Symbolic Unfolding of Similarity-based Fuzzy Logic Programs

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

José Antonio Riaza
Department of Computing Systems
University of Castilla-La Mancha
02071 Albacete (Spain)
Email: JoseAntonio.Riaza@uclm.es

***Abstract*—FASILL introduces "*Fuzzy Aggregators and Similarity Into a Logic Language*". In its symbolic extension, called sFASILL, some truth degrees, similarity annotations and fuzzy connectives can be left unknown, so that the user can easily figure out the impact of their possible values at execution time. In this paper, we adapt to this last setting a similarity-based, symbolic variant of unfolding rule, which is very well known in most declarative frameworks. This semantics-preserving transformation technique is based on the application of computational steps on the bodies of program rules for improving efficiency. The method has been implemented in a freely available online tool and, to the best of our knowledge, it represents the first approach for unfolding fuzzy logic programs coping with symbolic similarity relations.**

***Index Terms*—Fuzzy Logic Programming; Similarity; Symbolic Unfolding.**

I. Introduction

During the last decades, the logic language Prolog has been fuzzified by embedding similarity relations or using fuzzy connectives for dealing with truth degrees beyond $\{true, false\}$, respectively. We have recently combined both approaches in the design of FASILL [2], whose symbolic extension (inspired by our initial experiences with MALP [8]) is called sFASILL [11].

This last symbolic language is useful for *flexibly tuning* (according to users preferences) the fuzzy components of fuzzy logic programs. Although there exist other approaches which are able to *tune* fuzzy truth degrees and connectives [15][16][17], none of them manage similarity relations as the tuning technique we describe in [11] does. We have used sFASILL, and its tuning engine, for developing two real world applications in the fields of the semantic web [1] and neural networks [9].

Besides this, unfolding is a well-known and widely used semantics-preserving program transformation rule, which is able to improve programs, generating more efficient code. The unfolding transformation traditionally considered in pure logic programming consists in the replacement of a program clause C by the set of clauses obtained after applying a computation step in all its possible forms on the body of C [14][19].

In order to briefly illustrate the essence and benefits of the transformation, consider a very simple Prolog program containing a clause, say $p(X)$:-$q(X)$, and a fact, say $q(a)$, for defining two (crisp, not fuzzy) predicates, p and q. It is easy to see that both rules must be used in two computational steps for successfully executing a goal like $p(a)$. Alternatively, we can unfold the first clause by applying a computational step on its body $q(X)$ (using the fact $q(a)$) and next instantiating the head with the achieved substitution $\{X/a\}$. Then, the new unfolded rule is just the simple fact $p(a)$, which must be used in only one computational step (instead of two, as before) to solve goal $p(a)$. This very simple example reveals that all computational steps applied at unfolding time *remain compiled* on unfolded rules forever, and hence, those steps have no longer to be repeated in all subsequent executions of the transformed programs. This justifies why unfolding is able to improve the efficiency of transformed programs by accelerating their computational behaviour.

In [3][4], we successfully adapted such operation to fuzzy logic programs dealing with lattices of truth degrees and similarity relations, but this type of unfolding was not symbolic yet. On the contrary, in [10] we defined a symbolic version of the transformation but in absence of similarities. Inspired by both works, in this paper we plan to go an step beyond by fusing both approaches in the definition of a similarity-based symbolic transformation.

The structure of this paper is as follows. After summarizing, in Section II, the syntax of FASILL and sFASILL, in Section III we detail how to execute and unfold such programs. Finally, we conclude and propose future work in Section IV.

II. The FASILL Language and its Symbolic Extension

In this work, given a complete lattice L, we consider a first order language $\mathcal{L}_L$ built upon a signature Σ_L, that contains the elements of a countably infinite set of variables $\mathcal{V}$, function and predicate symbols (denoted by $\mathcal{F}$ and Π, respectively) with an associated arity—usually expressed as pairs f/n or p/n, respectively, where n represents its arity—, and the truth degree literals Σ_L^T and connectives Σ_L^C from L. Therefore, a well-formed formula in $\mathcal{L}_L$ can be either:

- A *value* $v \in \Sigma_L^T$, which will be interpreted as itself, i.e., as the truth degree $v \in L$.
- $p(t_1, \ldots, t_n)$, if $t_1, \ldots, t_n$ are terms over $\mathcal{V} \cup \mathcal{F}$ and p/n is an n-ary predicate. This formula is called *atomic* (atom, for short).
- $\varsigma(e_1, \ldots, e_n)$, if $e_1, \ldots, e_n$ are well-formed formulas and ς is an n-ary connective with truth function $[\![\varsigma]\!] : L^n \mapsto L$.

Definition 1 (Complete Lattice)**.** A *complete lattice* is a partially ordered set $(L, \leq)$ such that every subset S of L

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

$\&_{\texttt{prod}}(x,y) \triangleq x * y$	$\|_{\texttt{prod}}(x,y) \triangleq x + y - xy$	*Product logic*
$\&_{\texttt{godel}}(x,y) \triangleq \min(x,y)$	$\|_{\texttt{godel}}(x,y) \triangleq \max(x,y)$	*Gödel logic*
$\&_{\texttt{luka}}(x,y) \triangleq \max(0,x+y-1)$	$\|_{\texttt{luka}}(x,y) \triangleq \min(x+y,1)$	*Łukasiewicz logic*

Fig. 1. Conjunctions and disjunctions of three different fuzzy logics over $([0,1],\leq)$.

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.

Example 1. In this paper, we use the lattice $([0,1],\leq)$, where $\leq$ is the usual ordering relation on real numbers, and three sets of conjunctions/disjunctions corresponding to the fuzzy logics of Gödel, Łukasiewicz and Product (with different capabilities for modelling *pessimistic*, *optimistic* and *realistic scenarios*), defined in Figure 1. It is possible to also include other fuzzy connectives (aggregators) like the arithmetical average $@_{\texttt{aver}}(x,y) \triangleq (x+y)/2$ or the linguistic modifier $@_{\texttt{very}}(x) \triangleq x^2$.

Definition 2 (Similarity Relation)**.** 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 $\mathcal{U}$, that is, a fuzzy subset on $\mathcal{U}\times\mathcal{U}$ (namely, a mapping $\mathcal{R}:\mathcal{U}\times\mathcal{U}\to L$) fulfilling the following properties: reflexive $\forall x\in\mathcal{U}, \mathcal{R}(x,x)=\top$, symmetric $\forall x,y\in\mathcal{U}, \mathcal{R}(x,y)=\mathcal{R}(y,x)$, and transitive $\forall x,y,z\in\mathcal{U}, \mathcal{R}(x,z)\geq \mathcal{R}(x,y)\wedge\mathcal{R}(y,z)$.

The fuzzy logic language FASILL relies on complete lattices and similarity relations [2]. We are now ready for summarizing its *symbolic* extension where, in essence, we allow some undefined values (truth degrees) and connectives in program rules as well as in the associated similarity relation, so that these elements can be systematically computed afterwards. The symbolic extension of FASILL we initially presented in [11] is called sFASILL.

Given a complete lattice L, we consider an augmented signature $\Sigma_L^\#$ producing an augmented language $\mathcal{L}_L^\# \supseteq \mathcal{L}_L$, which may also include a number of symbolic values and symbolic connectives, which do not belong to L. Symbolic objects are usually denoted as $o^\#$ with a superscript $\#$ and, in our tool, their identifiers always start with $\#$. An $L^\#$-*expression* is now a well-formed formula of $\mathcal{L}_L^\#$, which is composed by values and connectives from L as well as by symbolic values and connectives. We let $\exp_L^\#$ denote the set of all $L^\#$-expressions in $\mathcal{L}_L^\#$. Given a $L^\#$-expression E, $[\![E]\!]$ refers to the new $L^\#$-expression obtained after evaluating as much as possible the connectives in E. Particularly, if E does not contain any symbolic value or connective, then $[\![E]\!]=v\in L$.

In the following, we consider *symbolic substitutions* that are mappings from symbolic values and connectives to expressions over $\Sigma_L^T\cup\Sigma_L^C$. We let $\mathsf{sym}(o^\#)$ denote the symbolic values and connectives in $o^\#$. Given a symbolic substitution Θ for $\mathsf{sym}(o^\#)$, we denote by $o^\#\Theta$ the object that results from $o^\#$ by replacing every symbolic symbol $e^\#$ by $e^\#\Theta$.

Definition 3 (Symbolic Similarity Relation)**.** Given a domain $\mathcal{U}$ and a lattice L with a fixed —possibly symbolic— t-norm $\wedge$, a *symbolic similarity relation* is a mapping $\mathcal{R}^\#:\mathcal{U}\times\mathcal{U}\to \exp_L^\#$ such that, for any symbolic substitution Θ for $\mathsf{sym}(\mathcal{R}^\#)$, the result of fully evaluating all L-expressions in $\mathcal{R}^\#\Theta$, say $[\![\mathcal{R}^\#\Theta]\!]$, is a similarity relation.

Definition 4 (Symbolic Rule and Symbolic Program)**.** Let L be a complete lattice. A *symbolic rule* over L is a formula $A\leftarrow\mathcal{B}$, where the following conditions hold:

- A is an atomic formula of $\mathcal{L}_L$ (the head of the rule);
- $\leftarrow$ is an implication from L or a symbolic implication;
- $\mathcal{B}$ (the body of the rule) is a symbolic goal, i.e., a well-formed formula of $\mathcal{L}_L^\#$;

A sFASILL *program* is a tuple $\mathcal{P}^\#=\langle\Pi^\#,\mathcal{R}^\#,L\rangle$ where $\Pi^\#$ is a set of symbolic rules, $\mathcal{R}^\#$ is a symbolic similarity relation between the elements of the signature Σ of $\Pi^\#$, and L is a complete lattice.

Example 2. Consider a symbolic sFASILL program $\mathcal{P}^\#=\langle\Pi^\#,\mathcal{R}^\#,L\rangle$ based on lattice $L=([0,1],\leq)$, where $\Pi^\#$ is the following set of symbolic rules:

$$\Pi^\#=\begin{cases} R_1: & vanguardist(rizt)\leftarrow 0.9\\ R_2: & elegant(hydropolis)\leftarrow s_3^\#\\ R_3: & close(hydropolis,taxi)\leftarrow 0.7\\ R_4: & good\_hotel(x)\leftarrow \\ & \qquad @_{s4}^\#(elegant(x),@_{very}(close(x,metro))) \end{cases}$$

Note here that we leave unknown the level in which the hotel $hydropolis$ is more or less elegant (see the symbolic constant $s_3^\#$ in the second fact) as well as which should be the most appropriate connective for combining two features required on good hotels (see the symbolic constant $@_{s4}^\#$ in the body of the fourth rule).

The symbolic similarity relation $\mathcal{R}^\#$ on $\mathcal{U}=\{vanguardist,\ elegant, modern, metro, taxi, bus\}$, is represented by the graph shown in Figure 2 (a matrix can be also used to represent this concept).

This symbolic similarity relation $\mathcal{R}^\#$ has been obtained after applying the closure algorithm we initially introduced in [11], which is inspired by [6][7][13] and, in essence, is an adaptation of the classical Warshall's algorithm for computing transitive closures. In this particular example, we have selected the symbolic t-norm $\&_{s2}^\#$ and the following set of similarity equations: $elegant\sim modern=s_0^\#$, $modern\sim vanguardist=0.9$, $metro\sim bus=0.5$ and $bus\sim taxi=s_1^\#$.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

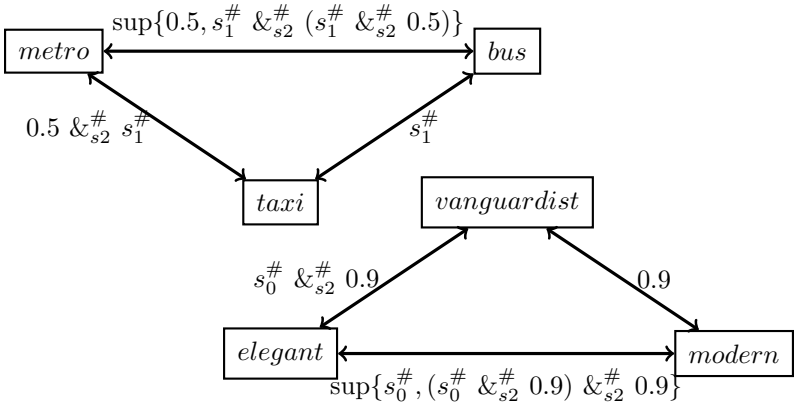


Fig. 2. Example of symbolic similarity relation $\mathcal{R}^{\#}$.

III. Running and Unfolding sFASILL Programs

As a logic language, sFASILL inherits the concepts of substitution, unifier and most general unifier (mgu) from pure logic programming, but extending some of them in order to cope with similarities, as Bousi~Prolog [5] does, where the concept of most general unifier is replaced by the one of *weak most general unifier* (w.m.g.u.). One step beyond, in [11] we extended again this notion by referring to *symbolic weak most general unifiers* (s.w.m.g.u.) and a symbolic weak unification algorithm was introduced to compute them. Roughly speaking, the *symbolic 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 —possibly symbolic— degree (i.e. $\mathcal{R}^{\#}(f, g) = r \neq \bot$) and each of their arguments t_i and s_i weakly unify. Therefore, there is a symbolic weak unifier for two expressions even if the symbols at their roots are not syntactically equal ($f \not\equiv g$).

More technically, the symbolic weak unification algorithm can be seen as an reformulation/extension of the ones appearing in [18] (since now we manage arbitrary complete lattices) and [2][5] (because now we deal with symbolic similarity relations). In essence, the *symbolic weak most general unifier* of two expressions $\mathcal{E}_1$ and $\mathcal{E}_2$, say $wmgu^{\#}(\mathcal{E}_1, \mathcal{E}_2) = \langle \sigma, E \rangle$, is the simplest *symbolic substitution* σ of $\mathcal{E}_1$ and $\mathcal{E}_2$ together with its *symbolic unification degree* E verifying that $E = \hat{\mathcal{R}}(E_1\sigma, E_2\sigma)$.

Example 3. Given the complete lattice $L = ([0, 1], \leq)$ of Example 1 and the symbolic similarity relation $\mathcal{R}^{\#}$ of Example 2, we can use the symbolic t-norm $\&_{s2}^{\#}$ for computing the following two symbolic symbolic weak most general unifiers: $wmgu^{\#}(modern(taxi), vanguardist(bus)) = \langle \{\}, 0.9 \ \&_{s2}^{\#} \ s_1^{\#} \rangle$ and $wmgu^{\#}(close\_to(X, taxi), close\_to(ritz, bus)) = \langle \{X/ritz\}, s_1^{\#} \rangle$

In order to describe the procedural semantics of the sFASILL 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 $\mathcal{Q}$, L is the complete lattice associated to $\Pi^{\#}$ and, as usual, rules are renamed apart:

Definition 5 (Computational Step)**.** Let $\mathcal{Q}$ be a goal and σ a substitution. The pair $\langle \mathcal{Q}; \sigma \rangle$ is a *state*. Given a symbolic program $\langle \Pi^{\#}, \mathcal{R}^{\#}, L \rangle$ and a (possibly symbolic) t-norm $\wedge$ in L, 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 $\overset{\text{SS}}{\rightsquigarrow}$):

$$\frac{A' \leftarrow \mathcal{B} \in \Pi^{\#} \qquad \langle \mathcal{Q}[A], \sigma \rangle \qquad wmgu^{\#}(A, A') = \langle \theta, E \rangle \qquad E \neq \bot}{\langle \mathcal{Q}[A/E \wedge \mathcal{B}]\theta, \sigma\theta \rangle} \text{ SS}$$

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

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

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

$$\frac{\langle \mathcal{Q}; \sigma \rangle \text{ where } \mathcal{Q} \text{ is a } L^{\#}\text{-expression}}{\langle [\![\mathcal{Q}]\!]; \sigma \rangle} \text{ IS}$$

Definition 6 (Derivation and Symbolic Fuzzy Computed Answer)**.** A *derivation* is a sequence of arbitrary length $\langle \mathcal{Q}; id \rangle \rightsquigarrow^* \langle \mathcal{Q}'; \sigma \rangle$. When $\mathcal{Q}'$ is an $L^{\#}$-expression that cannot be further reduced, $\langle \mathcal{Q}'; \sigma' \rangle$, where $\sigma' = \sigma[\mathcal{V}ar(Q)]$, is called a *symbolic fuzzy computed answer* (sfca). Also, if $\mathcal{Q}'$ is a concrete value of L, we say that $\langle \mathcal{Q}'; \sigma' \rangle$ is a *fuzzy computed answer* (fca).

The following example illustrates the operational semantics of sFASILL.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

Example 4. Let $\mathcal{P}^{\#} = \langle \Pi^{\#}, \mathcal{R}^{\#}, L \rangle$ be the program from Example 2. It is possible to perform the following derivation for $\mathcal{P}^{\#}$ and goal $\mathcal{Q} = good\_hotel(x)$ obtaining the sfca $\langle \mathcal{Q}_1; \sigma_1 \rangle = \langle @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), 0.0); \{x/ritz\} \rangle$:

$$\langle \underline{good\_hotel(x)}, id \rangle \overset{\text{SS}}{\rightsquigarrow}{}^{R4}$$
$$\langle @^{\#}_{s4}(\underline{elegant(x_1)}, @_{very}(close(x_1, metro))), \{x/x_1\} \rangle \overset{\text{SS}}{\rightsquigarrow}{}^{R1}$$
$$\langle @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), @_{very}(\underline{close(ritz, metro)})), \{x/ritz\} \rangle \overset{\text{FS}}{\rightsquigarrow}$$
$$\langle @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), \underline{@_{very}(0.0)}), \{x/ritz\} \rangle \overset{\text{IS}}{\rightsquigarrow}$$
$$\langle @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), 0.0), \{x/ritz\} \rangle$$

Apart from this derivation, there exists a second one ending with the alternative sfca $\langle \mathcal{Q}_2; \sigma_2 \rangle = \langle @^{\#}_{s4}(s^{\#}_3, @_{very}(\&^{\#}_{s2}(\&^{\#}_{s2}(0.5, s^{\#}_1), 0.7))); \{x/hydropolis\} \rangle$ associated to the same goal. Observe the presence of symbolic constants coming from the symbolic similarity relation, which contrast with our precedent work [8].

Now, let $\Theta = \{s^{\#}_0/0.8, s^{\#}_1/0.8, \&^{\#}_{s2}/\&_{\texttt{luka}}, s^{\#}_3/1.0, @^{\#}_{s4}/@_{\texttt{aver}}\}$ be a symbolic substitution that can be used for instantiating the previous sFASILL program in order to obtain a non-symbolic, fully executable FASILL program. This substitution can be automatically obtained by the tuning tool we described in [11] after introducing a couple of test cases (i.e., `0.4-> good_hotel(hydropolis)` and `0.6-> good_hotel(ritz)`), which represent the desired degrees for two goals accordingly to the user preferences.

Now we are ready to introduce the similarity-based symbolic unfolding transformations relying on the operational semantics described so far.

Definition 7 (Symbolic Unfolding)**.** Let $\mathcal{P}^{\#} = \langle \Pi^{\#}, \mathcal{R}^{\#}, L \rangle$ be a sFASILL program and $R : (H \leftarrow \mathcal{B}) \in \Pi^{\#}$ be a rule (with non-empty body $\mathcal{B}$). Then, the *symbolic unfolding* of rule R in program $\mathcal{P}^{\#}$ is the new sFASILL program $\mathcal{P}'^{\#} = \langle \Pi'^{\#}, \mathcal{R}^{\#}, L \rangle$, where $\Pi'^{\#} = (\Pi^{\#} - \{R\}) \cup \{H\sigma \leftarrow \mathcal{B}' \mid \langle \mathcal{B}; id \rangle \rightsquigarrow \langle \mathcal{B}'; \sigma \rangle\}$.

Example 5. Let us built a transformation sequence where each sFASILL program in the sequence is obtained from the immediately preceding one by applying symbolic unfolding, except the initial one $\mathcal{P}^{\#}_0 = \langle \Pi^{\#}_0, \mathcal{R}^{\#}, L \rangle$, which, in our case, is the one illustrated in Example 2, that is:

$$\Pi^{\#}_0 = \begin{cases} R_1: & vanguardist(rizt) \leftarrow 0.9 \\ R_2: & elegant(hydropolis) \leftarrow s^{\#}_3 \\ R_3: & close(hydropolis, taxi) \leftarrow 0.7 \\ R_4: & good\_hotel(x) \leftarrow \\ & \quad @^{\#}_{s4}(elegant(x), @_{very}(close(x, metro))) \end{cases}$$

Program $\mathcal{P}^{\#}_1 = \langle \Pi^{\#}_1, \mathcal{R}^{\#}, L \rangle$ is obtained after unfolding rule R_4 (with selected atom $elegant(x)$) by applying a $\overset{\text{SS}}{\rightsquigarrow}$ step with rules R_1 and R_2:

$$\Pi^{\#}_1 = \begin{cases} R_1: & vanguardist(rizt) \leftarrow 0.9 \\ R_2: & elegant(hydropolis) \leftarrow s^{\#}_3 \\ R_3: & close(hydropolis, taxi) \leftarrow 0.7 \\ R_{41}: & good\_hotel(ritz) \leftarrow @^{\#}_{s4}(\\ & \quad \&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), \\ & \quad @_{very}(close(ritz, metro))) \\ R_{42}: & good\_hotel(hydropolis) \leftarrow \\ & \quad @^{\#}_{s4}(s^{\#}_3, @_{very}(close(hydropolis, metro))) \end{cases}$$

After unfolding rule R_{41} (with selected atom $close(ritz, metro)$) by applying a $\overset{\text{FS}}{\rightsquigarrow}$ step, we obtain program $\mathcal{P}^{\#}_2 = \langle \Pi^{\#}_2, \mathcal{R}^{\#}, L \rangle$:

$$\Pi^{\#}_2 = \begin{cases} R_1: & vanguardist(rizt) \leftarrow 0.9 \\ R_2: & elegant(hydropolis) \leftarrow s^{\#}_3 \\ R_3: & close(hydropolis, taxi) \leftarrow 0.7 \\ R_{41F}: & good\_hotel(ritz) \leftarrow \\ & \quad @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), @_{very}(0.0)) \\ R_{42}: & good\_hotel(hydropolis) \leftarrow \\ & \quad @^{\#}_{s4}(s^{\#}_3, @_{very}(close(hydropolis, metro))) \end{cases}$$

When unfolding rule R_{42} (with selected atom $close(hydropolis, metro)$) by applying a $\overset{\text{SS}}{\rightsquigarrow}$ step with rule R_3, we reach the program $\mathcal{P}^{\#}_3 = \langle \Pi^{\#}_3, \mathcal{R}^{\#}, L \rangle$:

$$\Pi^{\#}_3 = \begin{cases} R_1: & vanguardist(rizt) \leftarrow 0.9 \\ R_2: & elegant(hydropolis) \leftarrow s^{\#}_3 \\ R_3: & close(hydropolis, taxi) \leftarrow 0.7 \\ R_{41F}: & good\_hotel(ritz) \leftarrow \\ & \quad @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), @_{very}(0.0)) \\ R_{423}: & good\_hotel(hydropolis) \leftarrow \\ & \quad @^{\#}_{s4}(s^{\#}_3, @_{very}(\&^{\#}_{s2}(\&^{\#}_{s2}(0.5, s^{\#}_1), 0.7))) \end{cases}$$

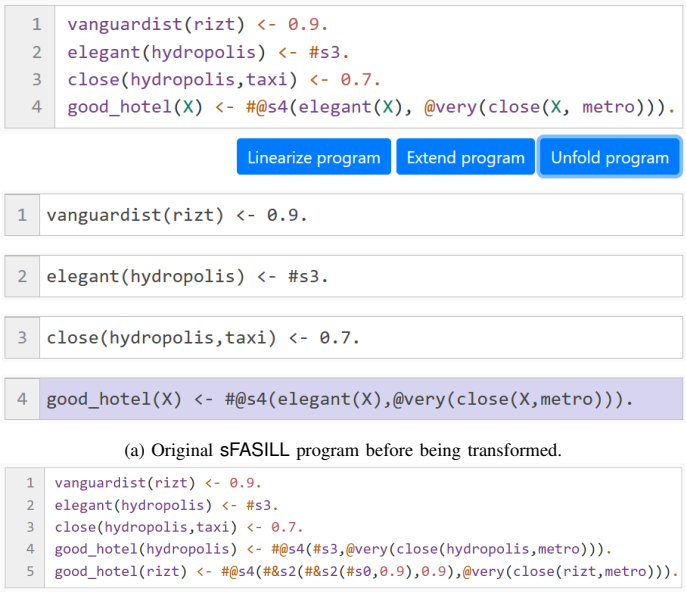
Finally, by unfolding rule R_{41F} (with selected expression $@_{very}(0.0)$) after applying a $\overset{\text{IS}}{\rightsquigarrow}$ step, we obtain the final program $\mathcal{P}^{\#}_4 = \langle \Pi^{\#}_4, \mathcal{R}^{\#}, L \rangle$:

$$\Pi^{\#}_4 = \begin{cases} R_1: & vanguardist(rizt) \leftarrow 0.9 \\ R_2: & elegant(hydropolis) \leftarrow s^{\#}_3 \\ R_3: & close(hydropolis, taxi) \leftarrow 0.7 \\ R_{41FI}: & good\_hotel(ritz) \leftarrow \\ & \quad @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), 0.0) \\ R_{423}: & good\_hotel(hydropolis) \leftarrow \\ & \quad @^{\#}_{s4}(s^{\#}_3, @_{very}(\&^{\#}_{s2}(\&^{\#}_{s2}(0.5, s^{\#}_1), 0.7))) \end{cases}$$

In the previous example, it is easy to see that each program in the sequence produces the same set of sfca's for a given goal but reducing the length of derivations. For instance, the derivation performed w.r.t. the original program $\mathcal{P}^{\#}_0$ illustrated in Example 4, can be emulated in the final program $\mathcal{P}^{\#}_4$ with just one computational step (instead of four) as:

$$\langle good\_hotel(x); id \rangle \overset{\text{SS}}{\rightsquigarrow}{}^{R41FI}$$
$$\langle @^{\#}_{s4}(\&^{\#}_{s2}(\&^{\#}_{s2}(s^{\#}_0, 0.9), 0.9), 0.0); \{x/ritz\} \rangle.$$

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org



(a) Original sFASILL program before being transformed.

(b) sFASILL program obtained after unfolding the last program rule.

Fig. 3. The FASILL online tool unfolding a symbolic program.

However, in the symbolic case, the unfolding transformation is not always safe, as the following example reveals.

Example 6. Consider a sFASILL program $\mathcal{P} = \langle \Pi^{\#}, \mathcal{R}^{\#}, L \rangle$ whose symbolic similarity relation establishes that $\mathcal{R}^{\#}(a,b) = v^{\#}$ and $\mathcal{R}^{\#}(q,r) = 0.5$ with a fixed t-norm $\wedge = \&_{luka}$:

$$\Pi^{\#} = \left\{ \begin{array}{l} p(x) \leftarrow q(x,b) \\ r(a,a) \leftarrow 0.5 \end{array} \right. \quad \Pi'^{\#} = \left\{ \begin{array}{l} p(a) \leftarrow 0.5 \wedge 0.5 \wedge v^{\#} \\ r(a,a) \leftarrow 0.5 \end{array} \right.$$

Now, if we apply to $\mathcal{P}$ a symbolic substitution Θ which replaces $v^{\#}$ by 0.4 then, the unfolding of $\mathcal{P}\Theta = \langle \Pi^{\#}\Theta, \mathcal{R}^{\#}\Theta, L \rangle$, where $(\mathcal{R}^{\#}\Theta)(a,b) = 0.4$, produces the following set of rules:

$$(\Pi^{\#}\Theta)' = \{p(x) \leftarrow 0,\ r(a,a) \leftarrow 0.5\}$$

This program is different to the instantiated unfolded one (observe, in particular, that the head of the first rule in both programs are different):

$$\Pi'^{\#}\Theta = \{p(a) \leftarrow 0.5 \wedge 0.5 \wedge 0.4,\ r(a,a) \leftarrow 0.5\}$$

IV. Conclusion and Future Work

The symbolic extension of the FASILL language based on symbolic similarity relations we introduced in [11] has been used in this paper for developing an effective unfolding technique for sFASILL programs, which is available at [12] (see in Figure 3 two screenshots of a work session with the FASILL online tool, before and after unfolding a symbolic program). Here, we have surpassed both the similarity-based (but non-symbolic) unfolding of [3][4], thus permitting the optimization of sFASILL programs in an unified, similarity-based symbolic framework.

As ongoing work, we are nowadays developing the formal proofs that ensure the correctness of the transformation under certain safe applicability conditions.

Acknowledgment

This work has been partially supported by the EU (FEDER), the State Research Agency (AEI) of the Spanish Ministry of Science and Innovation under grant PID2019-104735RB-C42 (SAFER).

References

[1] J. M. Almendros-Jiménez, A. Becerra-Terón, G. Moreno, and J. A. Riaza, "Tuning fuzzy sparql queries," *International Journal of Approximate Reasoning*, vol. 170, pp. 109209, 2024.

[2] P. Julián, G. Moreno, and J. Penabad, "Thresholded semantic framework for a fully integrated fuzzy logic language," *J. Log. Algebr. Meth. Program.*, vol. 93, pp. 42–67, 2017.

[3] P. Julián, G. Moreno, and J. A. Riaza, "Seeking a safe and efficient similarity-based unfolding rule," *Int. J. Approx. Reason.*, vol. 163, pp. 109038, 2023.

[4] P. Julián, G. Moreno, and J. A. Riaza, "Some properties of substitutions in the framework of similarity relations," *Fuzzy Sets Syst.*, vol. 465, pp. 108510, 2023.

[5] P. Julián and C. Rubio, "A declarative semantics for bousi∼prolog," In *Proc. of 11th Int. ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'09, ACM*, pp. 149–160, 2009.

[6] P. Julián, "A procedure for the construction of a similarity relation," In *Proc. of the 12th International Conference on Information Processing and Management of Uncertainty in Knoledge-based Systems, IPMU'08, U. Málaga (ISBN 978-84-612-3061-7)*, pp. 489–496, 2008.

[7] A. Kandel and L. Yelowitz, "Fuzzy chains," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-4, no. 5, pp. 472–475, 1974.

[8] G. Moreno, J. Penabad, J. A. Riaza, and G. Vidal, "Symbolic execution and thresholding for efficiently tuning fuzzy logic programs," In *Logic-Based Program Synthesis and Transformation, Proc. of the 26th International Symposium LOPSTR'16*, vol. 10184 LNCS-Springer, pp. 131–147, 2016.

[9] G. Moreno, J. Pérez, and J. A. Riaza, "Fuzzy logic programming for tuning neural networks," In *Rules and Reasoning - Proc. of the Third International Joint Conference, RuleML+RR'19*, vol. 11784 LNCS-Springer, pages 190–197, 2019.

[10] G. Moreno and J. A. Riaza, "An online tool for unfolding symbolic fuzzy logic programs," In *Advances in Computational Intelligence - Proc. of the 15th International Work-Conference on Artificial Neural Networks (Part II), IWANN'19*, vol. 11507 LNCS-Springer, pp. 475–487, 2019.

[11] G. Moreno and J. A. Riaza, "A safe and effective tuning technique for similarity-based fuzzy logic programs," In *Advances in Computational Intelligence - Proc. of the 16th International Work-Conference on Artificial Neural Networks, IWANN'21*, vol. 12861 LNCS-Springer, pp. 190–201, 2021.

[12] G. Moreno and J. A. Riaza, "FASILL: Sandbox," https://dectau.uclm.es/fasill/sandbox. *Accessed: 2024-06-24.*

[13] H. Naessens, H. De Meyer, and B. De Baets, "Algorithms for the computation of t-transitive closures," *IEEE Trans. Fuzzy Systems*, vol. 10, no. 4, pp. 541–551, 2002.

[14] A. Pettorossi and M. Proietti, "Rules and strategies for transforming functional and logic programs," *ACM Computing Surveys*, vol. 28, no. 2, pp. 360–414, 1996.

[15] L. De Raedt and A. Kimmig, "Probabilistic (logic) programming concepts," *Mach. Learn.*, vol. 100, no. 1, pp. 5–47, 2015.

[16] F. Riguzzi and T. Swift, "The PITA system: Tabling and answer subsumption for reasoning under uncertainty," *Theory Pract. Log. Program.*, vol. 11, no. 4-5, pp. 433–449, 2011.

[17] K. F. Sagonas, T. Swift, and D. S. Warren, "XSB as an efficient deductive database engine," In *Proc. of the ACM SIGMOD International Conference on Management of Data*, pp. 442–453, ACM Press, 1994.

[18] M. I. Sessa, "Approximate reasoning by similarity-based SLD resolution," *Theoretical Computer Science*, vol. 275, no. 1-2, pp. 389–426, 2002.

[19] H. Tamaki and T. Sato, "Unfold/Fold transformations of logic programs," In *Proc. of the Second International Conference on Logic Programming*, pp. 127–139, 1984.

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org