Acta Cybernetica **23** (2017) 319–326.

Overview of an Abstract Fixed Point Theory for Non-Monotonic Functions and its Applications to Logic Programming

Angelos Charalambidis[a] and Panos Rondogiannis[a]

Abstract

The purpose of the present paper is to give an overview of our joint work with Zoltán Ésik, namely the development of an abstract fixed point theory for a class of non-monotonic functions [4] and its use in providing a novel denotational semantics for a very broad extension of classical logic programming [1]. Our purpose is to give a high-level presentation of the main developments of these two works, that avoids as much as possible the underlying technical details, and which can be used as a mild introduction to the area.

Keywords: fixed point theory, higher-order logic programming, semantics of logic programming

1 Introduction

The purpose of this paper is to present an overview of the authors' joint work with Zoltán Ésik. This work [4] concerned the development of an abstract fixed point theory for a class of functions that exhibit a type of "monotonicity in layers" but which are overall non-monotonic. Such functions prove to be quite common in various investigations in logic programming and formal language theory, and may potentially have other applications. We also describe our development [1], based on the aforementioned abstract framework, of a novel denotational semantics for a very broad extension of classical logic programming. In the rest of this section we provide a short description of the beginnings of our collaboration with Zoltán that led to the above results.

In 2005, the second author together with Bill Wadge proposed [5] the infinite-valued semantics for logic programs with negation. This particular work was somewhat ad-hoc, namely the main results relied on techniques custom-tailored for logic programming. In 2013, the second author of the present paper, together with Zoltán

[a]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, E-mail: {a.charalambidis,prondo}@di.uoa.gr

DOI: 10.14232/actacyb.23.1.2017.17

Ésik started a collaboration supported by a "Greek-Hungarian Scientific Collaboration Program" with title "*Extensions and Applications of Fixed Point Theory for Non-Monotonic Formalisms*". The purpose of the program was to create an abstract fixed point theory based on the infinite-valued approach, namely a theory that would not only be applicable to logic programs but also to other non-monotonic formalisms. This abstract theory was successfully developed and is described in detail in [4]. As an application of these results, this abstract theory was used in [1] in order to obtain the first extensional semantics for higher-order logic programs with negation. Another application of the new theory to the area of non-monotonic formal grammars was proposed in [3]. Moreover, Zoltán himself further investigated the foundations and the properties of the infinite-valued approach [2], highlighting some of its desirable characteristics. Unfortunately, the further joint development of the abstract infinite-valued approach to non-monotonic fixed point theory, was abruptly interrupted by the untimely loss of Zoltán.

In the next section we describe the basic concepts behind the abstract approach to non-monotonic fixed point theory. In Section 3 we describe the application of the theory to the class of higher-order logic programs with negation. The paper concludes by giving pointers for future work.

2 Non-Monotonic Fixed Point Theory

Suppose that $(L, \leq)$ is a complete lattice in which the least upper bound operation is denoted by $\bigvee$ and the least element is denoted by $\perp$. Let $\kappa > 0$ be a fixed ordinal. We assume that for each ordinal $\alpha < \kappa$, there exists a preordering $\sqsubseteq_\alpha$ on L. We denote with $=_\alpha$ the equivalence relation determined by $\sqsubseteq_\alpha$. We define $x \sqsubset_\alpha y$ iff $x \sqsubseteq_\alpha y$ but $x =_\alpha y$ does not hold. Finally, we define $\sqsubset = \bigcup_{\alpha<\kappa} \sqsubset_\alpha$ and let $x \sqsubseteq y$ iff $x \sqsubset y$ or $x = y$. Given an ordinal $\alpha < \kappa$ and $x \in L$, define $(x]_\alpha = \{y \in L : \forall \beta < \alpha \quad x =_\beta y\}$. We require of our relations to satisfy the following axioms:

Axiom 1. *For all ordinals $\alpha < \beta < \kappa$, $\sqsubseteq_\beta$ is included in $=_\alpha$.*

Axiom 2. *$\bigcap_{\alpha<\kappa} =_\alpha$ is the identity relation on L.*

Axiom 3. *For each $x \in L$, for every ordinal $\alpha < \kappa$, and for any $X \subseteq (x]_\alpha$ there is some $y \in (x]_\alpha$ such that:*

- *$X \sqsubseteq_\alpha y$, and*
- *for all $z \in (x]_\alpha$, if $X \sqsubseteq_\alpha z$ then $y \sqsubseteq_\alpha z$ and $y \leq z$.*

Axiom 4. *If $x_j, y_j \in L$ and $x_j \sqsubseteq_\alpha y_j$ for all $j \in J$ then $\bigvee\{x_j : j \in J\} \sqsubseteq_\alpha \bigvee\{y_j : j \in J\}$.*

The element y specified by the Axiom 3 above, can be shown to be unique and we denote it by $\bigsqcup_\alpha X$.

In the following, we will often talk about "*models of the Axioms 1-4*" (or simply "*models*"). More formally:

Definition 1. *A* model of Axioms 1-4 *or simply* model *consists of a complete lattice $(L, \leq)$, an ordinal $\kappa > 0$ and a set of preorders $\sqsubseteq_\alpha$ for every $\alpha < \kappa$, such that Axioms 1-4 are satisfied.*

Under the above axioms, the following theorem is established in [4]:

Theorem 1. *$(L, \sqsubseteq)$ is a complete lattice.*

The following definition will lead us to the main theorem of [4]:

Definition 2. *Suppose that L is a model and let $\alpha < \kappa$. A function $f : L \to L$ is called α-monotonic if for all $x, y \in L$, if $x \sqsubseteq_\alpha y$ then $f(x) \sqsubseteq_\alpha f(y)$.*

The central fixed point theorem of [4] can now be stated:

Theorem 2. *Let L be a model. Suppose that $f : L \to L$ is α-monotonic for each ordinal $\alpha < \kappa$. Then f has a least pre-fixed point with respect to the partial order $\sqsubseteq$, which is also the least fixed point of f.*

The article [4] contains many more results, but one could say that the above theorem is possibly the main technical achievement. Actually, the above theorem is also the main tool that we will need in the developments of the next section.

3 Higher-Order Logic Programs with Negation

In this section we present the application of the non-monotonic fixed point theory to the class of higher-order logic programs with negation. The approach presented naturally extends the ideas behind the infinite-valued approach proposed in [5] into a higher-order setting. The basic idea behind the approach in [5] is that in order to obtain minimum model semantics for higher-order logic programs with negation it is necessary to consider a multi-valued logic. We first present the syntax and then the semantics of our language.

3.1 Syntax

Our higher-order logic programming language is based on a simple type system that supports two base types: o, the boolean domain, and ι, the domain of individuals (data objects). The composite types are partitioned into three classes: functional (assigned to individual constants, individual variables and function symbols), predicate (assigned to predicate constants and variables) and argument (assigned to parameters of predicates).

Definition 3. *A type τ can either be functional, argument, or predicate, denoted as σ, π and ρ respectively and defined as:*

$$\sigma := \iota \mid \iota \to \sigma$$
$$\pi := o \mid \rho \to \pi$$
$$\rho := \iota \mid \pi$$

Definition 4. *The set of* expressions *of our higher-order language is defined as follows:*

1. *Every predicate variable (respectively, predicate constant) of type π is an expression of type π; every individual variable (respectively, individual constant) of type ι is an expression of type ι; the propositional constants* false *and* true *are expressions of type o.*
2. *If f is an n-ary function symbol and $\mathsf{E}_1, \ldots, \mathsf{E}_n$ are expressions of type ι, then $(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)$ is an expression of type ι.*
3. *If E_1 is an expression of type $\rho \to \pi$ and E_2 is an expression of type ρ, then $(\mathsf{E}_1\ \mathsf{E}_2)$ is an expression of type π.*
4. *If V is an argument variable of type ρ and E is an expression of type π, then $(\lambda \mathsf{V}.\mathsf{E})$ is an expression of type $\rho \to \pi$.*
5. *If $\mathsf{E}_1, \mathsf{E}_2$ are expressions of type π, then $(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)$ and $(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)$ are expressions of type π.*
6. *If E is an expression of type o, then $(\sim\!\mathsf{E})$ is an expression of type o.*
7. *If $\mathsf{E}_1, \mathsf{E}_2$ are expressions of type ι, then $(\mathsf{E}_1 \approx \mathsf{E}_2)$ is an expression of type o.*
8. *If E is an expression of type o and V is a variable of type ρ then $(\exists_\rho \mathsf{V}\, \mathsf{E})$ is an expression of type o.*

The notions of *free* and *bound* variables of an expression are defined as usual. An expression is called *closed* if it does not contain any free variables.

A *program clause* is a clause $\mathsf{p} \leftarrow_\pi \mathsf{E}$ where p is a predicate constant of type π and E is a closed expression of type π. A *program* is a finite set of program clauses.

3.2 Semantics

We start by examining the semantics of types. The most crucial case is that of the boolean domain o. The boolean values range over a partially ordered set $(V, \leq)$ of truth values. The number of truth values of V will be specified with respect to an ordinal $\kappa > 0$. The set $(V, \leq)$ is the following:

$$F_0 < F_1 < \cdots < F_\alpha < \cdots < 0 < \cdots < T_\alpha < \cdots < T_1 < T_0$$

where $\alpha < \kappa$. Intuitively, F_0 and T_0 are the classical *False* and *True* values and 0 is the undefined value. The new values express different levels of truthness and falsity. The *order* of a truth value is defined as follows: $order(T_\alpha) = \alpha$, $order(F_\alpha) = \alpha$ and $order(0) = +\infty$.

We define the following preorderings $\sqsubseteq_\alpha$ on the set V for each $\alpha < \kappa$:

1. $x \sqsubseteq_\alpha x$ if $order(x) < \alpha$;
2. $F_\alpha \sqsubseteq_\alpha x$ and $x \sqsubseteq_\alpha T_\alpha$ if $order(x) \geq \alpha$;
3. $x \sqsubseteq_\alpha y$ if $order(x), order(y) > \alpha$.

We then have the following result from [1]:

Lemma 1. *$(V, \leq)$ is a complete lattice and a model.*

Let us denote by $[A \stackrel{m}{\to} B]$ the set of functions from A to B that are α-monotonic for all $\alpha < \kappa$. Based on the above discussion, we can now state the semantics of all the types of our language:

Definition 5. *Let D be a nonempty set. Then:*

- $[\![o]\!]_D = V$, *and $\leq_o$ is the partial order of V;*
- $[\![\iota]\!]_D = D$, *and $\leq_\iota$ is the trivial partial order such that $d \leq_\iota d$, for all $d \in D$;*
- $[\![\iota^n \to \iota]\!]_D = D^n \to D$. *A partial order in this case will not be needed;*
- $[\![\iota \to \pi]\!]_D = D \to [\![\pi]\!]_D$, *and $\leq_{\iota\to\pi}$ is the partial order defined as follows: for all $f, g \in [\![\iota \to \pi]\!]_D$, $f \leq_{\iota\to\pi} g$ iff $f(d) \leq_\pi g(d)$ for all $d \in D$;*
- $[\![\pi_1 \to \pi_2]\!]_D = [[\![\pi_1]\!]_D \stackrel{m}{\to} [\![\pi_2]\!]_D]$, *and $\leq_{\pi_1\to\pi_2}$ is the partial order defined as follows: for all $f, g \in [\![\pi_1 \to \pi_2]\!]_D$, $f \leq_{\pi_1\to\pi_2} g$ iff $f(d) \leq_{\pi_2} g(d)$ for all $d \in [\![\pi_1]\!]_D$.*

Moreover, we have the following relations $\sqsubseteq_\alpha$ on our domains:

- The relation $\sqsubseteq_\alpha$ on $[\![o]\!]_D$ is the relation $\sqsubseteq_\alpha$ on V.
- The relation $\sqsubseteq_\alpha$ on $[\![\rho \to \pi]\!]_D$ is defined as follows: $f \sqsubseteq_\alpha g$ iff $f(d) \sqsubseteq_\alpha g(d)$ for all $d \in [\![\rho]\!]_D$.

The following lemma can then be established following the results of [4]:

Lemma 2. *Let D be a non-empty set and π be a predicate type. Then, $([\![\pi]\!]_D, \leq_\pi)$ is a complete lattice and a model.*

For the rest of the section we focus on Herbrand interpretations and we assume for a program P, $D = U_\mathsf{P}$ where U_P is the Herbrand universe and therefore we simple write $[\![\tau]\!]$ instead of $[\![\tau]\!]_{U_\mathsf{P}}$. A Herbrand interpretation I for a program P is a function that maps a predicate of type π to an element of $[\![\pi]\!]$. The set of all the interpretation of P is denoted by $\mathcal{I}_\mathsf{P}$. It follows directly from the results of [4] that $\mathcal{I}_\mathsf{P}$ is a complete lattice and a model. A Herbrand state s is a function that assigns to each argument variable V of type ρ, of an element $s(\mathsf{V}) \in [\![\rho]\!]_{U_\mathsf{P}}$.

Let I be a Herbrand interpretation and s be a Herbrand state. The semantics of expressions with respect to I and s, is defined as follows:

1. $[\![\mathsf{false}]\!]_s(I) = F_0$
2. $[\![\mathsf{true}]\!]_s(I) = T_0$
3. $[\![\mathsf{c}]\!]_s(I) = I(\mathsf{c})$, for every individual constant c
4. $[\![\mathsf{p}]\!]_s(I) = I(\mathsf{p})$, for every predicate constant p
5. $[\![\mathsf{V}]\!]_s(I) = s(\mathsf{V})$, for every argument variable V
6. $[\![(\mathsf{f}\ \mathsf{E}_1 \cdots \mathsf{E}_n)]\!]_s(I) = I(\mathsf{f})\ [\![\mathsf{E}_1]\!]_s(I) \cdots [\![\mathsf{E}_n]\!]_s(I)$, for every n-ary function symbol f
7. $[\![(\mathsf{E}_1 \mathsf{E}_2)]\!]_s(I) = [\![\mathsf{E}_1]\!]_s(I)([\![\mathsf{E}_2]\!]_s(I))$
8. $[\![(\lambda \mathsf{V}.\mathsf{E})]\!]_s(I) = \lambda d.[\![\mathsf{E}]\!]_{s[\mathsf{V}/d]}(I)$, where d ranges over $[\![type(\mathsf{V})]\!]_D$
9. $[\![(\mathsf{E}_1 \bigvee_\pi \mathsf{E}_2)]\!]_s(I) = \bigvee_\pi\{[\![\mathsf{E}_1]\!]_s(I), [\![\mathsf{E}_2]\!]_s(I)\}$, where $\bigvee_\pi$ is the lub function on $[\![\pi]\!]_D$
10. $[\![(\mathsf{E}_1 \bigwedge_\pi \mathsf{E}_2)]\!]_s(I) = \bigwedge_\pi\{[\![\mathsf{E}_1]\!]_s(I), [\![\mathsf{E}_2]\!]_s(I)\}$, where $\bigwedge_\pi$ is the glb function on $[\![\pi]\!]_D$
11. $[\![(\sim\mathsf{E})]\!]_s(I) = \begin{cases} T_{\alpha+1} & \text{if } [\![\mathsf{E}]\!]_s(I) = F_\alpha \\ F_{\alpha+1} & \text{if } [\![\mathsf{E}]\!]_s(I) = T_\alpha \\ 0 & \text{if } [\![\mathsf{E}]\!]_s(I) = 0 \end{cases}$
12. $[\![(\mathsf{E}_1 \approx \mathsf{E}_2)]\!]_s(I) = \begin{cases} T_0, & \text{if } [\![\mathsf{E}_1]\!]_s(I) = [\![\mathsf{E}_2]\!]_s(I) \\ F_0, & \text{otherwise} \end{cases}$
13. $[\![(\exists \mathsf{V}\ \mathsf{E})]\!]_s(I) = \bigvee_{d \in [\![type(\mathsf{V})]\!]_D} [\![\mathsf{E}]\!]_{s[\mathsf{V}/d]}(I)$

Definition 6. *Let* P *be a program and let* M *be a Herbrand interpretation of* P*. Then* M *will be called a* model *of* P *iff for all clauses* $\mathsf{p} \leftarrow_\pi \mathsf{E}$ *of* P*, it holds* $[\![\mathsf{E}]\!](M) \leq_\pi M(\mathsf{p})$*, where* $M(\mathsf{p}) \in [\![\pi]\!]$.

We can now define the immediate consequence operator for our language:

Definition 7. *Let* P *be a program. The mapping* $T_{\mathsf{P}} : \mathcal{I}_{\mathsf{P}} \to \mathcal{I}_{\mathsf{P}}$ *is defined for every* $\mathsf{p} : \pi$ *and for every* $I \in \mathcal{I}_{\mathsf{P}}$ *as*

$$T_{\mathsf{P}}(I)(\mathsf{p}) = \bigvee \{ [\![\mathsf{E}]\!](I) : (\mathsf{p} \leftarrow_{\pi} \mathsf{E}) \in \mathsf{P} \}$$

As it turns out, T_{P} enjoys the α-monotonicity property [1]:

Lemma 3. *For all* $\alpha < \kappa$*,* T_{P} *is* α*-monotonic.*

We now have all we need in order to apply the main Theorem of [4], getting the following result [1]:

Theorem 3 (Least Fixed Point Theorem)**.** *Let* P *be a program and let* $\mathcal{M}$ *be the set of all its Herbrand models. Then,* T_{P} *has a least fixed point* M_{P} *which is the least model of* P*.*

4 Conclusions

We have presented an overview of the abstract fixed point theory developed in [4] and its application [1] on a very broad class of logic programs, namely higher-order logic programs with negation. It is our belief that the framework of [4] can find other interesting applications, especially ones where non-monotonicity plays a prevailing role. In particular, we believe that an area that has not yet been sufficiently explored is that of non-monotonic formal grammars. In [3] it was demonstrated that the semantics of Boolean grammars can be easily captured through an extension of the framework of [4]. However, it is conceivable to have other non-monotonic extensions of formal grammars apart from the Boolean ones, such as for example macro-grammars with conjunction and negation in rule bodies. We believe that the results of [1] can be used as a yardstick in order to approach the semantics of such grammar formalisms.

References

[1] Charalambidis, Angelos, Ésik, Zoltán, and Rondogiannis, Panos. Minimum model semantics for extensional higher-order logic programming with negation. *TPLP*, 14(4-5):725–737, 2014.

[2] Ésik, Zoltán. Equational properties of stratified least fixed points (extended abstract). In de Paiva, Valeria, de Queiroz, Ruy J. G. B., Moss, Lawrence S., Leivant, Daniel, and de Oliveira, Anjolina Grisi, editors, *Logic, Language, Information, and Computation - 22nd International Workshop, WoLLIC 2015, Bloomington, IN, USA, July 20-23, 2015, Proceedings*, volume 9160 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2015.

[3] Ésik, Zoltán and Rondogiannis, Panos. Theorems on pre-fixed points of non-monotonic functions with applications in logic programming and formal grammars. In Kohlenbach, Ulrich, Barceló, Pablo, and de Queiroz, Ruy J. G. B., editors, *Logic, Language, Information, and Computation - 21st International Workshop, WoLLIC 2014, Valparaíso, Chile, September 1-4, 2014. Proceedings*, volume 8652 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2014.

[4] Ésik, Zoltán and Rondogiannis, Panos. A fixed point theorem for non-monotonic functions. *Theor. Comput. Sci.*, 574:18–38, 2015.

[5] Rondogiannis, Panos and Wadge, William W. Minimum model semantics for logic programs with negation-as-failure. *ACM Trans. Comput. Log.*, 6(2):441–467, 2005.