Seeking a Safe and Efficient Similarity-based Unfolding Rule*

Pascual Julián-Iranzo[a], Ginés Moreno[b], José Antonio Riaza[b]

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

Abstract

The unfolding transformation has been widely used in many declarative frameworks for improving the efficiency of programs. Inspired by our previous experiences in fuzzy logic languages not dealing with similarity relations, in this work we try to adapt such operation to the so-called FASILL language (acronym of "Fuzzy Aggregators and Similarity Into a Logic Language") which has been developed in our research group for coping with implicit/explicit truth degree annotations, a great variety of connectives and unification by similarity. The traditional unfolding transformation is based on the application of unifiers on the heads and computational steps on the bodies of program rules. However, when considering similarity relations, the premature generation and application of weak (similarity-based) unifiers at unfolding time could destroy the correctness (i.e., the soundness and completeness) of the transformation. In this paper we study how to avoid this risk by compiling what we call *similarity constraints* on transformed rules, whose further evaluation is delayed at running time. Moreover, our technique minimizes the size and number of occurrences of such constructs in transformed programs to gain efficiency while preserving semantics.

Keywords: Fuzzy Logic Programming, Similarity Relations, Unfolding

*This work was supported by the State Research Agency (AEI) of the Spanish Ministry of Science and Innovation under grant PID2019-104735RB-C42 (SAFER)

Email address: {Pascual.Julian, Gines.Moreno, JoseAntonio.Riaza}@uclm.es (José Antonio Riaza)

1. Introduction

The challenging research area of *Fuzzy Logic Programming* is devoted to introduce *fuzzy logic* concepts into *logic programming* in order to explicitly deal with vagueness and uncertainty in a natural way. It has provided an extensive variety of Prolog dialects along the last three decades. *Fuzzy logic languages* can be classified (among other criteria) according to the emphasis they assign to fuzzifying the original unification/resolution mechanisms of Prolog. Whereas some approaches are able to cope with similarity/proximity relations at unification time [11, 5], 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]. We adopt a unifying approach, where complete lattices of truth degrees cohabit with similarity relations, leading to the design of the FASILL language [2, 3], for which we have developed the FASILL system that has been used for coding real-word applications (see [1]) and can be also executed online via `https://dectau.uclm.es/fasill/sandbox`. Although FASILL can work with any complete lattice –with top ($\top$) and bottom ($\bot$) elements–, for the sake of simplicity, in the examples, we use the lattice of real numbers $([0,1], \leq)$ for modeling truth degrees and, in particular, we also assume the presence of fuzzy aggregators such as the arithmetical average whose truth function is defined as $@_{aver}(x,y) = \frac{x+y}{2}$, and $@_{very}(x) = x^2$, as well as fuzzy connectives like *Gödel's conjunction* defined as $x \wedge y = min(x,y)$.

On the other hand, *unfolding* is a well-known, widely used, semantics-preserving program transformation rule which is able to improve programs, generating more efficient code. The unfolding transformation traditionally considered in classical logic programming consists in the replacement of a program clause C by the set of clauses obtained after applying a symbolic computation step, in all its possible forms, on the body of C [12, 9]. Although in [4] we successfully adapted such operation to fuzzy logic programs dealing with lattices of truth degrees and syntactic unification, there are not precedents coping with similarity relations, which motivates the present work.

The structure of this paper is as follows. After presenting in Section 2 the FASILL syntax and operational semantics, in Section 3 we provide a safe definition of similarity-based unfolding which goes beyond the classical transformation style by delaying some critical unification problems thanks to the use of *similarity constraints*. Such definition is next improved in Section

4 by reducing the size and number of this kind of guards in the bodies of unfolded rules. Finally, in Section 5 we conclude and present some lines of future work.

2. The Fuzzy Logic Language FASILL

FASILL is a first order language built upon a signature Σ (i.e., a disjoint union of a set of variables, $\mathcal{V}$, and sets of function and predicate symbols with an associated arity) and a wide set of connectives: t-norms (&); t-conorms (|); aggregators (@); and implication symbols ($\leftarrow$).

The language combines the elements of the algebraic part of Σ (i.e, variables and function symbols) to build *terms* in the usual form. An *atomic formula* (or atom) is a n-ary predicate symbol applied to n terms or a literal constant $\alpha \in \Sigma_L$, where Σ_L is a set of constants which are the syntactic representation of the truth values in a complete lattice L. In this paper, we use the name "*expression*" to refer either to a term or an atom interchangeably.

Definition 1 (Similarity relation). *[2] Given a domain $\mathcal{U}$ and a lattice L with a fixed t-norm $\wedge$, a* similarity relation *$\mathcal{R}$ is a fuzzy binary relation on $\mathcal{U}$,*[1] *fulfilling the the following properties: reflexive ($\mathcal{R}(x,x) = \top, \forall x \in \mathcal{U}$), symmetric ($\mathcal{R}(x,y) = \mathcal{R}(y,x), \forall x,y \in \mathcal{U}$) and transitive ($\mathcal{R}(x,z) \geq \mathcal{R}(x,y) \wedge \mathcal{R}(y,z), \forall x,y,z \in \mathcal{U}$).*

Definition 2 (Rule and Program). *[2] A* rule *has the form $A \leftarrow \mathcal{B}$, where A is an atomic formula on a signature Σ, called the* head*, and $\mathcal{B}$ is a well-formed formula on Σ and Σ_L (i.e., built from atomic formulas $B_1, \ldots, B_n$ on Σ, constants of Σ_L and connectives –excluding implications–) which is called the* body*. A* FASILL program*, $\mathcal{P}$, is a tuple $\langle \Pi, \mathcal{R}, L \rangle$ where Π is a set of rules, $\mathcal{R}$ is a similarity relation whose domain is the signature Σ, and L is a complete lattice.*

Example 1. *In this paper we will deal with the following program $\mathcal{P} = \langle \Pi, \mathcal{R}, L \rangle$ based on lattice $L = ([0,1], \leq)$ and the set of rules Π, and similarity relation $\mathcal{R}$, represented as a matrix on $\mathcal{U} = \{vanguardist, elegant, metro,$*

[1] A *fuzzy binary relation* on $\mathcal{U}$ is a fuzzy subset on $\mathcal{U} \times \mathcal{U}$ (i.e., a mapping $\mathcal{R} : \mathcal{U} \times \mathcal{U} \rightarrow L$).

taxi, *bus*}*) below:*

$$\Pi = \begin{cases} R_1: & vanguardist(hydropolis) & \leftarrow 0.9 \\ R_2: & elegant(ritz) & \leftarrow 0.8 \\ R_3: & close(hydropolis, taxi) & \leftarrow 0.7 \\ R_4: & good\_hotel(x) & \leftarrow @_{aver}(elegant(x), \\ & & \quad @_{very}(close(x, metro))) \end{cases}$$

$\mathcal{R}$	*vanguardist*	*elegant*	*metro*	*taxi*	*bus*
vanguardist	*1*	*0.6*	*0*	*0*	*0*
elegant	*0.6*	*1*	*0*	*0*	*0*
metro	*0*	*0*	*1*	*0.4*	*0.5*
taxi	*0*	*0*	*0.4*	*1*	*0.4*
bus	*0*	*0*	*0.5*	*0.4*	*1*

It is easy to check that $\mathcal{R}$ fulfills the reflexive, symmetric and transitive properties. Particularly, we have that: $\mathcal{R}(taxi, metro) \geq \mathcal{R}(metro, bus) \wedge \mathcal{R}(bus, taxi) = min(0.5, 0.4) = 0.4$. Moreover, the natural extension of $\mathcal{R}$ from symbols to expressions, denoted as $\hat{\mathcal{R}}$, determines that $elegant(taxi)$ and $vanguardist(metro)$ are similar atoms, since:

$\hat{\mathcal{R}}(elegant(taxi), vanguardist(metro)) =$
$\mathcal{R}(elegant, vanguardist) \wedge \hat{\mathcal{R}}(taxi, metro) =$
$0.6 \wedge \mathcal{R}(taxi, metro) = 0.6 \wedge 0.4 = min(0.6, 0.4) = 0.4.$ ■

Instead of syntactic unification, and similarly to other fuzzy languages [6, 10], FASILL uses weak unification for coping with similarity relations [11, 5]. In essence, the *weak most general unifier* of level λ of two terms t and s, denoted by $wmgu(t, s) = \langle \sigma, r \rangle$, is a unifier substitution σ such that, for any other unifier θ, there exists a substitution δ such that, $\mathcal{R}(x\sigma\delta, x\theta) \geq \lambda$. So, w.r.t. the previous example: $wmgu(elegant(taxi), vanguardist(metro)) = \langle id, 0.4 \rangle$, being id the empty substitution, whereas $wmgu(vanguardist(x), elegant(taxi)) = \langle \{x/taxi\}, 0.6 \rangle$.

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

domain to $\mathcal{V}ar(s)$. In the next definition, we always consider that A is the selected atom in a goal $\mathcal{Q}$ and L is the complete lattice associated to Π.

Definition 3 (Computational Step). *[2] Let $\mathcal{Q}$ be a goal and let σ be a substitution. The pair $\langle \mathcal{Q}; \sigma \rangle$ is a* state. *Given a program $\langle \Pi, \mathcal{R}, L \rangle$ and a t-norm $\wedge$ in L (also associated to $\mathcal{R}$), a* computation *is formalized as a state transition system, whose transition relation $\rightsquigarrow$ is the smallest relation satisfying these rules:*

1. Successful step *(denoted as $\overset{SS}{\rightsquigarrow}$):* $\langle \mathcal{Q}[A], \sigma \rangle \overset{SS}{\rightsquigarrow} \langle \mathcal{Q}[A/\mathcal{B} \wedge r]\theta, \sigma\theta \rangle$, *if $A' \leftarrow \mathcal{B} \in \Pi$ and $wmgu(A, A') = \langle \theta, r \rangle$.*
2. Failure step *(denoted as $\overset{FS}{\rightsquigarrow}$):* $\langle \mathcal{Q}[A], \sigma \rangle \overset{FS}{\rightsquigarrow} \langle \mathcal{Q}[A/\bot], \sigma \rangle$, *if $\nexists A' \leftarrow \mathcal{B} \in \Pi : wmgu(A, A') = \langle \theta, r \rangle$ and $r > \bot$.*
3. Interpretive step *(denoted as $\overset{IS}{\rightsquigarrow}$):* $\langle \mathcal{Q}[@(r_1, \ldots, r_n)]; \sigma \rangle \overset{IS}{\rightsquigarrow} \langle \mathcal{Q}[@(r_1, \ldots, r_n)/r_{n+1}]; \sigma \rangle$, *if $\dot{@}(r_1, \ldots, r_n) = r_{n+1}$, where $\dot{@}$ is interpretation of $@$.*

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

Example 2. *Let $\mathcal{P} = \langle \Pi, \mathcal{R}, L \rangle$ be the program from Example 1. It is possible to perform this derivation with fuzzy computed answer $\langle 0, 4, \{x/ritz\} \rangle$ for $\mathcal{P}$ and goal $\mathcal{Q} = good\_hotel(x)$:*

$$
\begin{array}{lll}
D1: & \langle good\_hotel(x), id \rangle & \overset{SS}{\rightsquigarrow}{}^{R4} \\
 & \langle @_{aver}(elegant(x), @_{very}(close(x, metro))), \{x_1/x\} \rangle & \overset{SS}{\rightsquigarrow}{}^{R2} \\
 & \langle @_{aver}(0.8, @_{very}(close(ritz, metro))), \{x_1/ritz, x/ritz\} \rangle & \overset{FS}{\rightsquigarrow} \\
 & \langle @_{aver}(0.8, @_{very}(0)), \{x_1/ritz, x/ritz\} \rangle & \overset{IS}{\rightsquigarrow} \\
 & \langle @_{aver}(0.8, 0), \{x_1/ritz, x/ritz\} \rangle & \overset{IS}{\rightsquigarrow} \\
 & \langle 0.4, \{x_1/ritz, x/ritz\} \rangle &
\end{array}
$$

Apart from this derivation, there exists a second one ending with the alternative f.c.a. $\langle 0.38, \{x/hydropolis\} \rangle$ associated to the same goal. ■

3. Towards an Unfolding Rule for FASILL Programs

Let us start this section by showing the problems arisen when unfolding FASILL programs, if we simply do a naive adaptation of the unfolding rule defined in [4] which does not treat with similarity relations.

Definition 4 (Similarity-based Unfolding –naive version–). *For a* FASILL *program* $\mathcal{P} = \langle \Pi, \mathcal{R}, L \rangle$ *and a program rule with no empty body* $R : A \leftarrow \mathcal{B} \in \Pi$*, the similarity-based unfolding of* $\mathcal{P}$ *w.r.t. the rule* R *is the new program*

$$\mathcal{P}' = (\mathcal{P} - \{R\}) \cup \mathcal{U}$$

where $\mathcal{U} = \{A\sigma \leftarrow \mathcal{B}' \mid \langle \mathcal{B}; id \rangle \rightsquigarrow \langle \mathcal{B}'; \sigma \rangle\}$.

The following examples illustrate how to unfold rules and FASILL programs.

Example 3. *Given the* FASILL *program* $\mathcal{P} = \{R_1 : p(x) \leftarrow @_{aver}(q(x), 1); R_2 : r(a) \leftarrow 0.6\}$ *and the similarity* $\mathcal{R}$*, with entry* $\mathcal{R}(q, r) = 0.8$*, unfolding rule* R_1 *(w.r.t. rule* R_2*) leads to the set*

$$\mathcal{U} = \{p(a) \leftarrow @_{aver}(0.6 \ \wedge \ 0.8, 1)\}$$

since $\langle @_{aver}(q(x), 1), id \rangle \overset{SS}{\rightsquigarrow}_{R_2} \langle @_{aver}(0.6 \ \wedge \ 0.8, 1), \{X/a\} \rangle$ *and, therefore, the transformed program is:*

$$\mathcal{P}' = \{R_{1-2} : p(a) \leftarrow @_{aver}(0.6 \ \wedge \ 0.8, 1); \quad R_2 : r(a) \leftarrow 0.6\}.$$

■

Example 4. *Given the* FASILL *program* $\mathcal{P} = \{R_1 : p(x) \leftarrow @_{very}(q(x)); R_2 : r(b)\}$ *and the similarity* $\mathcal{R}$*, with entries* $\mathcal{R}(a, b) = 0.4$ *and* $\mathcal{R}(q, r) = 0.5$*, unfolding rule* R_1 *(w.r.t. rule* R_2*) leads to the set*

$$\mathcal{U} = \{p(b) \leftarrow @_{very}(0.5)\}$$

since $\langle @_{very}(q(x)), id \rangle \overset{SS}{\rightsquigarrow}_{R_2} \langle @_{very}(0.5), \{X/b\} \rangle$ *and, thus, the transformed program is:*

$$\mathcal{P}' = \{R_{1-2} : p(b) \leftarrow @_{very}(0.5); \quad R_2 : r(b)\}.$$

■

Unfortunately, examples 3 and 4 also illustrate that unfolding FASILL programs using Definition 4 is not safe in general.

Considering the program $\mathcal{P}$ in Example 3, it is possible to construct the following derivation:[2]

$$D_1: \quad \langle p(b); id\rangle \overset{\text{SS}}{\rightsquigarrow}_{R_1} \langle @_{aver}(q(b),1); \{x_1/b\}\rangle \overset{\text{FS}}{\rightsquigarrow} \langle @_{aver}(0,1); \{x_1/b\}\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.5; \{x_1/b\}\rangle$$

But in the unfolded program $\mathcal{P}'$ only the derivation D_1' can be constructed:

$$D_1': \langle p(b); id\rangle \overset{\text{FS}}{\rightsquigarrow} \langle 0; id\rangle$$

Observe that these derivations return different f.c.a.'s for the same goal, thus indicating that, in general, unfolding, as defined above, does not preserve those f.c.a.'s.

In the same way, for the program $\mathcal{P}$ in Example 4, it is possible to construct the following derivation:

$$D_2: \quad \langle p(a); id\rangle \overset{\text{SS}}{\rightsquigarrow}^{R_1} \langle @_{very}(q(a)); \{x_1/a\}\rangle \overset{\text{SS}}{\rightsquigarrow}_{R_2} \langle @_{very}(0.4 \ \wedge \ 0.5); \{x_1/a\}\rangle \overset{\text{IS}}{\rightsquigarrow} \langle @_{very}(0.4); \{x_1/a\}\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.16; \{x_1/a\}\rangle$$

However, in the unfolded program $\mathcal{P}'$ only the derivation D_2' can be constructed:

$$D_2': \langle p(a); id\rangle \overset{\text{SS}}{\rightsquigarrow}_{R_{1-2}} \langle 0.4 \ \wedge \ @_{very}(0.5); id\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.4 \ \wedge \ 0.25; id\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.25; id\rangle.$$

Once again, these derivations show that the unfolding transformation is neither sound nor complete. Therefore, some conditions must be imposed when unfolding FASILL programs if we want to recover the correctness of the transformation. In the following subsection we deal with this issue. But first we have to study the source of the problems.

3.1. Sources of the problems when unfolding FASILL *programs*

When studying in more detail the problem posed by examples 3 and 4, we can think that the difficulty appears because a "naive" unfolding is being

[2]For the sake of simplicity, in the discussions of this section, we are assuming that the t-norm $\wedge$ is the minimum t-norm.

carried out, based on traditional techniques, which does not take into account the semantics introduced by the fuzzy relation of similarity. By applying a traditional, purely syntactic unfolding, you are losing the semantics of the rule with which the unfolding is done (the so-called unfolding rule).

Example 5. *Consider once again the program and similarity relation in Example 4. In this example the unfolding rule was* $R_2 : r(b)$ *and in the context of the similarity* $\mathcal{R}$ *(with entries* $\mathcal{R}(a,b) = 0.4$ *and* $\mathcal{R}(q,r) = 0.5$*) this rule has an associated meaning. We can understood the meaning of* R_2 *as the following set of rules:*

$$\{R_{21} : r(b), \quad R_{22} : r(a) \leftarrow 0.4, \quad R_{23} : q(b) \leftarrow 0.5, \quad R_{24} : q(a) \leftarrow 0.5 \wedge 0.4\}$$

which reflects that when we write "$r(b)$*" in a program we are writing as well "*$r(a) \leftarrow 0.4$*" because* $r(b)$ *is similar to* $r(a)$ *with a* 0.4 *degree and analogously for the rest of the rules. Now, if we unfold* R_1 *taken into account the rules above, we obtain the unfolded rules:*

$$\{R_{1-21} : p(b) \leftarrow @very(0.5), \quad R_{1-22} : p(a) \leftarrow @very(0.5 \wedge 0.4)\}$$

what leads to the transformed program:

$$\mathcal{P}'' = \{R_{1-21} : p(b) \leftarrow @very(0.5), R_{1-22} : p(a) \leftarrow @very(0.5 \wedge 0.4), R2 : r(b)\}.$$

■

For this new transformed program $\mathcal{P}''$, there exists the derivation:

$$\langle p(a); id\rangle \overset{\text{SS}}{\rightsquigarrow}_{R_{1-22}} \langle @very(0.5 \wedge 0.4); id\rangle \overset{\text{IS}}{\rightsquigarrow} \langle @very(0.4); id\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.16; id\rangle,$$

which reproduce Derivation $\mathcal{D}_2$ in Example 4 but it is still possible to construct the wrong derivation (performing the first step with the rule R_{1-21}):

$$\langle p(a); id\rangle \overset{\text{SS}}{\rightsquigarrow}_{R_{1-21}} \langle 0.4 \wedge @very(0.5); id\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.4 \wedge 0.25; id\rangle \overset{\text{IS}}{\rightsquigarrow} \langle 0.25; id\rangle.$$

Therefore, even taken into account the semantics aspects of the problem, we obtain a transformed program $\mathcal{P}''$ that, although it is complete (in the sense that it can reproduce the derivations of the initial program $\mathcal{P}$), it is not sound (because it can compute answers which are not computed by $\mathcal{P}$).

There must be other aspects to consider in the solution of this problem. Studying Example 4 more in detail we can see that a great part of the

problems come from the way substitutions interact with the head of the rule to be unfolded (in this case, the rule $R_1 \in \mathcal{P}$).

Although substitutions can be "fuzzified" in some respect, note that FASILL (as well as it is done by other proximity-based languages like Bousi∼Prolog [6, 10]) uses the classical notion of a substitution disregarding other possible alternatives. At the syntactic level, in a substitution variables are bound to terms and some meaning is lost in the context of a similarity unless we agree that such substitution is a representative of a class of similar substitutions. For instance in Example 4 the substitution $\{X/b\}$ computed when unfolding rule $R_1 \in \mathcal{P}$ should be seen as a representative of the substitution $\{X/a\}$ (with a 0.4 degree) because $\mathcal{R}(a, b) = 0.4$. So, this fact needs to be taken into account in the process of unfolding a rule.

Finally note that, the problems we are describing in this section may be related with the use of rules which contain aggregators or disjunctions. Therefore, it is important to determine the role of aggregators or disjunctions in this problem.

3.2. A correct similarity-based unfolding algorithm

Studying more deeply Example 4, we can see that the problems with the current unfolding rule arise because, in this case, the rule $R_1 : p(x) \leftarrow @_{very}(q(x))$ to be unfolded has a variable x in the head of the rule that also appears in the body of the rule as part of a expression $@_{very}(q(x))$ with an aggregator in its context $@_{very}(q([\,]))$. Then, when we apply the substitution $\{x/b\}$ during the unfolding process, we obtain the unfolded rule $R_{1-2} : p(b) \leftarrow @_{very}(0.5)$. This leads to a wrong combination of the approximations when we compute goals like $\langle p(a); id\rangle$.

A way to resolve the last problem and, at the same time, to take into account the semantic introduced by the similarity $\mathcal{R}$, is to carry out a previous transformation step on the unfolding rule $R_2 : r(b)$. The idea is to perform a partial weak unification process, freezing some steps and introducing what we call "similarity constraints" (or "similarity guards") which represent delayed weak unification steps. A *similarity constraint* is a unification problem of the $t_1 \approx t_2$, where t_1 and t_2 are terms and predicate $\approx$ is pre-defined by means of the fact $x \approx x$, such that, if $wmgu(t_1, t_2) = \langle \theta, r\rangle$, then $\langle \mathcal{Q}[t_1 \approx t_2], id\rangle \rightsquigarrow \langle \mathcal{Q}[r]\theta, \theta\rangle$.

With this default predicate, we can make an intermediate transformation on rule $R_2 : r(b)$ consisting on flattening it: $R_2' : r(x) \leftarrow x \approx b$. Now, by

unfolding $R1$ w.r.t. R'_2 we obtain the transformed program:

$$\mathcal{P}'' = \{R_{1-2} : p(x) \leftarrow @_{very}(0.5 \wedge x \approx b); \quad R_2 : r(b)\},$$

which restores the correction of the transformation.

This reflection leads to the following unfolding algorithm, where a sequence of n objects (i.e., variables, terms, and so on) $o_1, \ldots, o_n$, is abbreviated as $\overline{o_n}$.

Definition 5 (Similarity-based Unfolding). *Let $\mathcal{P} = \langle \Pi, \mathcal{R}, L \rangle$ be a* FASILL *program, let $R : A \leftarrow \mathcal{B}[q(\overline{s_n})] \in \Pi$ be a program rule with no empty body and $\mathcal{U} = \{R_i : r(\overline{t_n}) \leftarrow \mathcal{Q} \in \Pi \mid wmgu(q(\overline{s_n}), r(\overline{t_n})) \neq fail\}$ the set of unfolding rules for R. Then, in order to obtain the similarity-based unfolding of program $\mathcal{P}$ w.r.t. the rule R, follow these steps:*

1. *Construct the set of flat unfolding rules:*

$$\mathcal{F} = \{R'_i : r(\overline{x_n}) \leftarrow g(\overline{x_n}) \approx g(\overline{t_n}) \wedge \mathcal{Q} \mid R_i : r(\overline{t_n}) \leftarrow \mathcal{Q} \in \mathcal{U}\}$$

   where $x_1, \ldots, x_n$ are fresh variables and g is a special function symbol used for packing the sequences of terms $\overline{x_n}$ and $\overline{t_n}$ into a single weak unification problem.[3]
2. *Construct the the intermediate program: $\mathcal{P}' = (\mathcal{P} - \mathcal{U}) \cup \mathcal{F}$.*

Then, the similarity-based unfolding of $\mathcal{P}$ w.r.t. the rule R is the new program

$$\mathcal{P}'' = (\mathcal{P} - \{R\}) \cup \mathcal{U}'$$

where $\mathcal{U}' = \{A\sigma \leftarrow \mathcal{B}' \mid \langle \mathcal{B}; id \rangle \rightsquigarrow \langle \mathcal{B}'; \sigma \rangle$ in $\mathcal{P}'\}$.

The main drawback of Definition 5 is that it may multiply the number of similarity constraints in each unfolding step, which can reduce the efficiency of the unfolding transformation by having to evaluate the delayed constraints later.

[3]The name g stands for "guard"'. It is used for constructing a similarity guard. By means of the similarity guard we force the satisfaction of all partial unification problems, $x_i \approx t_i$, before the rest of the computation could continue.

4. An efficient similarity-based unfolding transformation

Recovering the efficiency of the similarity-based unfolding transformation requires to eliminate the proliferation of similarity constraints in the unfolded rules. For this purpose, in this section, we propose a new unfolding algorithm. First we need to introduce some concepts and new definitions.

The operational semantics of FASILL operates on subgoals at positions of goals that contain them. In order to define the unfolding algorithm we need a more accurate treatment of the positions of an expression. *Positions* of an expression t (also called *occurrences*) are represented by sequences of natural numbers used to address subterms of t. The concatenation of the sequences p and w is denoted by $p.w$. We let ϵ denote the empty sequence. The prefix order, defined as $u \leq v$ iff there exist two w such that $v = u.w$ is a partial order on positions. $\mathcal{P}os(t)$ denotes the set of positions of an expression t. If $p \in \mathcal{P}os(t)$, $t[p]$ denotes the symbol of t at position p, $t|_p$ denotes the subterm of t at position p. Also we use the notation $t[s]_p$ to denote that the term t contains the term s in its position p [4] and $t[s_1, \ldots, s_n]_{p_1,\ldots,s_n}$ a term t with the subterm s_i at position p_i, $i \in \{1, \ldots, n\}$.

Definition 6 (Critical expression). *Let $A[x] \leftarrow \mathcal{B}$ be a rule to be unfolded. $\mathcal{B}$ is a* critical expression *for the variable x if there exist positions u and w in $\mathcal{P}os(\mathcal{B})$ such that $\mathcal{B}[u]$ refers to a connective different from $\wedge$ (i.e., the t-norm associated to the similarity relation $\mathcal{R}$), $\mathcal{B}|_w = x$ and $u \leq w$.*

Example 6. *Consider once again the program of Example 3, in the rule $R_1 : p(x) \leftarrow @_{aver}(q(x), 1)$ the body $\mathcal{B} \equiv @_{aver}(q(x), 1)$ is a critical expression for x because $\mathcal{B}[\epsilon] = @_{aver}$, $\mathcal{B}|_{1.1} = x$ and $\epsilon \leq 1.1$. Roughly speaking, $\mathcal{B}$ is a critical expression for x if the position of the variable x has an ancestor position with a connective different from $\wedge$ ($@_{aver}$ in this case).* ■

With regard to Definition 5, the new unfolding transformation we are proposing is more selective at the time of selecting the positions in which the unfolding rules are flattened, avoiding all unnecessary flattening.

[4] Note that we are using a non-standard notation in this case. It is usual (e.g. in the field of term rewriting systems) to employ the notation $t[s]_p$ to denote the result of replacing the subterm $t|_p$ with the term s. But this is not the meaning we are using.

Definition 7 (Flattening Positions). *Let $A[x] \leftarrow \mathcal{B}$ be a rule to be unfolded and $\mathcal{B}$ a critical expression for the variable x. If $B \leftarrow \mathcal{Q}$ is an unfolding rule, the atom $\mathcal{B}|_u$ weakly unifies with B and $\mathcal{B}[u.w] = x$ then w is a flattening position of B. We denote the set of flattening position of B by $\mathcal{F}lat\mathcal{P}os(B)$.*

The flattening positions of an unfolding rule are the unique positions that need to be flattened to preserve a correct behavior of the unfolding transformation in the presence of a similarity relation.

Definition 8 (Flat Unfolding Rule). *Let $R \equiv B \leftarrow \mathcal{Q}$ be an unfolding rule and $\mathcal{F}lat\mathcal{P}os(B) = \{w_1, \ldots, w_n\}$. The* flat unfolding rule *of R is*

$$B' \leftarrow g(\overline{x_n}) \approx g(\overline{s_n}) \wedge \mathcal{Q}$$

where $B'|_{w_i} = x_i$ and $s_n = B|_{w_i}$ and g is a special function symbol used for constructing a similarity guard. Note that if $\mathcal{F}lat\mathcal{P}os(B) = \emptyset$, the flat unfolding rule of R is itself.

Now, we are at position of defining an efficient similarity-based unfolding transformation.

Definition 9 (Efficient Similarity-based Unfolding). *Let $\mathcal{P} = \langle \Pi, \mathcal{R}, L \rangle$ be a* FASILL *program, let $R : A \leftarrow \mathcal{B}[C] \in \Pi$ be a program rule with no empty body and $\mathcal{U} = \{B \leftarrow \mathcal{Q} \in \Pi \mid wmgu(C, B) \neq fail\}$ the set of unfolding rules for R.*

1. *Construct the set of flat unfolding rules $\mathcal{F}$:*

$$\mathcal{F} = \{B[\overline{x_n}]_{\overline{w_n}} \leftarrow g(\overline{x_n}) \approx g(\overline{s_n}) \wedge \mathcal{Q} \mid B[\overline{s_n}]_{\overline{w_n}} \leftarrow \mathcal{Q} \in \mathcal{U}, \mathcal{F}lat\mathcal{P}os(B) = \{\overline{w_n}\}\}$$

   where $x_1, \ldots, x_n$ are fresh variables and g is a special function symbol used for constructing a similarity guard.
2. *Construct the the intermediate program: $\mathcal{P}' = (\mathcal{P} - \mathcal{U}) \cup \mathcal{F}$.*

Then, the similarity-based unfolding of $\mathcal{P}$ w.r.t. the rule R is the new program

$$\mathcal{P}'' = (\mathcal{P} - \{R\}) \cup \mathcal{U}'$$

where $\mathcal{U}' = \{A\sigma \leftarrow \mathcal{B}' \mid \langle \mathcal{B}; id \rangle \rightsquigarrow \langle \mathcal{B}'; \sigma \rangle$ in $\mathcal{P}'\}$.

Example 7. *Coming back to example 4, Since* $R_2 : r(b)$ *is the only unfolding rule to be flattened, we get* $\mathcal{F} = \{R'_2 : r(x) \leftarrow g(x) \approx g(b)\}$ *and the intermediate program:*

$$\mathcal{P}' = \{R_1 : p(x) \leftarrow @_{very}(q(x)); \quad R'_2 : r(x) \leftarrow g(x) \approx g(b)\}$$

Next, the final unfolded program is:

$$\mathcal{P}'' = \{R_{1-2} : p(x) \leftarrow @_{very}(0.5 \wedge g(x) \approx g(b)); \quad R_2 : r(b)\}$$

which coincides with the one we would have also obtained after applying Definition 5.

Anyway, note that we can avoid the flattening process when considering programs without critical expressions on rules to be unfolded. This is the case, for example, of the classical operation for concatenating lists:

$$\begin{array}{rcl} \{R_1 & : & append([], l, l). \\ R_2 & : & append([h|t], l, [h|r]) \leftarrow append(t, l, r).\} \end{array}$$

that can be safely and efficiently unfolded according Definition 9 following the classical unfolding style without using similarity guards:

$$\begin{array}{rcl} \{R_1 & : & append([], l, l). \\ R_2 & : & append([h], l, [h|l]). \\ R_{2-2} & : & append([h, h'|t], l, [h, h'|r]) \leftarrow append(t, l, r).\} \end{array}$$

■

5. Conclusions and Future Work

FASILL is a fuzzy logic programming language with implicit/explicit truth degree annotations, a great variety of connectives and unification by similarity. In [2, 3] we have recently provided the syntax, operational/declarative semantics, and implementation issues of this language which properly manages similarity and truth degrees in a single framework. In this work we have provided a safe and effective formulation of an unfolding transformation for optimizing FASILL programs. We have pointed out that, in contrast with other precedent languages, the treatment of similarities introduces several risks that can lead to loss of the correctness of the transformation and we have identified the conditions that preserve the soundness, completeness and efficiency of the new unfolding operation.

We are nowadays implementing the technique explained so far into the online FASILL programming environment (freely available at `https://dectau.uclm.es/fasill/sandbox`) and, on the theoretical side, we want to formally prove the correctness of the transformation.

References

[1] Jesús Manuel Almendros-Jiménez, Antonio Becerra-Terón, and Ginés Moreno. Fuzzy queries of social networks with FSA-SPARQL. *Expert Syst. Appl.*, 113:128–146, 2018.

[2] Pascual Julián Iranzo, Ginés Moreno, and Jaime Penabad. Thresholded semantic framework for a fully integrated fuzzy logic language. *J. Log. Algebr. Meth. Program.*, 93:42–67, 2017.

[3] Pascual Julián Iranzo, Ginés Moreno, and José Antonio Riaza. The fuzzy logic programming language FASILL: design and implementation. *Int. J. Approx. Reason.*, 125:139–168, 2020.

[4] P. Julián-Iranzo, G. Moreno, and J. Penabad. On Fuzzy Unfolding. A Multi-adjoint Approach. *Fuzzy Sets and Systems*, 154:16–33, 2005.

[5] P. Julián-Iranzo and C. Rubio-Manzano. An efficient fuzzy unification method and its implementation into the bousi∼prolog system. In *Proc. of the IEEE Int. Conference on Fuzzy Systems, Barcelona, Spain*, pages 1–8. IEEE, 2010. http://dx.doi.org/10.1109/FUZZY.2010.5584193.

[6] P. Julián-Iranzo and C. Rubio-Manzano. A sound and complete semantics for a similarity-based logic programming language. *Fuzzy Sets and Systems*, pages 1–26, 2017.

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

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

[9] A. Pettorossi and M. Proietti. Rules and Strategies for Transforming Functional and Logic Programs. *ACM Computing Surveys*, 28(2):360–414, 1996.

[10] C. Rubio-Manzano and P. Julián-Iranzo. A fuzzy linguistic prolog and its applications. *Journal of Intelligent and Fuzzy Systems*, 26(3):1503–1516, 2014.

[11] M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.

[12] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In S. Tärnlund, editor, *Proc. of Second Int'l Conf. on Logic Programming*, pages 127–139, 1984.