Implementing Fuzzy OSF Logic Unification and Normalization (Short Paper)

Gian Carlo Milanese[0000−0002−1757−0152]✉ and Gabriella Pasi[0000−0002−6080−8170]

University of Milano-Bicocca, DISCo, Milan, Italy
{giancarlo.milanese, gabriella.pasi}@unimib.it

Abstract. We present an implemented reasoner for fuzzy order-sorted feature (OSF) logic, supporting fuzzy unification and normalization of OSF terms modulo a sort theory. Fuzzy OSF logic is a knowledge representation and reasoning language based on feature symbols, denoting functions, and sort symbols, denoting fuzzy sets. Sort symbols are organized in a fuzzy subsumption relation that extends to OSF terms, record-like structures representing classes of entities. The unification algorithm for these structures provides a calculus of fuzzy type subsumption. We demonstrate the system's behavior on representative examples, such as computing the membership degree of an instance to a sort, and computing subsumption degrees between sorts and OSF terms. We also report a comparison with a resolution-based fuzzy logic programming system.

Keywords: Fuzzy logic · Fuzzy unification · Order-sorted logic

1 Introduction

Order-sorted feature (OSF) logic is a knowledge representation and reasoning language that originates in Hassan Aït-Kaci's work [1]. As its name suggests, OSF logic is based on *sorts*, symbols that represent sets of objects such as *person* or *movie*, and *features*, symbols that denote functional attributes such as *title* or *directed_by*. Sorts are organized in a subsumption relation that extends to OSF terms, record-like structures representing classes of entities. Fuzzy OSF logic [25] is a generalization of OSF logic in which sorts and OSF terms are interpreted as *fuzzy subsets* of the domain of an interpretation rather than crisp subsets, and where the sort subsumption relation is generalized to a *fuzzy subsumption relation between sorts*, which is then extended to OSF terms. Reasoning in (fuzzy) OSF logic is based on the *unification* of OSF terms, a procedure that aims to combine the constraints expressed by two OSF terms in a consistent way, taking the (fuzzy) sort subsumption relation into account. In short, fuzzy OSF logic allows reasoning about structured objects with graded membership and subsumption degrees, providing a calculus of fuzzy type subsumption.

Contributions. We introduce `fosf`, a Python implementation of the core reasoning algorithms of fuzzy OSF logic.[1] Its main features are: (i) computing the

[1] Source code and examples are available at https://github.com/fuzzyosf/fosf.

degree of membership of an instance to a sort, computing the greatest lower bound (GLB) of two sorts and sort subsumption degrees; (ii) normalizing OSF constraints and OSF terms, performing unification of two or more OSF terms, and computing the subsumption degree of one OSF term with respect to another; (iii) normalizing an OSF term modulo an OSF theory, i.e., a collection of sort definitions that impose structural constraints on terms, and computing the satisfaction degree of an OSF term with respect to the OSF theory. These features are presented, respectively, in Sections 2 to 4, alongside the essential definitions behind fuzzy OSF logic and high-level implementation details. Section 5 reports a comparison with the fuzzy logic programming system Bousi∼Prolog [19].

Related work. Implementations of (crisp) OSF logic include the constraint logic programming language LIFE [2] and the CEDAR semantic web reasoner [10]. Closely related to OSF logic, description logics (DLs) and their fuzzy extensions are supported by several reasoners (e.g., [11, 29–31, 34]). The similarities and differences between OSF logic and DLs have been studied extensively [3, 20, 27, 28]. In particular, fuzzy sort subsumption in fuzzy OSF logic differs significantly from graded subsumption in fuzzy DLs, which is typically defined via a fuzzy implication operator [12, 33]. The meaning of GLB is also different in fuzzy DLs, where it is defined as a lower bound for the degree of truth of an axiom [32]. Related notions of fuzzy unification have been defined in fuzzy logic programming, where unification is guided by similarity or proximity relations on functors (e.g., [7, 16, 18, 21]), rather than a fuzzy subsumption. Implementations include systems like Bousi∼Prolog [19] and FASILL [17].

2 Implementing Fuzzy OSF Taxonomic Reasoning

At the core of fuzzy OSF logic is a fuzzy sort taxonomy, a finite set of sort symbols $\mathcal{S}$ that is ordered in a fuzzy lattice $\preceq : \mathcal{S} \times \mathcal{S} \to [0,1]$ with top element $\top$ and bottom element $\bot$.[2] We simply write $s_0 \preceq s_1$ if $\preceq(s_0, s_1) > 0$. Given a fuzzy OSF interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, each sort $s \in \mathcal{S}$ is interpreted as a fuzzy subset $s^\mathcal{I} : \Delta^\mathcal{I} \to [0,1]$ that assigns a membership degree to each element of the domain. In fuzzy OSF logic the fuzzy subsumption relation $s_0 \preceq s_1$ models a weak notion of set inclusion, where an element $d \in \Delta^\mathcal{I}$ can be an instance of ${s_1}^\mathcal{I}$ with a degree that *may possibly be smaller than* its degree of membership to ${s_0}^\mathcal{I}$. Formally, the relation $\preceq$ has the following semantics [25]:

$$\text{if } \preceq(s_0, s_1) = \beta, \text{ then } \forall d \in \Delta^\mathcal{I} : s_0^\mathcal{I}(d) \wedge \beta \leq s_1^\mathcal{I}(d). \tag{1}$$

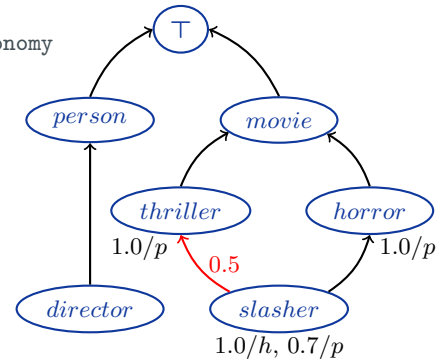
For example, assuming that $\preceq(\mathit{slasher}, \mathit{thriller}) = 0.5$, the movie *halloween* may be such that $\mathit{slasher}^\mathcal{I}(\mathit{halloween}) = 1$ and $\mathit{thriller}^\mathcal{I}(\mathit{halloween}) = 0.6$. Note that Zadeh's inclusion of fuzzy sets [15] is a special case of (1) where $\beta = 1$.

Parsing and encoding a fuzzy sort taxonomy. Fig. 1a shows how `fosf` supports parsing a fuzzy taxonomy, specified as a string of fuzzy subsumption and fuzzy instance declarations. The result is a Python object `ftax` of type

[2] See Appendix A. In practice, fuzzy partial orders may also be adopted [26].

```
from fosf.parsers import parse_taxonomy
ftax_str = """
# Fuzzy subsumption declarations
director < person.
slasher < thriller (0.5), horror.
thriller, horror < movie.
# Fuzzy instance declarations
{h, 0.7/p} < slasher.
{p} < horror, thriller.
"""
ftax = parse_taxonomy(ftax_str)
```

(a) Parsing a fuzzy taxonomy in `fosf`.



(b) Graphical representation.

Fig. 1: Example of how a fuzzy sort taxonomy is parsed in `fosf`, and its graphical representation (where $\bot$ is omitted). If the bottom and top elements are missing from the subsumption declarations, they are automatically added. The names h and p are short for the movies Halloween and Psycho.

`FuzzySortTaxonomy`. The fuzzy taxonomy is represented as a weighted directed acyclic graph (DAG), and `fosf` employs the bottom-up DAG encoding algorithm of [5] to assign a bitcode to each sort, represented as an integer. The fuzzy instance declarations are stored in a hash table named `ftax.instances`: for example, `ftax.instances["p"][Sort("slasher")]` is `0.7`. When fuzzy instance declarations are processed, `fosf` checks that they satisfy (1).

Computing GLBs of sorts. Thanks to the encoding, the GLB $s_0 \curlywedge s_1$ of two sorts s_0 and s_1 can be obtained simply by taking the bitwise `AND` of their corresponding codes, and decoding the result [5]. If the GLB exists, decoding consists of simply finding the sort corresponding to the code, which can be performed efficiently by maintaining a hash table mapping integers to sorts. If the GLB does not exist (in case the fuzzy taxonomy is just a fuzzy partial order, rather than a lattice), decoding yields the set of maximal lower bounds of s_0 and s_1.[3] For instance, `ftax.glb("thriller", "horror")` returns `"slasher"`. Checking if $s_0 \preceq s_1$ amounts to verifying whether $s_0 \curlywedge s_1 = s_0$. E.g., `ftax.is_subsort("slasher", "movie")` returns `True`.

Computing sort subsumption degrees. Obtaining $\preceq(s_0, s_1) \in [0, 1]$ requires traversing the DAG representation of the fuzzy sort taxonomy in order to compute the reflexive-transitive closure of the fuzzy subsort declarations. This is performed through a shortest-path-like algorithm that traverses the DAG in topological order, and that is further optimized thanks to the information provided by the sort encoding: during the traversal, while visiting a node s such that $s_0 \preceq s$, any immediate supersort s' of s can be excluded from the traversal if $s' \not\preceq s_1$, potentially avoiding the needless traversal of large portions of the sort

[3] The encoding and decoding algorithms are described in detail, e.g., in [4, 5].

```
from fosf.parsers import parse_term
term_str = "X0:movie(directed_by -> X1:person, written_by -> X1)"
term = parse_term(term_str); print(term.to_clause())
# Output: X0:movie & X0.directed_by=X1 & X0.written_by=X1 & X1:person.
```

Fig. 2: Parsing an OSF term in `fosf` and converting it to an OSF clause.

taxonomy [24]. As an example, `ftax.degree("slasher", "thriller")` returns `0.5`, while `ftax.degree("slasher", "movie")` returns `1.0`.

Computing instance membership degrees. Given an instance d and a sort s, computing $s^{\mathcal{I}}(d)$ amounts to taking the minimum of `ftax.instances[d][si]` and `ftax.degree(si, s)` for each $s_i \in \mathcal{S}$ of which d is a declared instance, and then taking the maximum value. E.g., `ftax.membership_degree("h","movie")` returns `1.0`, and `ftax.membership_degree("h","thriller")` returns `0.5`.

3 Implementing Fuzzy OSF Term Unification

OSF terms and clauses. In (fuzzy) OSF logic, sort symbols may be used together with variables (also called tags) $X_0, X_1 \ldots \in \mathcal{V}$ and features symbols $f_0, \ldots, f_n \in \mathcal{F}$ (interpreted as functions $f_i^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}$) to build *OSF terms*. These are record-like representations of (fuzzy) sets of objects, such as $t = X_0 : \mathit{movie}(\mathit{directed\_by} \to X_1 : \mathit{person}, \mathit{written\_by} \to X_1)$, which denotes the class of movies that are written and directed by the same person. The set of variables of t is denoted $\mathit{Tags}(t)$, and the variable X_0 is called the *root tag* of t. In general, OSF terms may include redundant or even contradictory information (e.g., the OSF term $s(f \to s_0, f \to s_1)$ is contradictory if $s_0 \curlywedge s_1 = \bot$). OSF terms that are well-behaved to this regard are called *normal OSF terms* or ψ*-terms* [8], and they are also denoted ψ, ψ_i, and so on. Every OSF term t enjoys an equivalent representation as an *OSF clause* $\phi(t)$, i.e., a conjunction of OSF constraints of shape $X : s$, $X \doteq X'$, or $X.f \doteq X'$. Fig. 2 shows how an OSF term can be parsed in `fosf` to build an internal representation as a Python object of type `Term`, and how it can be converted into an equivalent OSF clause.

Fuzzy OSF term subsumption. Normal OSF terms are ordered in a (fuzzy) subsumption lattice which extends the one on sort symbols [8, 25]. Specifically, a normal OSF term ψ_0 is subsumed by ψ_1 with degree α (notation: $\preceq(\psi_0, \psi_1) = \alpha$) if there is a mapping $h : \mathit{Tags}(\psi_1) \to \mathit{Tags}(\psi_0)$ such that (i) h maps the root of ψ_1 to the root of ψ_0, (ii) if X_0 and X_1 are connected by f in ψ_1, then $h(X_0)$ and $h(X_1)$ are connected by f in ψ_0, (iii) $\alpha = \min\{\preceq(\mathit{Sort}_{\psi_0}(h(X)), \mathit{Sort}_{\psi_1}(X)) \mid X \in \mathit{Tags}(\psi_1)\}$, where $\mathit{Sort}_{\psi}(X)$ is the most specific sort s such that $X : s$ appears in ψ. The value of $\preceq(\psi_0, \psi_1)$ can be computed through their unification.

OSF term normalization and unification. Two normal OSF terms ψ_0 and ψ_1 with roots, respectively, X_0 and X_1 can be unified by applying the OSF constraint normalization rules of Fig. 3 to the clause $\phi(\psi_0)$ & $\phi(\psi_1)$ & $X_0 \doteq X_1$, and translating the result back into an OSF term [8]. An analogous process can

Sort Intersection
$$\frac{\phi \,\&\, X : s \,\&\, X : s'}{\phi \,\&\, X : s \curlywedge s'}$$

Tag Elimination
$$\frac{[Y \in \mathit{Tags}(\phi)] \quad \phi \,\&\, X \doteq Y}{\phi[X/Y] \,\&\, X \doteq Y}$$

Failure
$$\frac{\phi \,\&\, X : \bot}{X : \bot}$$

Feature Functionality
$$\frac{\phi \,\&\, X.f \doteq Y \,\&\, X.f \doteq Y'}{\phi \,\&\, X.f \doteq Y \,\&\, Y \doteq Y'}$$

Fig. 3: OSF constraint normalization rules, where ϕ is an OSF clause. Each rule expresses that, whenever the (optional) condition in square brackets holds, the expression above the line can be simplified into the one below.

```
from fosf.reasoning import unify_terms
t1 = parse_term("horror(directed_by -> director)")
t2 = parse_term("thriller(directed_by -> X:person , written_by -> X)")
print(unify_terms([t1, t2], ftax, return_degree=True))
# Output: X0:slasher(directed_by -> X1:director, written_by -> X1) 0.5
```

Fig. 4: Unifying two OSF terms in `fosf`.

be applied to a single OSF term t in order to normalize it. Unification yields the GLB $\psi_0 \curlywedge \psi_1$ of ψ_0 and ψ_1 in the (fuzzy) OSF term subsumption lattice [8, 25]. Clearly, $\psi_0 \preceq \psi_1$ if ψ_0 is equal to $\psi_0 \curlywedge \psi_1$ (modulo an equivalence relation [25]). Based on [8, 25], an algorithm for normalizing OSF clauses is implemented in `fosf`, and works by maintaining (i) equivalence classes of tags via a union-find structure, (ii) a mapping from representative tags to bitcodes, and (iii) a mapping from representative tags to a feature→tag map. The algorithm iterates over the constraints, merging tags and their related structures when processing equality constraints, refining bitcodes when processing sort constraints, and ensuring functionality of features. When unifying ψ_0 and ψ_1, the algorithm also computes the mappings $h_i : \mathit{Tags}(\psi_i) \to \mathit{Tags}(\psi_0 \curlywedge \psi_1)$ necessary for computing the subsumption degrees $\preceq(\psi_0 \curlywedge \psi_1, \psi_i)$. The unification degree is computed as the minimum subsumption degree of $\psi_0 \curlywedge \psi_1$ with respect to ψ_0 and ψ_1 [25]. Fig. 4 shows how to perform the unification of two OSF terms in `fosf`, and returning the unification degree.

4 Fuzzy Term Normalization Modulo an OSF Theory

An OSF theory is a function $\Theta : \mathcal{S} \to \Psi$, where Ψ is the set of normal OSF terms, such that $\Theta(\bot) = \bot$, $\Theta(\top) = \top$, and the root sort of $\Theta(s)$ is s for all $s \in \mathcal{S}$ [9]. Intuitively, the *sort definition* $\Theta(s)$ specifies structural constraints that an OSF term of type s should satisfy. Fig. 5 shows an example of parsing an OSF theory in `fosf` and ensuring its *order-consistency*, that is, ensuring that $\preceq(\Theta(s), \Theta(s')) > 0$ if $\preceq(s, s') > 0$, for all $s, s' \in \mathcal{S}$. This is done by traversing the fuzzy sort taxonomy top-down and setting $\Theta(s)$ to be the GLB of the definitions of the immediate supersorts of s. In the example, the definition for the sort *person* requires that whenever a *person* X_0's *spouse* is X_1, then X_1 must be a *person* whose *spouse* is X_0. Moreover, if X_0 has a favourite movie, then (for the

```
from fosf.parsers import parse_theory
from fosf.reasoning import normalize_term
theory_str = ftax_str + """
person := Yperson:person(spouse->Y0:person(spouse->Yperson),
                         fav_movie->Y1:thriller).
movie  := Ymovie:movie(directed_by->Y2:director)."""
theory = parse_theory(theory_str, ensure_closed=True)
print(theory['slasher'])
# Output: Yslasher:slasher(directed_by->Y5:director)
t_str = "X0:person(spouse->X1(fav_movie->slasher(directed_by->X0)))"
t = parse_term(t_str)
nt, degree = normalize_term(t, ftax, theory, return_degree=True)
print(degree, nt) # Output:
# 0.5 X0:director(spouse->X1:person(spouse->X0,
#                                   fav_movie->X2:slasher(directed_by->X0))
```

Fig. 5: Ensuring order-consistency of an OSF theory and performing OSF term normalization modulo a theory in `fosf`.

sake of the example) it must be a *thriller*. In the fuzzy setting, the constraints imposed by an OSF theory may be satisfied up to a degree $\alpha \in [0, 1]$. For instance, X_1's favourite movie in the term `nt` of Fig. 5 is of sort *slasher* rather than *thriller*, so that `nt` satisfies $\Theta(person)$ with degree $\preceq(slasher, thriller) = 0.5$.

Normalizing an OSF term modulo an OSF theory. The library `fosf` implements an algorithm for normalizing an input (normal) OSF term ψ modulo an OSF theory according to the OSF theory normalization rules of [9], as also shown in Fig. 5. The algorithm associates each tag in ψ with at most one *frame*, which represents the portion of the OSF theory materialized for that variable. Frames are created, updated, and merged as constraints are processed, refining sorts and checking compatibility with theory-defined feature declarations. For instance, if $\psi = X : s_0(f_0 \to X_0 : \top(f_1 \to X_1 : \top))$, and $\Theta(s_0) = Y_{s_0} : s_0(f_0 \to Y_0 : s_0, f_1 \to Y_1 : s_1)$, then the normalization procedure must ensure that X_0 be of sort s_0, and then X_1 be of sort s_1. Note that OSF theory normalization is not guaranteed to terminate [9]; this is due to a single rule whose application in `fosf` is delayed until no other rule is applicable. In addition to normalizing a term ψ according to the OSF theory, `fosf`'s normalization algorithm computes a mapping from $Tags(\psi)$ to the relevant theory variables in order to compute the satisfaction degree of ψ with respect to the theory.

5 Comparison with Bousi∼Prolog

One advantage of (fuzzy) OSF logic is that a single unification step may replace several resolution steps, possibly leading to more efficient computations [6, 14]. To illustrate this effect in a representative reasoning task, we compared `fosf` and Bousi∼Prolog [19] on the task of computing membership degrees on randomly

Table 1: Average runtime (μs) for instance checking and membership degree computation for different DAG configurations (the average outdegree is around 1.35 for all DAGs). System details: Intel Core i7-8565U CPU, 16GB RAM, Ubuntu 20.04, Python 3.10.18, and Bousi∼Prolog (B∼P) v. 4.0.

Task \ Graph Depth		10	20	40	60	80	100	120
	#Nodes	74	178	726	2238	6122	16792	44090
Instance checking	B∼P	1.11E1	1.74E1	2.96E1	4.37E1	5.59E1	7.05E1	8.45E1
	`fosf`	6.46	6.55	6.87	2.81	2.86	2.94	3.27
Membership degree	B∼P	4.65E1	2.68E2	8.23E4	9.76E6	1.05E8[5]	–	–
	`fosf`	9.10E1	1.66E2	4.87E2	9.30E2	1.97E3	5.93E3	2.31E4

generated weighted graphs of varying depths and sizes. More specifically, the graphs are layered DAGs in which edges connect only consecutive layers, and nodes are grouped into multi-node branches that persist across layers and may stochastically split. One instance is added for each direct supersort of $\bot$.

Bousi∼Prolog was chosen as it natively supports weighted facts and rules, allowing a direct representation of fuzzy subsumption and fuzzy instance declarations as weighted facts, such as "`isa(s0, s1) with 0.5.`" and "`inst(d, s0) with 0.1.`", respectively. Additional rules define transitive closure and derived instance relations. Computing the membership degree in Bousi∼Prolog requires enumerating all derivations to obtain truth degrees, which are then aggregated. Encoding subsumptions as weighted rules like "`s1(X) :- s0(X) with 0.5.`", rather than facts, was avoided due to considerable slowdown.

For the comparison, we compute membership degrees for a sample of (d, s) pairs where d is an instance and s is a direct subsort of $\top$. We only sample pairs with positive membership degrees to ensure a non-trivial traversal of the graph in `fosf`. The random generation of DAGs and sampling was seeded for reproducibility.[4] The average runtime over the sampled pairs is reported in Table 1. The comparison highlights the computational differences between unification-based and resolution-based fuzzy subsumption reasoning. The observed performance gap reflects the contrast between optimized taxonomy traversal in `fosf` and the derivation-based strategy required by the Bousi∼Prolog encoding to compute subsumption degrees; it is not intended as a system-level benchmark.

6 Future Work

While `fosf` currently serves as a prototype implementation, it provides a foundation for further development. Planned extensions include similarity-based OSF term unification (which is built on top of fuzzy OSF logic [26]), support for resolution over predicates with OSF terms as arguments, and performance enhancements, for example by porting performance-critical components to Cython.

[4] More details are available at https://github.com/fuzzyosf/fosf.

[5] For this case, test pairs that produced "out of global stack" errors were ignored.

Acknowledgments. The authors express their gratitude and a fond thought to Hassan Aït-Kaci, who with Gabriella Pasi set out to define a fuzzy version of OSF logic. Our development of fuzzy OSF logic originates from their work on the definition of similarity-based unification for OSF terms, extending the approach of [7].

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

A Fuzzy Set Theory

We recall the basic definitions of fuzzy set theory [15] to fix the notation. Whenever possible we use the same notation for fuzzy sets and orders as the one used for crisp sets and orders, but with the addition of a dot ($\cdot$) to avoid ambiguity. E.g., the symbol for the intersection of two crisp sets is $\cap$, while the symbol for the intersection of two fuzzy sets is $\dot{\cap}$. We adopt the minimum as a t-norm (denoted $\wedge$) and the maximum as a t-conorm (denoted $\vee$).

Definition 1 (Fuzzy subset). *A* fuzzy subset F *of a (crisp) set* X *is determined by its membership function* $\mu_F : X \to [0,1]$.

Definition 2 (Intersection and union of fuzzy subsets). *The* intersection $\dot{\bigcap}\mathcal{F}$ of a set $\mathcal{F}$ of fuzzy subsets of a set X *is defined by letting* $\mu_{\dot{\bigcap}\mathcal{F}}(x) \stackrel{\text{def}}{=} \inf(\{\mu_F(x) \mid F \in \mathcal{F}\})$. *The* union $\dot{\bigcup}\mathcal{F}$ of a set $\mathcal{F}$ of fuzzy subsets of a set X *is defined by letting* $\mu_{\dot{\bigcup}\mathcal{F}}(x) \stackrel{\text{def}}{=} \sup(\{\mu_F(x) \mid F \in \mathcal{F}\})$.

Definition 3 (Support). *The* support *of a fuzzy subset* F *of* X *is defined as* $|F| \stackrel{\text{def}}{=} \{x \in X \mid \mu_F(x) > 0\}$.

Remark 1 (Fuzzy set notation). Throughout the paper the membership function of a fuzzy subset F of X is simply written $F : X \to [0,1]$ instead of $\mu_F : X \to [0,1]$.

Definition 4 (Fuzzy binary relation). *A* fuzzy binary relation R *on a set* X *is a fuzzy subset of* $X \times X$*, i.e., it is a function* $R : X \times X \to [0,1]$.

Definition 5 (Fuzzy preorder). *A fuzzy binary relation* R *on a set* X *is a* fuzzy preorder *if it satisfies:*

$$\forall x \in X,\ R(x,x) = 1, \qquad \text{(Fuzzy Reflexivity)}$$

$$\forall x, y, z \in X,\ R(x,z) \geq R(x,y) \wedge R(y,z). \qquad \text{(Max-Min Transitivity)}$$

Definition 6 (Fuzzy partial order). *A fuzzy binary relation* R *on a set* X *is a* fuzzy partial order *if it satisfies* (Fuzzy Reflexivity) *and* (Max-Min Transitivity) *and*

$$\forall x, y \in X, \textit{if } R(x,y) > 0 \textit{ and } R(y,x) > 0, \textit{then } x = y. \qquad \text{(Strong Fuzzy Antisymmetry)}$$

The pair (X, R) *is called a* fuzzy partially ordered set *(*fuzzy poset*).*

Definition 7 (Composition of fuzzy binary relations). *The* (max-min) composition *of two fuzzy binary relations R and Q on a finite set X is the fuzzy binary relation $R \circ Q$ defined by the membership function $R \circ Q(x,z) \stackrel{\text{def}}{=} \bigvee_{y \in X}(R(x,y) \wedge Q(y,z))$. The n-ary composition of a fuzzy binary relation R with itself is defined by letting $R^1 \stackrel{\text{def}}{=} R$ and $R^n \stackrel{\text{def}}{=} R \circ R^{n-1}$ for $n > 1$.*

Definition 8 (Reflexive and transitive closure of a fuzzy binary relation). *The* transitive closure *of a fuzzy binary relation R is defined as $R^{\oplus} \stackrel{\text{def}}{=} \bigcup\!\!\!\cdot_{m \geq 1} R^m$. The* reflexive and transitive closure $R^{\circledast}$ *of a fuzzy binary relation R is obtained by letting $R^{\circledast}(x,y) \stackrel{\text{def}}{=} 1$ if $x = y$ and $R^{\circledast}(x,y) \stackrel{\text{def}}{=} R^{\oplus}(x,y)$ otherwise.*

Remark 2 (Infix notation). If R is a fuzzy binary relation, then $xR_{\alpha}y$ stands for $R(x,y) = \alpha$, and xRy stands for $R(x,y) > 0$.

We adopt the definitions of lower bounds, greatest lower bounds and fuzzy lattice from [13, 22, 23].

Definition 9 (Lower bounds in a fuzzy poset). *Let $(\mathcal{S}, \preceq)$ be a fuzzy poset and $S \subseteq \mathcal{S}$. The set of* lower bounds of S *is defined as $S^{fl} \stackrel{\text{def}}{=} \{s \in \mathcal{S} \mid \forall s' \in S, s \preceq s'\}$.*

Definition 10 (Fuzzy greatest lower bound). *Let $(\mathcal{S}, \preceq)$ be a fuzzy poset and $S \subseteq \mathcal{S}$. The* greatest lower bound (GLB) of S *is the unique $s \in S^{fl}$ such that, for all $s' \in S^{fl}$, $s' \preceq s$. If the GLB of S exists, it is denoted $\bigwedge S$, or simply $s \wedge s'$ in case $S = \{s, s'\}$.*

Definition 11 (Fuzzy lattice and bounded lattice). *A fuzzy poset $(\mathcal{S}, \preceq)$ is a* fuzzy lattice *if every pair of elements has a GLB. A fuzzy lattice $(\mathcal{S}, \preceq)$ is bounded if there are elements $\bot, \top \in \mathcal{S}$ such that, for all $s \in \mathcal{S}$, $\preceq(\bot, s) = 1$ and $\preceq(s, \top) = 1$.*

References

1. Aït-Kaci, H.: A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures. Ph.D. thesis, University of Pennsylvania (1984), https://www.proquest.com/openview/3816ec450cb910c1586b7627c6353da6
2. Aït-Kaci, H.: An introduction to LIFE-programming with logic, inheritance, functions, and equations. In: Proceedings of the 1993 International Symposium on Logic Programming. pp. 52–68. ILPS '93, MIT Press, Cambridge, MA, USA (1993), https://dl.acm.org/doi/10.5555/187812.187831
3. Aït-Kaci, H.: Data models as constraint systems: a key to the semantic web. Constraint Processing Letters **1** (2007), https://constraint-programming.com/letters/Papers/v1/hak.pdf
4. Aït-Kaci, H., Amir, S.: Classifying and querying very large taxonomies. Tech. rep., LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, Villeurbanne, France (2013), https://projet.liris.cnrs.fr/cedar/papers/ctr2.pdf, CEDAR Technical Report Number 2, CEDAR Project

5. Aït-Kaci, H., Boyer, R., Lincoln, P., Nasr, R.: Efficient implementation of lattice operations. ACM Trans. Program. Lang. Syst. **11**(1), 115–146 (Jan 1989). https://doi.org/10.1145/59287.59293
6. Aït-Kaci, H., Nasr, R.: Login: a logic programming language with built-in inheritance. The Journal of Logic Programming **3**(3), 185–215 (1986). https://doi.org/10.1016/0743-1066(86)90013-0
7. Aït-Kaci, H., Pasi, G.: Fuzzy lattice operations on first-order terms over signatures with similar constructors: A constraint-based approach. Fuzzy Sets and Systems **391**, 1–46 (2020). https://doi.org/10.1016/j.fss.2019.03.019
8. Aït-Kaci, H., Podelski, A.: Towards a meaning of life. The Journal of Logic Programming **16**(3), 195–234 (1993). https://doi.org/10.1016/0743-1066(93)90043-G
9. Aït-Kaci, H., Podelski, A., Goldstein, S.C.: Order-sorted feature theory unification. The Journal of Logic Programming **30**(2), 99–124 (1997). https://doi.org/10.1016/S0743-1066(96)00053-2
10. Amir, S., Aït-Kaci, H.: An efficient and large-scale reasoning method for the semantic web. J. Intell. Inf. Syst. **48**(3), 653–674 (Jun 2017). https://doi.org/10.1007/s10844-016-0435-2
11. Bobillo, F., Straccia, U.: Optimising fuzzy description logic reasoners with general concept inclusion absorption. Fuzzy Sets and Systems **292**, 98–129 (2016). https://doi.org/10.1016/j.fss.2014.10.029, Special Issue in Honor of Francesc Esteva on the Occasion of his 70th Birthday
12. Borgwardt, S., Peñaloza, R.: Fuzzy Description Logics – A Survey. In: Moral, S., Pivert, O., Sánchez, D., Marín, N. (eds.) Scalable Uncertainty Management. pp. 31–45. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-67582-4_3
13. Chon, I.: Fuzzy partial order relations and fuzzy lattices. Korean J. Math. **17**(4), 361–374 (2009), https://www.kkms.org/kkms/2009/170403.pdf
14. Cohn, A.G.: Taxonomic reasoning with many-sorted logics. Artificial intelligence review **3**(2-3), 89–128 (1989). https://doi.org/10.1007/BF00128778
15. Dubois, D., Prade, H.: Fuzzy Sets and Systems. Theory and Applications., Mathematics in Science and Engineering, vol. 144. Elsevier (1980), https://www.sciencedirect.com/bookseries/mathematics-in-science-and-engineering/vol/144/
16. Gerla, G., Sessa, M.I.: Similarity in Logic Programming, pp. 19–31. Springer US, Boston, MA (1999). https://doi.org/10.1007/978-1-4615-5261-1_2
17. Julián-Iranzo, P., Moreno, G., Riaza, J.A.: The Fuzzy Logic Programming language FASILL: Design and implementation. International Journal of Approximate Reasoning **125**, 139–168 (2020). https://doi.org/10.1016/j.ijar.2020.06.002
18. Julián-Iranzo, P., Sáenz-Pérez, F.: Proximity-Based Unification: An Efficient Implementation Method. IEEE Transactions on Fuzzy Systems **29**(5), 1238–1251 (2021). https://doi.org/10.1109/TFUZZ.2020.2973129
19. Julián-Iranzo, P., Sáenz-Pérez, F.: Bousi∼Prolog: Design and implementation of a proximity-based fuzzy logic programming language. Expert Systems with Applications **213**, 118858 (2023). https://doi.org/10.1016/j.eswa.2022.118858
20. Krieger, H.U., Schäfer, U.: DL Meet FL: A Bidirectional Mapping between Ontologies and Linguistic Knowledge. In: Huang, C.R., Jurafsky, D. (eds.) Coling 2010: Posters. pp. 588–596. Coling 2010 Organizing Committee, Beijing, China (Aug 2010), https://aclanthology.org/C10-2067/
21. Kutsia, T., Pau, C.: Solving Proximity Constraints. In: Gabbrielli, M. (ed.) Logic-Based Program Synthesis and Transformation. pp. 107–122. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-45260-5_7

22. Mezzomo, I., Bedregal, B., Santiago, R.H.N.: Operations on bounded fuzzy lattices. In: 2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS). pp. 151–156 (2013). https://doi.org/10.1109/IFSA-NAFIPS.2013.6608391
23. Mezzomo, I., Bedregal, B.C.: On fuzzy α-lattices. In: 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 775–781 (2016). https://doi.org/10.1109/FUZZ-IEEE.2016.7737766
24. Milanese, G.C., Pasi, G.: Conjunctive Reasoning on Fuzzy Taxonomies with Order-Sorted Feature Logic. In: 2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). pp. 1–7 (2021). https://doi.org/10.1109/FUZZ45933.2021.9494474
25. Milanese, G.C., Pasi, G.: Fuzzy order-sorted feature logic. Fuzzy Sets and Systems **477**, 108800 (2024). https://doi.org/10.1016/j.fss.2023.108800
26. Milanese, G.C., Pasi, G.: Similarity-Based Reasoning with Order-Sorted Feature Logic. IEEE Transactions on Fuzzy Systems **32**(5), 2797–2810 (2024). https://doi.org/10.1109/TFUZZ.2024.3362897
27. Nebel, B., Smolka, G.: Representation and reasoning with attributive descriptions. In: Bläsius, K.H., Hedtstück, U., Rollinger, C.R. (eds.) Sorts and Types in Artificial Intelligence. pp. 111–139. Springer Berlin Heidelberg, Berlin, Heidelberg (1990). https://doi.org/10.1007/3-540-52337-6_21
28. Nebel, B., Smolka, G.: Attributive description formalisms... and the rest of the world, pp. 439–452. Springer Berlin Heidelberg, Berlin, Heidelberg (1991). https://doi.org/10.1007/3-540-54594-8_74
29. Shearer, R., Motik, B., Horrocks, I.: Hermit: A Highly-Efficient OWL Reasoner. In: OWL: Experiences and Directions (2008), https://api.semanticscholar.org/CorpusID:7951194
30. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics **5**(2), 51–53 (2007). https://doi.org/10.1016/j.websem.2007.03.004
31. Stoilos, G., Simou, N., Stamou, G., Kollias, S.: Uncertainty and the Semantic Web. IEEE Intelligent Systems **21**(5), 84–87 (2006). https://doi.org/10.1109/MIS.2006.105
32. Straccia, U.: A fuzzy description logic for the semantic web. In: Sanchez, E. (ed.) Fuzzy Logic and the Semantic Web, Capturing Intelligence, vol. 1, pp. 73–90. Elsevier (2006). https://doi.org/10.1016/S1574-9576(06)80006-7
33. Straccia, U.: Foundations of Fuzzy Logic and Semantic Web Languages. Chapman and Hall, New York (2014). https://doi.org/10.1201/b15460
34. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Furbach, U., Shankar, N. (eds.) Automated Reasoning. pp. 292–297. Springer Berlin Heidelberg, Berlin, Heidelberg (2006). https://doi.org/10.1007/11814771_26