Journal of Logical and Algebraic Methods in Programming 93 (2017) 42–67

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

# Thresholded semantic framework for a fully integrated fuzzy logic language ☆

Pascual Julián-Iranzo [a,*], Ginés Moreno [b], Jaime Penabad [c]

[a] Dept. of Technologies and Information Systems, UCLM, 13071 Ciudad Real, Spain
[b] Dept. of Computing Systems, UCLM, 02071 Albacete, Spain
[c] Dept. of Mathematics, UCLM, 02071 Albacete, Spain

ARTICLE INFO

*Article history:*
Received 8 December 2016
Received in revised form 26 April 2017
Accepted 13 August 2017
Available online xxxx

*Keywords:*
Fuzzy logic programming
Similarity relations
Declarative semantics
Fixpoint semantics
Operational semantics
Correctness

ABSTRACT

This work proposes a declarative semantics based on a fuzzy variant of the classical notion of least Herbrand model for the so-called FASILL programming language (acronym of "Fuzzy Aggregators and Similarity Into a Logic Language") which has been recently designed and implemented for coping with implicit/explicit truth degree annotations, a great variety of connectives and unification by similarity. Also, we define an immediate consequence operator that allows us to give a fixpoint characterization of the least Herbrand model for a FASILL program. A remarkable aspect of our declarative and fixpoint semantics is the fact that both have been enriched with the use of thresholds thanks to the natural ability of the underlying fuzzy language for managing such constructs. Moreover, we have connected both semantics with an operational one, which also manages thresholds in an efficient way, by proving the correctness of the whole semantic framework.



## 1. Introduction

The research field of *Fuzzy Logic Programming* is devoted to introduce *fuzzy logic* concepts into *logic programming* in order to deal with vagueness in a natural way. It has provided an extensive variety of logic programming dialects along the last three decades. *Fuzzy logic languages* can be classified (among other criteria) by the emphasis placed on fuzzifying the traditional mechanisms of unification and resolution used in Logic Programming. Some approaches focus on replacing the classic unification algorithm by one based on the use of similarity/proximity relations [5,15,44,46]. Similarity/proximity relations connect the elements of a set with a certain approximation degree and serve for weakening the notion of equality and, hence, to deal with vague information. Other approaches modify the operational principle of pure logic programming to replace it by inference mechanisms based on fuzzy logic, which allow a wide variety of connectives and the use of a gradation of truth degrees (beyond the traditional values of *true* and *false*) [35,39,53].

Our research group has been involved in the development of both similarity-based logic programming systems and those that extend the resolution principle, as reveals the design of Bousi∼Prolog[1] [21,23,44], with an unification algorithm

☆ This work has been partially supported by the EU (FEDER), the State Research Agency (AEI) and the Spanish *Ministerio de Economía y Competitividad* under grant TIN2016-76843-C4-2-R (AEI/FEDER, UE).

* Corresponding author.
*E-mail address:* pascual.julian@uclm.es (P. Julián-Iranzo).

[1] Two different programming environments for the Bousi∼Prolog programming language are available at http://dectau.uclm.es/bousi/.

http://dx.doi.org/10.1016/j.jlamp.2017.08.002
2352-2208/

based on similarity/proximity relations, and the development of the FLOPER system,[2] which implements a fuzzy inference mechanism able to manage programs composed by rules richer than clauses [36,38].

Recently, we have combined these two lines of work embedding into FLOPER the *weak unification* algorithm of Bousi∼Prolog and leading to an integrated programming language named FASILL (acronym of "Fuzzy Aggregators and Similarity Into a Logic Language"). FASILL programs have three components: a set of (conditional) rules; a similarity relation, weakening equality; and a complete lattice, whose elements are interpreted as truth degrees. FASILL is publicly available: the last version of the FLOPER system which copes with similarity relations can be freely downloaded from http://dectau.uclm.es/floper/?q=sim and it can also be tested on-line through http://dectau.uclm.es/floper/?q=sim/test.

In this paper, we propose a semantic framework for the FASILL programming language. Following a scheme analogue to the one shown in [18,21,26], we introduce the concept of a fuzzy Herbrand interpretation and a notion of a fuzzy Herbrand model linked to an extended structure which embeds the meaning induced by the similarity relation component of a program. Then, we define a fuzzy immediate consequence operator as a basis for a fixpoint semantics. Also we provide an operational semantics and its *success set* semantics (i.e., the fuzzy set of ground atoms with a "successful" derivation). All these semantic concepts are characterized by a threshold, which is also called a $\lambda$-cut (see later).

Although in this study we use general techniques extracted from [26], there are remarkable differences between Bousi∼Prolog and FASILL which are important to note from the beginning:

1. From a syntactic point of view, contrary to Bousi∼Prolog, which does not allow rule annotations (in other words, Bousi∼Prolog rules are considered completely true), FASILL uses implicit annotations, by using a literal representing a lattice value (acting as a truth degree). Therefore, FASILL has two ways for expressing uncertainty: implicit annotations and similarity relations.
2. From the declarative (and fixpoint) semantics side, FASILL (because it is a heir of MALP) only considers "truth upper bounds", contrary to the option taken in Bousi∼Prolog, where only "truth lower bounds" are considered. This has a direct implication in the definition of the instrumental notion of *extended program* (see Section 4) as well as in the concepts it supports.
3. On the other hand, from the operational semantics point of view, Bousi∼Prolog is a refutational language that extends the classical SLD resolution procedure with the ability to deal with similarity-based unification. On the contrary, FASILL is not a refutational language but operates backwards using a kind of generalized *modus ponens*.

These are notable differences which make our approach a new framework without exact correspondence with the previous ones. Consequently, it is necessary to redefine and adapt the concepts derived from both Bousi∼Prolog and MALP. These adaptations, necessary for a correct integration, can be seen as one of the contributions of this work.

The outline of this paper is as follows. Firstly, in Section 2 we formally define the syntax of FASILL and we also illustrate some of its features. Afterwards, in Section 3 some notes on the implementation of the FLOPER system, supporting FASILL, are provided, along with an enumeration of practical applications which constitute a motivation for our work. Next, Section 4 details the declarative semantics of FASILL. In that section, we extend the classical concepts of Herbrand interpretation, Herbrand model and least Herbrand model of a program to the case of the new language. We perform this task in a conservative way by embedding the corresponding concepts of the multi-adjoint logic programming (MALP) framework [35] into FASILL. Section 5 provides a fixpoint characterization of the least fuzzy Herbrand model for a FASILL program. There, we determine the precise relationship between the declarative and the fixpoint semantics. In Section 6 we introduce the similarity-based unification algorithm we use and the thresholded operational semantics of FASILL. Next, Section 7 gives some correctness properties linking the declarative semantics and the operational semantics. Specifically, we prove the equivalence between the least Herbrand model of a program and its success set semantics with regard to a cut level. In Section 8 we present some pieces of related work, putting the FASILL language in context. We also show some of the differences between the FASILL programming language and other related approaches. Finally, in Section 9 we give our conclusions and future research lines.

Ending this section, we would like to say that some parts of this work have been published as immature versions in conference proceedings [17,20]. These parts have been improved and error pruned.

## 2. The FASILL language

FASILL is a first order language built upon a signature $\Sigma$, that contains the elements of a countably infinite set of variables $\mathcal{V}$, function symbols, and predicate symbols with an associated arity – usually expressed as pairs $f/n$ or $p/n$ where $n$ represents its arity –, the implication symbol ($\leftarrow$), and a wide set of other connectives $\zeta$ (t-norms, t-conorms and aggregators). The language combines the elements of $\Sigma$ as terms, atoms, rules, and formulas. A *constant* $c$ is a function symbol with arity zero. A *term* is a variable, a constant or a function symbol $f/n$ applied to $n$ terms $t_1, \ldots, t_n$, and is denoted as $f(t_1, \ldots, t_n)$. We allow the existence of a set of truth degree literals, $\Sigma_L$, as part of the signature $\Sigma$. These literals are

[2] The tool is freely accessible from the Web site http://dectau.uclm.es/floper/.