] form a ccc. Our approach provides a clear definition for exact higher-order elements, which was missing in the work [1].

References

[1] Angelos Charalambidis, Panos Rondogiannis, and Ioanna Symeonidou. Approximation fixpoint theory and the well-founded semantics of higher-order logic programs. *Theory Pract. Log. Program.*, 18(3-4):421–437, 2018.

[2] Ingmar Dasseville, Matthias van der Hallen, Bart Bogaerts, Gerda Janssens, and Marc Denecker. A compositional typed higher-order logic with definitions. In *ICLP (Technical Communications)*, volume 52 of *OASIcs*, pages 14:1–14:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[3] Ingmar Dasseville, Matthias van der Hallen, Gerda Janssens, and Marc Denecker. Semantics of templates in a compositional framework for building logics. *Theory Pract. Log. Program.*, 15(4-5):681–695, 2015.

[4] Marc Denecker, V. Wiktor Marek, Miroslaw Truszczynski. Uniform semantic treatment of default and autoepistemic logic. *Artif. Intell.*, 143(1):79–122, 2003.

[5] Marc Denecker, Victor Marek, and Mirosław Truszczyński. Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In *Logic-Based Artificial Intelligence*, volume 597, pages 127–144. Springer US, 2000.

[6] Marc Denecker, Victor Marek, Mirosław Truszczyński. Reiter's default logic is a logic of autoepistemic reasoning and a good one, too. In *Nonmonotonic Reasoning – Essays Celebrating Its 30th Anniversary*, pages 111–144. College Publications, 2011.

[7] Melvin Fitting. Fixpoint semantics for logic programming a survey. *Theor. Comput. Sci.*, 278(1-2):25–51, 2002.

[8] Panos Rondogiannis, and Ioanna Symeonidou. Extensional Semantics for Higher-Order Logic Programs with Negation. *Log. Methods Comput. Sci.*, 14(2), 2018.

[9] Maarten H. van Emden, and Robert A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *J. ACM*, 23(4):733–742, 1976.