A Fuzzy Logic Programming Environment for Managing Similarity and Truth Degrees (Tool System)*

Pascual Julián-Iranzo
Department of Technologies and Information Systems
University of Castilla-La Mancha
13071 Ciudad Real (Spain)
Pascual.Julian@uclm.es

Ginés Moreno
Department of Computing Systems
University of Castilla-La Mancha
02071 Albacete (Spain)
Gines.Moreno@uclm.es

Jaime Penabad
Department of Mathematics
University of Castilla-La Mancha
02071 Albacete (Spain)
Jaime.Penabad@uclm.es

Carlos Vázquez
Department of Computing Systems
University of Castilla-La Mancha
02071 Albacete (Spain)
Carlos.Vazquez@uclm.es

FASILL (acronym of "Fuzzy Aggregators and Similarity Into a Logic Language") is a fuzzy logic programming language with implicit/explicit truth degree annotations, a great variety of connectives and unification by similarity. FASILL integrates and extends features coming from MALP (*Multi-Adjoint Logic Programming*, a fuzzy logic language with explicitly annotated rules) and Bousi∼Prolog (which uses a weak unification algorithm and is well suited for flexible query answering). Hence, it properly manages similarity and truth degrees in a single framework combining the expressive benefits of both languages. This paper presents the main features and implementations details of FASILL. Along the paper we describe its syntax and operational semantics and we give clues of the implementation of the lattice module and the similarity module, two of the main building blocks of the new programming environment which enriches the FLOPER system developed in our research group.

Keywords: Fuzzy Logic Programming, Similarity Relations, Software Tools

1 Introduction

The challenging research area of *Fuzzy Logic Programming* is devoted to introduce *fuzzy logic* concepts into *logic programming* in order to explicitly treat with uncertainty in a natural way. It has provided a wide variety of PROLOG dialects along the last three decades. *Fuzzy logic languages* can be classified (among other criteria) regarding the emphasis they assign when fuzzifying the original unification/resolution mechanisms of PROLOG. So, whereas some approaches are able to cope with similarity/proximity relations at unification time [3, 2, 16], other ones extend their operational principles (maintaining syntactic unification) for managing a wide variety of fuzzy connectives and truth degrees on rules/goals beyond the simpler case of *true* or *false* [7, 8, 13]. Our research group has been involved in both alternatives, as reveals the design of the Bousi∼Prolog language[1] [5, 6, 15], where clauses cohabit with similarity/proximity equations, and the development of the FLOPER system[2], which manages fuzzy programs composed by rules richer than clauses [9, 12]. Our current goal for fusing both worlds

*This work was supported by the EU (FEDER), and the Spanish MINECO Ministry (*Ministerio de Economía y Competitividad*) under grant TIN2013-45732-C4-2-P.

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

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

To appear in EPTCS.

$\dot{\&}_{\mathtt{P}}(x,y) \triangleq x * y$	$\dot{\vert}_{\mathtt{P}}(x,y) \triangleq x+y-xy$	*Product*
$\dot{\&}_{\mathtt{G}}(x,y) \triangleq \min(x,y)$	$\dot{\vert}_{\mathtt{G}}(x,y) \triangleq max(x,y)$	*Gödel*
$\dot{\&}_{\mathtt{L}}(x,y) \triangleq \max(0,x+y-1)$	$\dot{\vert}_{\mathtt{L}}(x,y) \triangleq \min(x+y,1)$	*Łukasiewicz*

Figure 1: Conjunctions and disjunctions in $[0,1]$ for *Product*, *Łukasiewicz*, and *Gödel* fuzzy logics

is somehow inspired by [1], but in our framework we admit a wider set of connectives inside the body of programs rules. In this paper, we give a first step in our pending task from some years ago for embedding into FLOPER the *weak unification* algorithm of Bousi∼Prolog.

FASILL is a first order language built upon a signature Σ, that contains the elements of a countably infinite set of variables $\mathscr{V}$, 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 set of connectives. The language combines the elements of Σ 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 values of a lattice L as part of the signature Σ. Therefore, a well-formed formula can be either:

- r, if $r \in L$
- $p(t_1,\ldots,t_n)$, if $t_1,\ldots,t_n$ are terms and p/n is an n-ary predicate. This formula is called *atom*. Particularly, atoms containing no variables are called *ground atoms*, and atoms built from nullary predicates are called *propositional variables*
- $\varsigma(\mathscr{F}_1,\ldots,\mathscr{F}_n)$, if $\mathscr{F}_1,\ldots,\mathscr{F}_n$ are well-formed formulas and ς is an n-ary connective with truth function $\dot{\varsigma} : L^n \to L$

Definition 1.1 (Complete lattice). *A complete lattice is a partially ordered set $(L,\leq)$ such that every subset S of L has infimum and supremum elements. Then, it is a bounded lattice, i.e., it has bottom and top elements, denoted by $\bot$ and $\top$, respectively. L is said to be the carrier set of the lattice, and $\leq$ its ordering relation.*

The lattice is equipped with a set of *connectives*[3] including

- aggregators denoted by @, whose truth functions $\dot{@}$ fulfill the boundary condition: $\dot{@}(\top,\top)=\top$, $\dot{@}(\bot,\bot)=\bot$, and monotonicity: $(x_1,y_1)\leq(x_2,y_2) \Rightarrow \dot{@}(x_1,y_1)\leq\dot{@}(x_2,y_2)$.
- t-norms and t-conorms [14] (also named conjunctions and disjunctions, that we denote by & and |, respectively) whose truth functions fulfill the following properties:
  - · Commutative: $\dot{\&}(x,y)=\dot{\&}(y,x)$ $\quad \dot{|}(x,y)=\dot{|}(y,x)$
  - · Associative: $\dot{\&}(x,\dot{\&}(y,z))=\dot{\&}(\dot{\&}(x,y),z)$ $\quad \dot{|}(x,\dot{|}(y,z))=\dot{|}(\dot{|}(x,y),z)$
  - · Identity element: $\dot{\&}(x,\top)=x$ $\quad \dot{|}(x,\bot)=x$
  - · Monotonicity in each argument: $z\leq t \Rightarrow \begin{cases} \dot{\&}(z,y)\leq\dot{\&}(t,y) & \dot{\&}(x,z)\leq\dot{\&}(x,t) \\ \dot{|}(z,y)\leq\dot{|}(t,y) & \dot{|}(x,z)\leq\dot{|}(x,t) \end{cases}$

In this paper we use the lattice $([0,1],\leq)$, where $\leq$ is the usual ordering relation on real numbers, and three sets of connectives corresponding to the fuzzy logics of Gödel, Łukasiewicz and Product, defined in Figure 1, where labels L, G and P mean respectively *Łukasiewicz logic*, *Gödel logic* and *product logic* (with different capabilities for modeling *pessimistic*, *optimistic* and *realistic scenarios*.)

[3]Here, the connectives are binary operations but we usually generalize them with an arbitrary number of arguments.

Definition 1.2 (Similarity relation)**.** *Given a domain $\mathscr{U}$ and a lattice L with fixed t-norm $\wedge$, a similarity relation $\mathscr{R}$ is a fuzzy binary relation on $\mathscr{U}$, that is a fuzzy subset on $\mathscr{U} \times \mathscr{U}$ (namely, a mapping $\mathscr{R}$: $\mathscr{U} \times \mathscr{U} \to L$), such that fulfills the following properties*[4]*:*

- *Reflexive:* $\mathscr{R}(x,x) = \top, \forall x \in \mathscr{U}$
- *Symmetric:* $\mathscr{R}(x,y) = \mathscr{R}(y,x), \forall x,y \in \mathscr{U}$
- *Transitive:* $\mathscr{R}(x,z) \geq \mathscr{R}(x,y) \wedge \mathscr{R}(y,z), \forall x,y,z \in \mathscr{U}$

Certainly, we are interested in fuzzy binary relations on a syntactic domain. We primarily define similarities on the symbols of a signature, Σ, of a first order language. This makes possible to treat as indistinguishable two syntactic symbols which are related by a similarity relation $\mathscr{R}$. Moreover, a similarity relation $\mathscr{R}$ on the alphabet of a first order language can be extended to terms by structural induction in the usual way [16]:

1. let x be a variable, $\hat{\mathscr{R}}(x,x) = \mathscr{R}(x,x) = 1$,
2. let f and g be two n-ary function symbols and let $t_1, \ldots, t_n, s_1, \ldots, s_n$ be terms,
   $\hat{\mathscr{R}}(f(t_1,\ldots,t_n), g(s_1,\ldots,s_n)) = \mathscr{R}(f,g) \wedge (\bigwedge_{i=1}^{n} \hat{\mathscr{R}}(t_i,s_i))$
3. otherwise, the approximation degree of two terms is zero.

Analogously for atomic formulas. Note that, following on, we shall not make a notational distintion between the relation $\mathscr{R}$ and its extension $\hat{\mathscr{R}}$.

Definition 1.3 (Rule)**.** *A rule has the form $A \leftarrow \mathscr{B}$, where A is an atomic formula called head and $\mathscr{B}$, called body, is a well-formed formula (ultimately built from atomic formulas $B_1, \ldots, B_n$, truth values of L and connectives)* [5]*. In particular, when the body of a rule is $r \in L$ (an element of lattice L), this rule is called fact and can be written as $A \leftarrow r$ (or simply A if $r = \top$).*

Definition 1.4 (Program)**.** *A program $\mathscr{P}$ is a tuple $\langle \Pi, \mathscr{R}, L \rangle$ where Π is a set of rules, $\mathscr{R}$ is a similarity relation between the elements of Σ, and L is a complete lattice.*

2 Operational Semantics of FASILL

Rules in a FASILL program have the same role than clauses in PROLOG (or MALP [8, 4, 11]) programs, that is, stating that a certain predicate relates some terms (the *head*) if some conditions (the *body*) hold.

As a logic language, FASILL inherits the concepts of substitution, unifier and most general unifier (*mgu*). Some of them are extended to cope with similarities. Concretely, the most general unifier is replaced by the concept of *weak most general unifier* (w.m.g.u.), following the line of Bousi∼Prolog [5]. Roughly speaking, the *weak unification algorithm* states that two *expressions* (i.e, terms or atomic formulas) $f(t_1,\ldots,t_n)$ and $g(s_1,\ldots,s_n)$ weakly unify if the root symbols f and g are close with a certain degree (i.e. $\mathscr{R}(f,g) = r > \bot$) and each of their arguments t_i and s_i weakly unify. Therefore, there is a weak unifier for two expressions even if the symbols at their roots are not syntactically equals ($f \not\equiv g$).

More technically, the weak unification algorithm we are using is a reformulation/extension of the one which appears in [16] for arbitrary complete lattices. We formalize it as a transition system supported by a similarity-based unification relation "$\Rightarrow$". The unification of the expressions $\mathscr{E}_1$ and $\mathscr{E}_2$ is

[4] For convenience, $\mathscr{R}(x,y)$, also denoted $x\mathscr{R}y$, refers both the syntactic expression (that symbolizes that the elements $x,y \in \mathscr{U}$ are related by $\mathscr{R}$) and the truth degree $\mu_{\mathscr{R}}(x,y)$, i.e., the affinity degree of the pair $(x,y) \in \mathscr{U} \times \mathscr{U}$ with the verbal predicate $\mathscr{R}$.

[5] In order to subsume the syntactic conventions of MALP, in our programs we also admit *weighted rules* with shape "$A \leftarrow_i \mathscr{B}$ *with* v", which are internally treated as "$A \leftarrow (v \&_i \mathscr{B})$" (this transformation preserves the meaning of rules as proved in [10]).

obtained by a state transformation sequence starting from an initial state $\langle G \equiv \{\mathscr{E}_1 \approx \mathscr{E}_2\}, id, \alpha_0\rangle$, where id is the identity substitution and $\alpha_0 = \top$ is the supreme of $(L, \leq)$: $\langle G, id, \alpha_0\rangle \Rightarrow \langle G1, \theta_1, \alpha_1\rangle \Rightarrow \cdots \Rightarrow \langle G_n, \theta_n, \alpha_n\rangle$. When the final state $\langle G_n, \theta_n, \alpha_n\rangle$, with $G_n = \emptyset$, is reached (i.e., the equations in the initial state have been solved), the expressions $\mathscr{E}_1$ and $\mathscr{E}_2$ are unifiable by similarity with w.m.g.u. θ_n and *unification degree* α_n. Therefore, the final state $\langle \emptyset, \theta_n, \alpha_n\rangle$ signals out the unification success. On the other hand, when expressions $\mathscr{E}_1$ and $\mathscr{E}_2$ are not unifiable, the state transformation sequence ends with failure (i.e., $G_n = Fail$).

The *similarity-based unification relation*, "$\Rightarrow$", is defined as the smallest relation derived by the following set of transition rules (where $\mathscr{V}ar(t)$ denotes the set of variables of a given term t)

$$\frac{\langle \{f(t_1,\ldots,t_n) \approx g(s_1,\ldots,s_n)\} \cup E, \theta, r_1\rangle \qquad \mathscr{R}(f,g) = r_2 > \bot}{\langle \{t_1 \approx s_1, \ldots, t_n \approx s_n\} \cup E, \theta, r_1 \wedge r_2\rangle}\ 1$$

$$\frac{\langle \{X \approx X\} \cup E, \theta, r_1\rangle}{\langle E, \theta, r_1\rangle}\ 2 \qquad \frac{\langle \{X \approx t\} \cup E, \theta, r_1\rangle \qquad X \notin \mathscr{V}ar(t)}{\langle (E)\{X/t\}, \theta\{X/t\}, r_1\rangle}\ 3$$

$$\frac{\langle \{t \approx X\} \cup E, \theta, r_1\rangle}{\langle \{X \approx t\} \cup E, \theta, r_1\rangle}\ 4 \qquad \frac{\langle \{X \approx t\} \cup E, \theta, r_1\rangle \qquad X \in \mathscr{V}ar(t)}{\langle Fail, \theta, r_1\rangle}\ 5$$

$$\frac{\langle \{f(t_1,\ldots,t_n) \approx g(s_1,\ldots,s_n)\} \cup E, \theta, r_1\rangle \qquad \mathscr{R}(f,g) = \bot}{\langle Fail, \theta, r_1\rangle}\ 6$$

Rule 1 decomposes two expressions and annotates the relation between the function (or predicate) symbols at their root. The second rule eliminates spurious information and the fourth rule interchanges the position of the symbols to be coped by other rules. The third and fifth rules perform an occur check of variable X in a term t. In case of success, it generates a substitution $\{X/t\}$; otherwise the algorithm ends with failure. It can also end with failure if the relation between function (or predicate) symbols in $\mathscr{R}$ is $\bot$, as stated by Rule 6.

Usually, given two expressions $\mathscr{E}_1$ and $\mathscr{E}_2$, if there is a successful transition sequence, $\langle \{\mathscr{E}_1 \approx \mathscr{E}_2\}, id, \top\rangle \Rightarrow^\star \langle \emptyset, \theta, r\rangle$, then we write that $wmgu(\mathscr{E}_1, \mathscr{E}_2) = \langle \theta, r\rangle$, being θ the *weak most general unifier* of $\mathscr{E}_1$ and $\mathscr{E}_2$, and r is their *unification degree*.

Finally note that, in general, a w.m.g.u. of two expressions $\mathscr{E}_1$ and $\mathscr{E}_2$ is not unique [16]. Certainly, the weak unification algorithm only computes a representative of a w.m.g.u. class, in the sense that, if $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ is a w.m.g.u., with degree β, then, by definition, any substitution $\theta' = \{x_1/s_1, \ldots, x_n/s_n\}$, satisfying $\mathscr{R}(s_i, t_i) > \bot$, for any $1 \leq i \leq n$, is also a w.m.g.u. with approximation degree $\beta' = \beta \wedge (\bigwedge_1^n \mathscr{R}(s_i, t_i))$, where "$\wedge$" is a selected t-norm. However, observe that, the w.m.g.u. representative computed by the weak unification algorithm is one with an approximation degree equal or greater than other w.m.g.u. As in the case of the classical syntactic unification algorithm, our algorithm always terminates returning a success or a failure.

In order to describe the procedural semantics of the FASILL language, in the following we denote by $\mathscr{C}[A]$ a formula where A is a sub-expression (usually an atom) which occurs in the –possibly empty– context $\mathscr{C}[]$ whereas $\mathscr{C}[A/A']$ means the replacement of A by A' in the context $\mathscr{C}[]$. Moreover, $\mathscr{V}ar(s)$ denotes the set of distinct variables occurring in the syntactic object s and $\theta[\mathscr{V}ar(s)]$ refers to the substitution obtained from θ by restricting its domain to $\mathscr{V}ar(s)$. In the next definition, we always consider that A is the selected atom in a goal $\mathscr{Q}$ and L is the complete lattice associated to Π.

Definition 2.1 (Computational Step)**.** *Let $\mathscr{Q}$ be a goal and let σ be a substitution. The pair $\langle \mathscr{Q}; \sigma\rangle$ is a* state*. Given a program $\langle \Pi, \mathscr{R}, L\rangle$ and a t-norm $\wedge$ in L, a* computation *is formalized as a state transition*

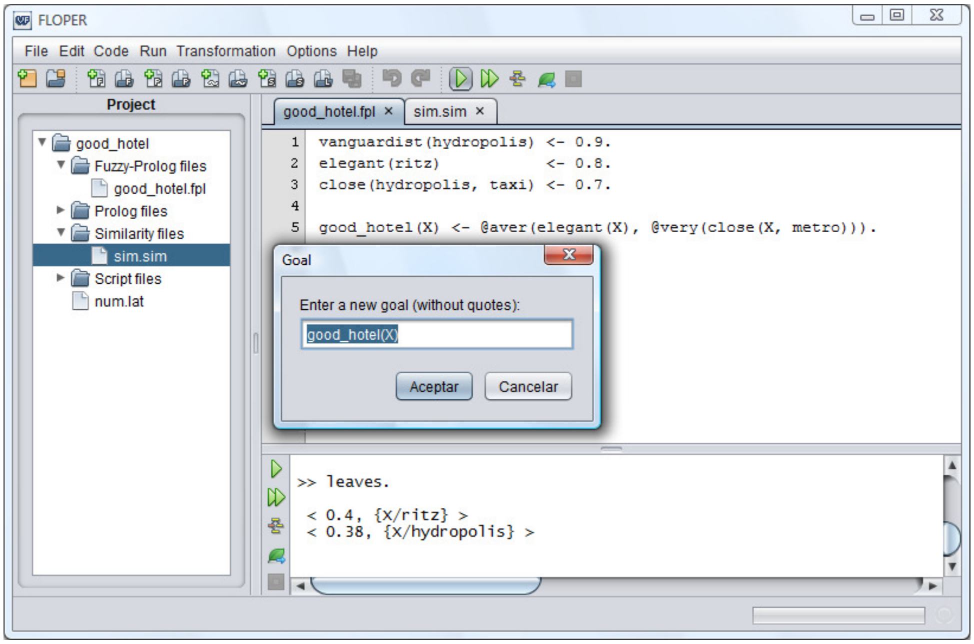

Figure 2: Screen-shot of a work session with FLOPER managing a FASILL program

system, whose transition relation $\leadsto$ *is the smallest relation satisfying these rules:*

1) Successful step *(denoted as* $\overset{SS}{\leadsto}$*):*

$$\frac{\langle \mathscr{Q}[A], \sigma \rangle \qquad A' \leftarrow \mathscr{B} \in \Pi \qquad wmgu(A, A') = \langle \theta, r \rangle}{\langle \mathscr{Q}[A/\mathscr{B} \wedge r]\theta, \sigma\theta \rangle} \text{ SS}$$

2) Failure step *(denoted as* $\overset{FS}{\leadsto}$*):*

$$\frac{\langle \mathscr{Q}[A], \sigma \rangle \qquad \nexists A' \leftarrow \mathscr{B} \in \Pi : wmgu(A, A') = \langle \theta, r \rangle, r > \bot}{\langle \mathscr{Q}[A/\bot], \sigma \rangle} \text{ FS}$$

3) Interpretive step *(denoted as* $\overset{IS}{\leadsto}$*):*

$$\frac{\langle \mathscr{Q}[@(r_1, \ldots, r_n)]; \sigma \rangle \qquad \dot{@}(r_1, \ldots, r_n) = r_{n+1}}{\langle \mathscr{Q}[@(r_1, \ldots, r_n)/r_{n+1}]; \sigma \rangle} \text{ IS}$$

A *derivation* is a sequence of arbitrary lenght $\langle \mathscr{Q}; id \rangle \leadsto^* \langle \mathscr{Q}'; \sigma \rangle$. As usual, rules are renamed apart. When $\mathscr{Q}' = r \in L$, the state $\langle r; \sigma \rangle$ is called a *fuzzy computed answer* (f.c.a.) for that derivation.

3 Implementation of FASILL in FLOPER

During the last years we have developed the FLOPER tool, initially intended for manipulating MALP programs[6]. In its current development state, FLOPER has been equipped with new features in order to

[6] The MALP language is nowadays fully subsumed by the new FASILL language just introduced in this paper, since, given a FASILL program $\mathscr{P} = \langle \Pi, \mathscr{R}, L \rangle$, if $\mathscr{R}$ is the identity relation (that is, the one where each element of a signature Σ is only similar to itself, with the maximum similarity degree) and L is a complete lattice also containing *adjoint pairs* [8], then $\mathscr{P}$ is a MALP program too.

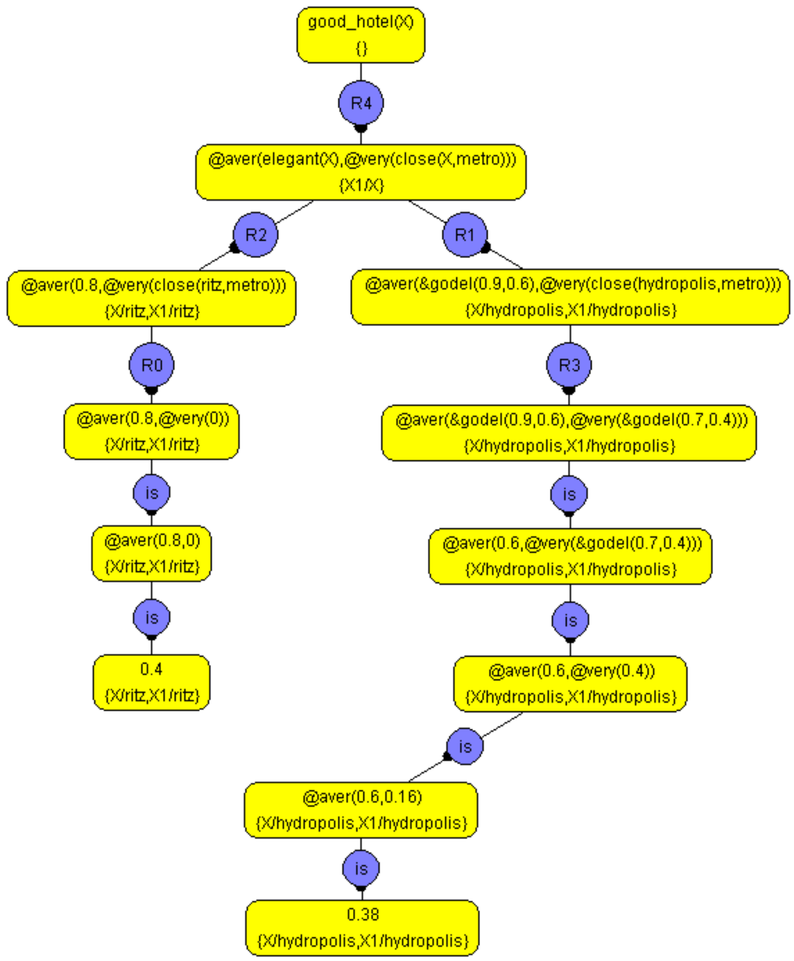

Figure 3: An execution tree as shown by the FLOPER system

cope with more expressive languages and, in particular, with FASILL (that is freely accessible in its url `http://dectau.uclm.es/floper/?q=sim` where it is possible to test/download the new prototype incorporating the management of similarity relations. In this section we briefly describe the main features of this tool before presenting the novelties introduced in this work.

FLOPER has been implemented in Sicstus Prolog v.3.12.5 (rounding about 1083 lines of code, where our last update supposes approximately a 30% of the final code) and it has been recently equipped with a graphical interface written in Java (circa 2000 lines of code). More detailed, the FLOPER system consists in a ".jar" java program that runs the graphical interface. This ".jar" program calls a ".pl" file containing the two main independent blocks: 1) the Parsing block parses FASILL files into two kinds of prolog code (a high level platform-independent Prolog program and a set of facts to be used by FLOPER), and 2) the Procedural block performs the evaluation of a goal against the program, implementing the procedural semantics previously described. This code is completed with a configuration file indicating the location of the Prolog interpreter as well as some other data.

When the graphical interface is executed, it offers a menu with a set of commands grouped in four submenus:

- "Program Menu": includes options for *parsing* a FASILL program from a file with extension "`.fpl`", *saving* the generated PROLOG code to a "`.pl`" file, *loading/parsing* a pure PROLOG program, *listing* the rules of the parsed program and *cleaning* the database.
- "Lattice Menu": allows the user to change and show the lattice (implemented in PROLOG) associated to a fuzzy program through options *lat* and *show*, respectively.
- "Similarity Menu": option *sim* allows the user to load a similarity file (with extension "`.sim`", and whose syntax is detailed further in the Similarity Module subsection) and *tnorm* sets the conjunction to be used in the transitive closure of the relation.
- "Goal Menu": by choosing option *intro* the user introduces the goal to be evaluated. Option *tree* draws the execution tree for that goal whereas *leaves* only shows the set of fuzzy computed answer contained on it, and *depth* is used for fixing its maximum depth.

The syntax of FASILL presented in Section 1 is easily translated to be written by a computer. As usual in logic languages, variables are written as identifiers beginning by an upper case character or an underscore "_", while function and predicate symbols are expressed with identifiers beginning by a lower case character, and numbers are literals. Terms and atoms have the usual syntax (the function or predicate symbol, if no nullary, is followed by its arguments between parentheses and separated by a colon). Connectives are labeled with their name immediately after. The implication symbol is written as "`<-`", and each rule ends with a dot. Additionally it is possible to include pure PROLOG expressions inside the body of a rule by encapsuling them between curly brackets "{}", and PROLOG clauses together with FASILL rules between the dollar symbol "$".

In the recent years we have equipped the tool with a graphical interface (written in Java) for allowing a friendship interaction with the user, as seen in Figure 2. The graphical interface shows three areas. The leftmost one draws the project tree (grouping each category of file into its own directory). In the right part, the upper area displays the selected file of the tree and the lower one shows the code and the solutions of executing a goal. This interface groups files into projects which include a set of *fuzzy* files (`.fpl`), PROLOG files (`.pl`), *similarity* files (`.sim`), *script* files -containing a list of commands to be executed consecutively- (`.vfs`) and just one lattice file (`.lat`). When executing a goal, the tool considers the whole program merged from the set of files, thus obtaining only one fuzzy program, one similarity relation, one lattice and one PROLOG file.

The lattice module. Lattices are described in a `.lat` file using a language that is a subset of PROLOG where the definition of some predicates are mandatory, and the definition of aggregations follows a certain syntax. The mandatory predicates are `member/1`, that identifies the elements of the lattice, `bot/1` and `top/1`, that stand for the infimum and supremum elements of the lattice, and `leq/2`, that implements the ordering relation. Predicate `members/1`, that returns in a list all the elements of the lattice, is only required if it is finite. Connectives are defined as predicates whose meaning is given by a number of clauses. The name of the predicate has the form `and`*_label*, `or`*_label* or `agr`*_label* whether it implements a conjunction, a disjunction or an aggregator, where *label* is an identifier of that particular connective (this way one can define several conjunctions, disjunctions and other kind of aggregators instead of only one). The arity of the predicate is $n+1$, where n is the arity of the connective that it implements, so its last parameter is a variable to be unified with the value resulting of its evaluation.

$$\left.\begin{array}{l} ?-\ agr\_label(r_1,\ldots,r_n,R). \\ R=r. \end{array}\right\}\texttt{if } @_{label}(r_1,\ldots,r_n)=r$$

Example 1. *For instance, the following clauses show the* PROLOG *program modeling the lattice of the real interval* $[0,1]$ *with the usual ordering relation and connectives (conjunction and disjunction of the* Product logic*, as well as the average aggregator):*

```
member(X):- number(X), 0=<X, X=<1.                  leq(X,Y):- X=<Y.
and_prod(X,Y,Z) :- Z is X*Y.                        bot(0).
or_prod(X,Y,Z)  :- U1 is X*Y, U2 is X+Y, Z is U2-U1.    top(1).
agr_aver(X,Y,Z) :- U1 is X+Y, Z is U1/2.
```

The similarity module. We describe now the main novelty performed in the tool, that is the ability to take into account a similarity relation. The similarity relation $\mathscr{R}$ is loaded from a file with extension `.sim` through option *sim*. The relation is represented following a concrete syntax:

$\langle Relation\rangle$ ::= $\langle Sim\rangle$ $\langle Relation\rangle$ | $\langle Sim\rangle$
$\langle Sim\rangle$ ::= $\langle Id_f\rangle$['/' $\langle Int_n\rangle$] '$\sim$' $\langle Id_g\rangle$['/' $\langle Int_n\rangle$] '=' $\langle r\rangle$ '.' | '$\sim$' 'tnorm' '=' $\langle tnorm\rangle$

The *Sim* option parses expressions like "$f \sim g = r$", where f and g are propositional variables or constants and r is an element of L. It also copes with expressions including arities, like "$f/n \sim g/n = r$" (then, f and g are function or predicate symbols). In this case, both arities have to be the same. It is also possible to explicit, through a line like "$\sim tnorm = \langle label\rangle$" the conjunction to be used further in the construction of the transitive closure of the relation. Internally FLOPER stores each relation as a fact r in an ad hoc module *sim* as $r(f/n, g/n, r)$, where $n = 0$ if it has not been specified (that is, the symbol is considered as a constant). The `.sim` file contains only a small set of similarity equations that FLOPER completes by performing the reflexive, symmetric and transitive closure. The first one simply consists of the assertion of the fact $r(A,A,\top)$. The symmetric closure produces, for each $r(a,b,r)$, the assertion of its symmetric entry $r(b,a,r)$ if there is not already some $r(b,a,r')$ where $r \leq r'$ (in this case $r(a,b,r)$ will be rewritten as $r(a,b,r')$ when considering $r(b,a,r)$). The transitive closure is computed by the next algorithm[7], where $\wedge$ stands for the conjunction specified by the directive "*tnorm*", and "*assert*" and "*retract*" are self-explainable and defined as in PROLOG:

Transitive Closure
forall r(A,B,r_1) **in** `sim`
  forall r(B,C,r_2) **in** `sim`
    $r = r_1 \wedge r_2$
    if r(A,C,r') **in** `sim` **and** $r' < r$
      retract r(A,C,r') **from** `sim`
      retract r(C,A,r') **from** `sim`
    end if
    if r(A,C,r') **not in** `sim`
      assert r(A,C,r) **in** `sim`
      assert r(C,A,r) **in** `sim`
    end if
  end forall
end forall

It is important to note that, it is not relevant if the user provides (apparently) inconsistent similarity equations, since FLOPER automatically changes the user values by the appropriate approximation de-

[7] It is important to note that this algorithm must be executed right after performing the symmetric, reflexive closure.

grees in order to preserve the properties of a similarity. For instance, if a user provides a set of equations such as, $a \sim b = 0.8$, $b \sim c = 0.6$ and $a \sim c = 0.3$, after the application of our algorithm for the construction of a similarity, results in the set of equations $a \sim b = 0.8$, $b \sim c = 0.6$ and $a \sim c = 0.6$, which positively preserves the transitive property[8].

Example 2. *In order to illustrate the enhanced expressiveness of* FASILL*, consider the program* $\langle \Pi, \mathscr{R}, L\rangle$ *(where L is the real interval* $[0,1]$ *and* $\leq$ *is the usual ordering relation on real numbers), that models the concept of* good hotel*, that is, an elegant hotel that is very close to a metro entrance, as seen in Figure 2. Here, we use an average aggregator defined as* $\dot{@}_{avg}(x,\ y) \triangleq (x+y)/2$*, whereas* very *is a linguistic modifier implemented as well as an aggregator (with arity 1) with truth function* $\dot{@}_{very}\ x \triangleq x^2$*. The similarity relation* $\mathscr{R}$ *states that* elegant *is similar to* vanguardist*, and* metro *to* bus *and (by transitivity) to* taxi*:*

```
~tnorm  = godel                              metro ~ bus  = 0.5.
elegant/1 ~ vanguardist/1  = 0.6.            bus   ~ taxi = 0.4.
```

We also state that the t-norm to be used in the transitive closure is the conjunction of Gödel (i.e., the infimum between two elements). With respect to this program (the set of rules from Figure 2, the lattice [0,1] with the usual ordering relation and the similarity relation just described before), the goal `good_hotel(X)` *produces two fuzzy computed answers:* `<0.4, X/ritz>` *and* `<0.38, X/hydropolis>`*. Each one corresponds to the leaves of the tree*[9] *depicted in Figure 2. Note that for reaching these solutions, a failure step was performed in the derivation of the left-most branch, whereas in the right-most one (and this is the crucial novelty w.r.t. previous versions of the* FLOPER *tool) there exist two successful steps exploiting the similarity relation which firstly relates* `elegant` *and* `vanguardist` *and secondly (by transitivity)* `metro` *and* `taxi` *when solving atom* $close(hydropolis, metro)$*, which illustrates the flexibility of our system.*

Ending this section, it is worthy to say that our approach differs from the one presented in [1] since they employ a combination of transformation techniques to first extract the definition of a predicate "$\sim$", simulating weak unification in terms of a set of complex program rules that extends the original program. Finally, this predicate "$\sim$" is reduced to a built-in proximity/similarity unification operator (in this case not implemented by rules and very close to the implementation of our weak unification algorithm) that highly improves the efficiency of their previos programming systems.

4 Conclusions and Future Work

This work was concerned with the last enrichment performed on our FLOPER system to cope with similarity relations. In [5, 4, 11] we provide some advances in the design of declarative semantics and/or correctness properties regarding the development of fuzzy logic languages dealing with similarity/proximity relations (Bousi$\sim$Prolog) or highly expressive lattices modeling truth degrees (MALP). As a matter of future work we want to establish that analogous –but reinforced– features also hold in the twofold integrated fuzzy language FASILL whose syntax, procedural principle (based on weak -instead of syntactic- unification for managing similarity relations) and implementation details were described along this paper.

[8] For simplicity we have omitted the equations obtained during the construction of the reflexive, symmetric closure.

[9] Each state contains its corresponding goal and substitution components and they are drawn inside yellow ovals. Computational steps, colored in blue, are labeled with the program rule they exploit in the case of *successful* steps or the annotation "R0" in the case of *failure* steps (observe that, "R0" is a simple notation and do not correspond with any existing rule). Finally, the blue circles annotated with the word "is", correspond to *interpretive* steps.

References

[1] R. Caballero, M. Rodríguez-Artalejo & C. Romero-Díaz (2014): *A Transformation-based implementation for CLP with qualification and proximity*. Theory and Practice of Logic Programming 14(1), pp. 1–63. Available at http://dx.doi.org/10.1017/S1471068412000014.

[2] F. A. Fontana (2002): *Likelog for flexible query answering*. Soft Computing 7(2), pp. 107–114.

[3] F. Formato, G. Gerla & M.I. Sessa (2000): *Similarity-based Unification.* Fundamenta Informaticae 41(4), pp. 393–414.

[4] P. Julián-Iranzo, G. Moreno & J. Penabad (2009): *On the Declarative Semantics of Multi-Adjoint Logic Programs*. In: Proc. of 10th Intl Work-Conference on Artificial Neural Networks (Part I), IWANN'09, Springer Verlag, LNCS 5517, pp. 253–260. Available at http://dx.doi.org/10.1007/978-3-642-02478-8_32.

[5] P. Julián-Iranzo & C. Rubio-Manzano (2009): *A declarative semantics for Bousi∼Prolog*. In: Proc. of 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'09, ACM, pp. 149–160. Available at http://doi.acm.org/10.1145/1599410.1599430.

[6] P. Julián-Iranzo & C. Rubio-Manzano (2010): *An efficient fuzzy unification method and its implementation into the Bousi∼Prolog system*. In: Proc. of the 2010 IEEE International Conference on Fuzzy Systems, IEEE, pp. 1–8. Available at http://dx.doi.org/10.1109/FUZZY.2010.5584193.

[7] M. Kifer & V.S. Subrahmanian (1992): *Theory of generalized annotated logic programming and its applications.* Journal of Logic Programming 12, pp. 335–367.

[8] J. Medina, M. Ojeda-Aciego & P. Vojtáš (2004): *Similarity-based Unification: a multi-adjoint approach*. Fuzzy Sets and Systems 146, pp. 43–62.

[9] P.J. Morcillo, G. Moreno, J. Penabad & C. Vázquez (2010): *A Practical Management of Fuzzy Truth Degrees using FLOPER*. In: Proc. of 4nd. Intl. Symposium on Rule Interchange and Applications, RuleML'10, Springer Verlag, LNCS 6403, pp. 20–34. Available at http://dx.doi.org/10.1007/978-3-642-16289-3_4.

[10] G. Moreno, J. Penabad & C. Vázquez (2013): *Relaxing the Role of Adjoint Pairs in Multi-adjoint Logic Programming*. In I. Hamilton & J. Vigo-Aguiar, editors: Proc. of 13th International Conference on Mathematical Methods in Science and Engineering, CMMSE'13 (Volume III), pp. 1156–1167.

[11] G. Moreno, J. Penabad & C. Vázquez (2014): *Fuzzy Sets for a Declarative Description of Multi-adjoint Logic Programming*. In: Proc. of the 9th International Conference on Rough Sets and Current Trends in Soft Computing, RSCTC 2014, Springer Verlag, LNCS 8536, pp. 71–82. Available at http://dx.doi.org/10.1007/978-3-319-08644-6_7.

[12] G. Moreno & C. Vázquez (2014): *Fuzzy Logic Programming in Action with FLOPER*. Journal of Software Engineering and Applications 7, pp. 237–298. Available at http://dx.doi.org/10.4236/jsea.2014.74028.

[13] S. Muñoz-Hernández, V. Pablos Ceruelo & H. Strass (2011): *RFuzzy: Syntax, semantics and implementation details of a simple and expressive fuzzy tool over Prolog*. Information Sciences 181(10), pp. 1951–1970. Available at http://dx.doi.org/10.1016/j.ins.2010.07.033.

[14] H. T. Nguyen & E. A. Walker (2006): *A First Course in Fuzzy Logic*. Chatman & Hall, Boca Ratón, Florida. Third edition.

[15] C. Rubio-Manzano & P. Julián-Iranzo (2014): *A Fuzzy linguistic prolog and its applications*. Journal of Intelligent and Fuzzy Systems 26(3), pp. 1503–1516. Available at http://dx.doi.org/10.3233/IFS-130834.

[16] M.I. Sessa (2002): *Approximate reasoning by similarity-based SLD resolution.* Theoretical Computer Science 275(1-2), pp. 389–426. Available at http://dx.doi.org/10.1016/S0304-3975(01)00188-8.