Theoretical Computer Science 668 (2017) 27–42

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Equivalence of two fixed-point semantics for definitional higher-order logic programs ☆,☆☆

Angelos Charalambidis, Panos Rondogiannis *, Ioanna Symeonidou

Department of Informatics & Telecommunications, University of Athens, Panepistimiopolis, 157 84 Athens, Greece

ARTICLE INFO

Article history:
Received 10 March 2016
Received in revised form 15 September 2016
Accepted 6 January 2017
Available online 18 January 2017
Communicated by U. Montanari

Keywords:
Extensional higher-order logic programming
Semantics of logic programming

ABSTRACT

Two distinct research approaches have been proposed for assigning extensional semantics to higher-order logic programming. The former approach [11] uses classical domain-theoretic tools while the latter [1] builds on a fixed-point construction defined on a syntactic instantiation of the source program. The relationships between these two approaches had not been investigated until now. In this paper we demonstrate that for a very broad class of programs, namely the class of *definitional programs* introduced by W.W. Wadge, the two approaches coincide with respect to ground atoms that involve symbols of the program. On the other hand, we argue that if existential higher-order variables are allowed to appear in the bodies of program rules, the two approaches are in general different. The results of the paper contribute to a better understanding of the semantics of higher-order logic programming.

1. Introduction

Extensional higher-order logic programming has been proposed [11,1,2,7,5,4] as a promising generalization of classical logic programming. The key idea behind this paradigm is that the predicates defined in a program essentially denote sets and therefore one can use standard extensional set theory in order to understand their meaning and reason about them. The main difference between the extensional and the more traditional *intensional* approaches to higher-order logic programming [9,6] is that the latter approaches have a much richer syntax and expressive capabilities but a non-extensional semantics.

Actually, despite the fact that only very few articles have been written regarding extensionality in higher-order logic programming, two main semantic approaches can be identified. The work described in [11,7,5,4] uses classical domain-theoretic tools in order to capture the meaning of higher-order logic programs. On the other hand, the work presented in [1,2] builds on a fixed-point construction defined on a syntactic instantiation of the source program in order to achieve an extensional semantics. Until now, the relationships between the above two approaches had not been investigated.

☆ This research was supported by the project "Handling Uncertainty in Data Intensive Applications", co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Program: THALES, Investing in knowledge society through the European Social Fund.

☆☆ A preliminary version of this paper appeared in the Proceedings of the Tenth International Workshop on Fixed Points in Computer Science, FICS 2015, Berlin, Germany, September 11–12, 2015, pp. 18–32.

* Corresponding author.
E-mail addresses: a.charalambidis@di.uoa.gr (A. Charalambidis), prondo@di.uoa.gr (P. Rondogiannis), i.symeonidou@di.uoa.gr (I. Symeonidou).

http://dx.doi.org/10.1016/j.tcs.2017.01.005
0304-3975/

In this paper we demonstrate that for a very broad class of programs, namely the class of *definitional programs* introduced by W.W. Wadge [11], the two approaches coincide with respect to ground atoms that involve symbols of the program. Intuitively, this means that for any given definitional program, the sets of true ground atoms of the program are identical under the two different semantic approaches. This result is interesting since it suggests that definitional programs are of fundamental importance for the further study of extensional higher-order logic programming. On the other hand, we argue that if we try to slightly extend the source language so as to allow existential higher-order variables in the bodies of program rules, the two approaches give different results in general; even in this extended case however, we demonstrate that under an additional assumption, the two approaches can still be shown to coincide. Overall, the results of the paper contribute to a better understanding of the semantics of higher-order logic programming and pave the road for designing a realistic extensional higher-order logic programming language.

The rest of the paper is organized as follows. Section 2 briefly introduces extensional higher-order logic programming and presents in an intuitive way the two existing approaches for assigning meaning to programs of this paradigm. Section 3 contains background material, namely the syntax of definitional programs and the formal details behind the two aforementioned semantic approaches. Section 4 demonstrates the equivalence of the two semantics for definitional programs. Section 5 investigates the case of non-definitional programs and Section 6 concludes the paper with pointers to future work.

2. Intuitive overview of the two extensional approaches

In this section we introduce extensional higher-order logic programming and present the two existing approaches for assigning meaning to programs of this paradigm. Since these two proposals were initially introduced by W.W. Wadge and M. Bezem respectively, we will refer to them as *Wadge's semantics* and *Bezem's semantics* respectively. The key idea behind both approaches is that in order to achieve an extensional semantics, one has to consider a fragment of higher-order logic programming that has a restricted syntax.

2.1. Extensional higher-order logic programming

Higher-order logic programming is an extension of classical logic programming which (roughly speaking) allows predicates to be passed as parameters of other predicates. The existing proposals for higher-order logic programming can be placed in two main categories, namely the *intensional* and the *extensional* ones. In the former category, the two most prominent languages are λProlog [9] and HiLog [6]. The latter category is much less developed: currently there exist two main proposals for extensional higher-order logic programming, namely [11] and [1,2], and only an experimental language, called HOPES,[1] has been built [5].

In an extensional language, two predicates that succeed for the same instances are considered equal. On the other hand, in an intensional language it is possible that predicates that are equal as sets will not be treated as equal. In other words, a predicate in an intensional language is more than just the set of arguments for which it is true. For example, in Hilog, two predicates are not considered equal unless their names are the same. The distinction between extensionality and intensionality has been widely discussed in the literature (see for example the detailed presentation in [6] and in [5]). In the rest of the paper we focus exclusively on the extensional approach.

The extensional approach is motivated by the following example:

Example 1. Consider a program that consists only of the following rule[2]:

```
p(Q):-Q(0),Q(1).
```

In an extensional language, predicate `p` above can be understood in purely set-theoretic terms: `p` is the set of all those sets that contain both `0` and `1`.

It should be noted that the above program is also a syntactically acceptable program of the existing intensional logic programming languages. The difference is that in an extensional language the above program has a set-theoretic semantics. □

As we are going to see shortly, extensional higher-order logic programming sacrifices some of the rich syntax of intensional higher-order logic programming in order to achieve semantic clarity.

[1] Available from https://www.github.com/acharal/hopes.

[2] For simplicity reasons, the syntax that we use in our example programs is Prolog-like. The syntax that we adopt in the next section is slightly different and more convenient for the theoretical developments that follow.