SCUOLA DI DOTTORATO

UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA

Department of

Informatics, Systems and Communication

Ph.D. program in Computer Science Cycle XXXVI

Approximate Reasoning with Order-Sorted Feature Logic

Milanese Gian Carlo

Registration number: 848629

Supervisor: Prof. Gabriella Pasi

Tutor: Prof. Elisabetta Fersini

Coordinator: Prof. Leonardo Mariani

ACADEMIC YEAR 2022/2023

Abstract

Order-Sorted Feature (OSF) logic is a Knowledge Representation and Reasoning (KRR) language originating in Hassan Aït-Kaci's work on designing a calculus of partially ordered type structures. The language was developed to model the notions of subsumption and unification in inheritance-based KRR formalisms, and it has been applied in computational linguistics and implemented in constraint logic programming languages and automated reasoners.

The language of OSF logic is based on function-denoting *feature symbols* and on set-denoting *sort symbols* ordered in a *subsumption (is-a) lattice*. Reasoning with OSF logic relies on the *unification* of set-denoting structures called *OSF terms*, a process that aims to combine the constraints expressed by two OSF terms into a single term. An advantage of OSF logic is that its unification algorithm takes into account the subsumption ordering between sorts, which may enable a single unification step to replace several inference steps, leading to more efficient computations.

This thesis deals with the theoretical development of approximate reasoning within the framework of OSF logic. Two approaches are investigated: (i) a fuzzy generalization of OSF logic that provides this language with the capability to represent graded subsumption relations and vague concepts, and (ii) an extension of the language of OSF logic with a similarity relation, leading to a flexible notion of OSF term unification that allows matching different and disjoint sorts. Both strategies enable OSF logic to return approximate answers to queries posed to a knowledge base.

The first key contribution of the thesis is the definition of *fuzzy OSF logic*, a fuzzy generalization of the semantics of OSF logic where sort symbols denote *fuzzy sets* rather than crisp sets, allowing to represent vague concepts. Moreover, the sorts symbols of fuzzy OSF logic are ordered in a *fuzzy subsumption relation* (formally a fuzzy lattice) rather than a crisp one, which provides more modeling flexibility by allowing to represent graded subsumption relations. The fuzzy sort subsumption relation is given a special semantics which generalizes Zadeh's definition of inclusion of fuzzy sets, and which is then extended to OSF terms.

We investigate whether several semantic and computational properties of crisp OSF logic are preserved in the fuzzy setting. For instance, we show that OSF terms are ordered in a fuzzy subsumption relation which extends the fuzzy ordering between sort symbols, and we prove that the unification of two OSF terms yields their greatest lower bound. We also define procedures to compute the subsumption degree between two sort symbols or between two OSF terms, and study their computational complexity.

The second key contribution of this thesis is the definition of *similarity-based reasoning with OSF logic*. Approximate reasoning based on fuzzy relations – in particular similarity relations – has been studied extensively in the fuzzy logic programming literature. Several approaches consist in extending the Prolog language with a similarity relation, which enables the definition of flexible notions of unification and SLD resolution, namely weak unification and similarity-based SLD resolution.

Along these lines, defining similarity-based reasoning with OSF logic involves the development of a flexible unification procedure for OSF terms that takes into account a similarity relation between sort symbols. Since the sort symbols of OSF logic are ordered in a subsumption relation, this procedure should also account for the interactions between the two relations. Our proposed solution consists in a transformation that takes a sort subsumption relation and a sort similarity as inputs, producing a fuzzy sort subsumption relation that intuitively combines the information from both relations and their interactions. The advantage is that, as a consequence of this transformation, the same unification rules of fuzzy OSF logic can be applied to this setting, and a single unification may be able to replace several similarity-based SLD resolution steps.

With respect to practical applications of our approach, we discuss logic programming languages based on fuzzy OSF logic and on similarity-based OSF logic, and a similarity-based extension of the CEDAR reasoner, a Semantic Web reasoner based on OSF logic. These applications rely on a fuzzy subsumption relation, or on a sort similarity relation, in order to provide approximate answers to queries posed to a knowledge base.

Acknowledgements

I would like to express my gratitude to Prof. Gabriella Pasi. The research in this thesis stems from her project with Hassan Aït-Kaci on developing a similarity-based extension of OSF logic, and I am truly grateful for the chance to expand and continue their work. I also wish to thank Gabriella for being an attentive and involved supervisor, and for her ability to motivate me and point me to the right direction each time I felt lost. Our meetings discussing fuzzy OSF logic, the intuitions behind its semantics, and how to correctly formalize them, were some of the most enjoyable moments in my PhD career.

I wish to thank the reviewers of the thesis manuscript for their comments and suggestions, which helped to significantly improve the clarity of the presentation.

I am grateful to my family for their presence and support, especially my parents, who are the first people I seek for encouragement and advice. I would also like to thank my brother Alessandro for helping me double check a few proof details.

I wish to thank Alinda for her invaluable assistance, from helping me rehearse presentations and coming up with intuitive examples, to proofreading my papers and thesis. Above all, I am profoundly thankful for becoming her husband, and for her constant presence by my side throughout these years.

Contents

Introduction

Knowledge Representation and Reasoning (KRR) is the area of Artificial Intelligence concerned with how knowledge can be explicitly represented by symbols and manipulated in an automated way by reasoning algorithms in order to to solve complex problems [31, 57]. Over the years, the field has embraced a diverse range of approaches, starting from early formalisms like frames [85] and conceptual graphs [108], to the most recent applications integrating knowledge graphs with machine learning and language modeling techniques.

This thesis focuses on Order-Sorted Feature (OSF) logic [10], a KRR language rooted in Hassan Aït-Kaci's work on designing a calculus of partially ordered type structures [2]. Originally, this language was meant to model the notions of subsumption and unification in inheritance-based KRR formalisms such as Brachman's structured inheritance networks [2, 4, 65]. OSF logic and related formalisms – e.g., feature logic [106], or the logic of typed feature structures [34] – have been applied in computational linguistics (e.g., [34, 46, 107, 119]) and implemented in constraint logic programming languages such as LOGIN [8], LIFE [10] and CIL [86]. OSF logic has also been proposed as a KRR language for the Semantic Web by Aït-Kaci [4], and a Semantic Web reasoner based on OSF logic has been implemented in the context of the CEDAR project [6, 15].

As the name of the language suggests, OSF logic is based on *sort symbols* (or sorts) and *feature symbols* (or features). Features denote functional attributes of objects, while sorts represent sets of objects. Sorts are *ordered* in a subsumption (is-a) relation that models an inclusion relation between classes. Starting from sorts and features, OSF logic allows to build OSF terms, record-like structures that denote complex classes of objects.

Closely related to OSF logic are Description Logics (DLs), a family of formal languages that also descend from Brachman's structured inheritance networks and from the effort to overcome the lack of a formal semantics in earlier KRR formalisms like frames and semantic networks [18, 19, 31]. DLs allow to represent knowledge by building concept descriptions using a variety of concept and role constructors. Depending on the set of supported constructors, each DL can strike a different balance between the reasoning complexity and the expressivity required by a given application. The most prominent application of DLs is perhaps the Semantic Web, as they provide, by design, the semantics for the Web Ontology Language developed by the World Wide Web Consortium [18].

Due to their common origin, OSF logic and DLs share a few similarities. For instance, both languages are based on set-denoting symbols and on symbols for expressing attributes: sorts and features in OSF logic, concepts and roles in DLs. The sorts and the OSF terms of OSF logic are ordered in a subsumption relation and, similarly, DL concepts are organized in a subsumption hierarchy. In both languages, one of the main reasoning problems consists in deciding whether two sorts, OSF terms, or DL concepts are subsumed by each other. The two formalisms also exhibit several differences. Perhaps the most apparent one is that the features of OSF logic denote total functions, while the roles of DL are interpreted as binary relations. One of the most significant differences between the two languages is that the semantics of OSF logic is based on the *closed world assumption*, while DLs adopt an open world semantics.

There are several advantages that motivate the adoption of OSF logic for the development of KRR applications.

- One convenient feature of OSF logic is the flexibility provided by its terms, record-like structures that generalize the terms of first-order logic. OSF logic does not constrain its terms to have a fixed arity or a fixed argument order [8]. For instance, an OSF logic signature admits terms with the same root sort but possibly a different number and order of arguments, such as $movie(year \rightarrow 1960, directed\_by \rightarrow director)$ and $movie(directed\_by \rightarrow person)$. Because the arguments of an OSF term are identified by feature names rather than positions, OSF terms are also easier to interpret.

- OSF terms also provide a natural way of representing partial information [34]. During a computation, an OSF term like $X : person$, representing the class of people, can become more specific by inheriting additional constraints, resulting, for instance, in the term $X : professor(works\_at \rightarrow Y : university)$, representing the class of professors working at some university.

- Since each subterm of an OSF term can be associated with a variable, it is possible to express cyclic coreferences within a term, allowing the concise representation of infinite structures [8], like in the term

$$X : person \left(\begin{array}{lcl} last\_name & \rightarrow & Y : string, \\ spouse & \rightarrow & person \left(\begin{array}{lcl} spouse & \rightarrow & X, \\ last\_name & \rightarrow & Y \end{array} \right) \end{array} \right).$$

- In a logic programming setting, replacing the first-order terms of Prolog with OSF terms offers the advantage of integrating inheritance (an is-a subsumption relation) into the unification process, rather than through a resolution-based inference mechanism, which can result in more efficient computations [8, 37]. This is due to the fact

that a sort subsumption relation is taken into account in the unification of two OSF terms. This enables the unification, for instance, of terms such as $X : professor$ and $X : person(works\_at \rightarrow Y : university)$.

- The implementation of OSF logic is based on graph encoding techniques that allow to perform Boolean operations on sorts very efficiently [6, 13]. By leveraging these techniques, the CEDAR Semantic Web reasoner based on OSF logic has been experimentally demonstrated to be a highly efficient alternative to Semantic Web reasoners based on DLs [15].

Research questions and contributions

This thesis deals with the theoretical development of approximate reasoning within the framework of OSF logic, driven by the following research questions.

1. How can OSF logic be provided with a fuzzy semantics, enabling this language to support vague concepts and real-valued truth degrees?

2. How can the language of OSF logic be extended with a similarity relation on sorts, so as to define a flexible notion of unification that allows matching different and disjoint sorts?

In order to address these research questions, the two key contributions of this thesis are the definition of *fuzzy OSF logic* and of *similarity-based reasoning with OSF logic*.

Fuzzy OSF logic Fuzzy OSF logic maintains the basic syntax of OSF logic, but it generalizes its semantics by interpreting sort symbols as *fuzzy sets* rather than crisp sets, allowing to represent vague concepts. Moreover, the sorts symbols of fuzzy OSF logic are ordered in a *fuzzy subsumption relation* (formally a fuzzy lattice) rather than a crisp one, which provides more modeling flexibility by allowing to represent graded subsumption relations. The semantics of the fuzzy sort subsumption relation generalizes Zadeh's definition of inclusion of fuzzy sets [44], and it follows the intuition according to which, if the subsumption degree of a sort s with respect to another sort s' is equal to β (in symbols, $\preceq(s, s') = \beta$), then every instance of the sort s must also be an instance of s', with degree of membership greater than or equal to β.

We investigate whether several semantic and computational properties of crisp OSF logic are preserved in the fuzzy setting. For instance, we show that the fuzzy sort subsumption ordering can be extended to OSF terms, and that formally this ordering constitutes a fuzzy lattice such that the greatest lower bound of two OSF terms can be computed through their unification.

The generalization to a fuzzy semantics provides OSF logic with the capability to perform approximate reasoning. In particular, we provide an algorithm for OSF term unification that can be employed to decide whether two OSF terms are subsumed by each other, and to which degree. We also define procedures to compute the subsumption degree between two sort symbols or between two OSF terms, and study their computational complexity.

A potential application of fuzzy OSF logic is discussed, consisting of a fuzzy logic programming language that is capable of providing approximate answers to queries by leveraging a fuzzy subsumption relation between sort symbols. The development of a fuzzy semantics of OSF logic also enables the definition of similarity-based reasoning with this language.

Similarity-based reasoning with OSF logic Approximate reasoning based on fuzzy relations – in particular similarity relations – has been studied extensively in the fuzzy logic programming literature (e.g., [64, 72, 74, 101, 118]). Several approaches extend the Prolog language with a similarity relation, enabling the definition of flexible notions of unification and SLD resolution, namely weak unification and similarity-based SLD resolution.

One motivation behind the similarity-based approaches is to model a form of reasoning that may be referred to as *reasoning by analogy or by similarity* [101]. For instance, this may be achieved by relaxing the *equality* constraint on two functor symbols to a flexible constraint of *similarity*, when unifying two first-order terms. For example, if the functors `thriller` and `horror` are assumed to be similar, then the term `thriller(X)` can unify with the term `horror("Psycho")` to some degree, leading to approximate solutions to queries posed to a knowledge base.

Along these lines, defining similarity-based reasoning with OSF logic involves the development of a flexible unification procedure for OSF terms that takes into account a similarity relation between sort symbols. Since the sort symbols of OSF logic are ordered in a subsumption relation, this procedure should also account for the interactions between the two relations. Our proposed solution consists in a transformation that, taken as inputs a sort subsumption relation and a sort similarity, outputs a fuzzy sort subsumption relation, which intuitively combines the information of both relations, and how they interact. Informally, this is achieved by applying the following inference, inspired by the similarity-based approaches to logic programming (e.g., [101]):

$$\frac{\begin{array}{l}\text{If the sort } s_0 \text{ is subsumed by the sort } s_1 \\ \text{and } s_1 \text{ is similar to the sort } s_2 \text{ with degree } \beta\end{array}}{\text{then } s_0 \text{ is subsumed by } s_2 \text{ with degree } \beta.}$$

For example, if *slasher*s are *horror* movies, and *horror* movies are similar to *thriller*s with similarity degree 0.5, then it is possible to conclude that *slasher*s are also *thriller*s with subsumption degree 0.5.

One advantage of this approach is that, as a consequence of this transformation, the same unification rules of fuzzy OSF logic can be applied to this setting, and a single unification may be able to replace several similarity-based SLD resolution steps.

Two applications of similarity-based reasoning with OSF logic are discussed, namely a fuzzy logic programming language based on OSF terms, and an extension of the CEDAR Semantic Web reasoner. Both applications rely on a sort similarity relation to provide approximate answers to queries posed to a knowledge base.

Outline of the thesis

The thesis is structured as follows.

- Chapter 1 provides the definitions from order theory and fuzzy set theory that are necessary for the development of fuzzy OSF logic and of similarity-based reasoning with OSF logic.

- Chapter 2 is a brief summary of KRR languages and approaches that are closely related to OSF logic, so as to provide the context of our research, and a means of comparison of our work with the current literature. In particular, we discuss DLs and fuzzy DLs, the fuzzy generalization of DLs whose goal is to handle vagueness and real-valued truth degrees. The chapter also discusses the research on approximate reasoning with logic programming based on fuzzy relations such as proximities and similarities, which has inspired our work in Chapter 5.

- Chapter 3 is a thorough overview of the syntax and semantics of OSF logic that sets the foundations for our development of fuzzy OSF logic in Chapter 4. Two applications based on OSF logic are reviewed: (i) the logic programming language LOGIN, an extension of Prolog where first-order terms are replaced by OSF terms, and (ii) the CEDAR Semantic Web reasoner. The chapter also provides an overview of the graph encoding techniques that enable the efficient implementation of OSF logic, which are also relevant for the implementation of fuzzy OSF logic.

- Chapter 4 develops fuzzy OSF logic, our fuzzy generalization of the semantics of OSF logic where sorts and OSF terms are interpreted as fuzzy subsets of a domain of interpretation. The semantics of fuzzy OSF logic is investigated thoroughly, generalizing several results from crisp OSF logic. We provide procedures for the computation of the subsumption degree between two OSF terms or between two sorts, and we discuss their complexity and implementation. A potential application of fuzzy OSF logic consisting of a fuzzy logic programming language is introduced.

- Chapter 5 deals with similarity-based reasoning with OSF logic. The language of OSF logic is extended with a similarity relation on sort symbols. By combining this similarity relation with the usual sort subsumption relation of this language it is possible to define a fuzzy subsumption relation which intuitively combines the information of both relations and their interaction. We start by combining the two relations into a *fuzzy subsumption preorder*, which may however contain cycles. To address this issue, we introduce a construction of a fuzzy partial order from a fuzzy preorder that generalizes a well-known result from order theory. A definition of a completion of a fuzzy poset into a fuzzy lattice is presented next, which finalizes the transformation of a subsumption relation and a similarity into a fuzzy subsumption relation. The resulting fuzzy subsumption is then taken into account when unifying two OSF terms, according to the constraint normalization rules of fuzzy OSF logic. The chapter presents two potential applications of similarity-based OSF logic: a fuzzy logic programming language and an extension of the CEDAR reasoner which are capable of returning approximate answer to queries.

- Appendix A contains the proofs of the main results of Chapters 4 and 5.

Publications

The main contributions of this thesis are part of the following publications.

- Gian Carlo Milanese and Gabriella Pasi. "Conjunctive Reasoning on Fuzzy Taxonomies with Order-Sorted Feature Logic". In: *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. Luxembourg, Luxembourg, 2021, pp. 1–7. DOI: 10.1109/FUZZ45933.2021.9494474

  Discussed in Section 4.8, specifically the computation of the subsumption degree of two sorts, and the encoding of a fuzzy lattice or fuzzy partial order.

- Gian Carlo Milanese and Gabriella Pasi. "Fuzzy Order-Sorted Feature Term Unification". In: *Federated Logic Conference. 36th International Workshop on Unification.* Haifa, Israel, 2022. URL: https://www.cs.cas.cz/unif-2022/Papers/UNIF_2022_paper_3.pdf

  Discussed in Sections 4.6 and 4.7, specifically the syntactic definition of OSF term subsumption, and the definition of fuzzy OSF term unification.

- Gian Carlo Milanese and Gabriella Pasi. "Fuzzy order-sorted feature logic". In: *Fuzzy Sets and Systems* 477 (2024), p. 108800. ISSN: 0165-0114. DOI: https://doi.org/10.1016/j.fss.2023.108800

Discussed in Sections 4.1 to 4.7, involving the fuzzy generalization of the semantics of OSF logic.

- Gian Carlo Milanese and Gabriella Pasi. "Similarity-Based Reasoning with Order-Sorted Feature Logic". In: *IEEE Transactions on Fuzzy Systems* (2024), pp. 1–14. DOI: 10.1109/TFUZZ.2024.3362897

  Discussed in Chapter 5, involving the definition of similarity-based reasoning with OSF logic.

Chapter 1

Preliminaries

One of the distinctive feature of Order-Sorted Feature (OSF) logic is that its set of sort symbols is ordered in a finite bounded lattice, which models an inclusion relation between sets of objects. It is thus convenient to recall a few essential notions from order theory before delving into the formal development of this language. The definitions of preorder, partial order and lattice are recalled in Section 1.1, where the order theoretic notation employed throughout the thesis is also fixed. This section also discusses the construction of a partial order from a preorder (Proposition 1.20), and of a lattice from a partial order (Definition 1.15). The two constructions are useful from the perspective of implementing OSF logic, and fuzzy generalization of these two results will also be essential for the development of similarity-based OSF logic in Chapter 5. The definitions of Section 1.1 are based on [41].

In Chapter 4 we develop fuzzy OSF logic, a fuzzy generalization of OSF logic in which sorts symbols are interpreted as *fuzzy sets*. Similarly to how OSF logic is based on a lattice of sort symbols, fuzzy OSF logic requires its sorts symbols to be ordered in a *fuzzy lattice*. Chapter 5, on the other hand, deals with an extension of the language of OSF logic with a similarity relation on its sort symbols, which is modeled as a *fuzzy equivalence relation*. All the definitions needed for the development of fuzzy OSF logic and similarity-based OSF logic are defined in Section 1.2, which follows the theory of fuzzy sets from [44, 68].

1.1 Lattices and orders

We recall a few essential definitions from order theory that are needed for the development of OSF logic in Chapter 3. We generally assume that the orders mentioned in this document are *finite*.

Definition 1.1 (Preorder). A binary relation $\preceq$ on a set X is called a *preorder* if it

satisfies:

$$\forall x \in X,\ x \preceq x, \tag{Reflexivity}$$

$$\forall x, y, z \in X,\ \text{if } x \preceq y \text{ and } y \preceq z,\ \text{then } x \preceq z. \tag{Transitivity}$$

A *preorder* constitutes the simplest ordering structure that may be given to a set. While a preorder $(X, \preceq)$ may contain distinct elements $x, y \in X$ such that $x \preceq y$ and also $y \preceq x$, a *partial order* is a refinement of a preorder that imposes the additional constraint whereby, whenever such a bidirectional relation occurs, the elements x and y must be identical.

Definition 1.2 (Partial order). A binary relation $\preceq$ on a set X is called a *partial order* if it satisfies (Reflexivity) and (Transitivity) and

$$\forall x, y \in X,\ \text{if } x \preceq y \text{ and } y \preceq x,\ \text{then } x = y. \tag{Antisymmetry}$$

A partial order is *bounded* if $\exists \top, \bot \in X$ such that, for all $x \in X$: $\bot \preceq x \preceq \top$. The pair $(X, \preceq)$ is also called a *partially ordered set* (*poset*).

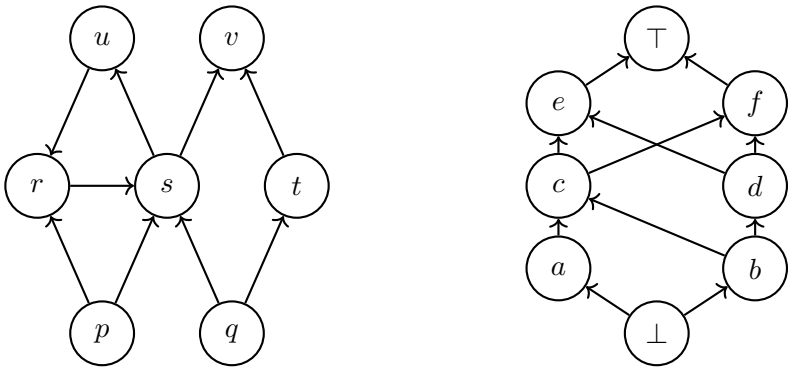
A preorder $(X, \preceq)$ can be represented as a directed graph (X, E) – where $E \subseteq X^2$ is a set of edges on X – such that $\preceq$ is the reflexive and transitive closure of E. Analogously, a poset $(X, \preceq)$ can be represented as a directed acyclic graph (DAG) (X, E) such that $\preceq$ is the reflexive and transitive closure of E.

Definition 1.3 (Composition of two binary relations). Let $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ be two binary relations. The *composition of R and S* is the binary relation defined as $R \circ S \stackrel{\text{def}}{=} \{(x, z) \in X \times Z \mid \exists y \in Y \text{ s.t. } (x, y) \in R, (y, z) \in S\}$. Let Q be a binary relation on X. The n-th composition of Q with itself is defined inductively by letting $Q^1 \stackrel{\text{def}}{=} Q$, and $Q^n \stackrel{\text{def}}{=} Q \circ Q^{n-1}$ for $n > 1$.

Definition 1.4 (Reflexive and transitive closure of a relation). Let R be a binary relation on a set X. The *transitive closure* of R is the binary relation $R^+ \stackrel{\text{def}}{=} \bigcup_{n \geq 1} R^n$. The *reflexive and transitive closure* of R is the binary relation $R^* \stackrel{\text{def}}{=} R^+ \cup \{(x, x) \mid x \in X\}$.

Example 1.5 (Preorders and partial orders). Consider the graph (X, E) represented in Fig. 1.1a. Its reflexive and transitive closure (X, E^*) is a preorder, but it does not satisfy antisymmetry since, for instance, $(u, s) \in E^*$ and $(s, u) \in E^*$, but $u \neq s$. This can also be verified visually, since the graph (X, E) contains a cycle involving the nodes u, r, and s. On the other hand, the reflexive and transitive closure of the graph represented in Fig. 1.1b is an example of a bounded partial order with least element $\bot$ and greatest element $\top$. ▷

Given a subset S of a partial order $(X, \preceq)$ it is often convenient to consider the elements of X that are above or below *every element of S*, i.e., the upper bounds and the lower bounds of S.



(a) Preorder (b) Partial order

Figure 1.1: A preorder and a partial order.

Definition 1.6 (Upper and lower bounds). Let $(X, \preceq)$ be a poset and $S \subseteq X$. The set of *lower bounds of* S is defined as $S^l_\preceq \stackrel{\text{def}}{=} \{x \in X \mid \forall s \in S, x \preceq s\}$, i.e., it is the set of elements of X that are below *every element of* S. The set of *upper bounds of* S is defined as $S^u_\preceq \stackrel{\text{def}}{=} \{x \in X \mid \forall s \in S, s \preceq x\}$, i.e., it is the set of elements of X that are above *every element of* S.

Analogously, it is also convenient to consider the set of elements that are above or below *some elements of* S, i.e., the up-set and the down-set of S.

Definition 1.7 (Up-sets and down-sets). Let $(X, \preceq)$ be a poset. The *down-set of an element* $x \in X$ is defined as $x{\downarrow} \stackrel{\text{def}}{=} \{y \in X \mid y \preceq x\}$, while its *up-set* is defined as $x{\uparrow} \stackrel{\text{def}}{=} \{y \in X \mid y \succeq x\}$. The *down-set of a subset* $S \subseteq X$ is defined as $S{\downarrow} \stackrel{\text{def}}{=} \{x \in X \mid \exists s \in S, x \preceq s\}$, while its *up-set* is defined as $S{\uparrow} \stackrel{\text{def}}{=} \{x \in X \mid \exists s \in S, s \preceq x\}$.

Note that $\{x\}^l_\preceq = x{\downarrow}$, $\{x\}^u_\preceq = x{\uparrow}$, $S^u_\preceq = \bigcap_{s \in S} s{\uparrow}$, and $S^l_\preceq = \bigcap_{s \in S} s{\downarrow}$.

OSF logic requires its set of sorts symbols $\mathcal{S}$ to be ordered in a finite bounded lattice $(\mathcal{S}, \preceq)$ which models a concept subsumption relation. In other words, for each pair of sorts $s_0, s_1 \in \mathcal{S}$ there must exist a sort s such that $s \preceq s_0$ and $s \preceq s_1$ – that is, s is a *lower bound*, or a common subsort, of s_0 and s_1 – and s is the most general sort with this property – that is, s is the *greatest lower bound* (GLB) of s_0 and s_1. Finding the GLB of two sorts is one of the core operations in reasoning with OSF logic, as will be seen in Chapter 3. Lattices are formally defined next.

Definition 1.8 (Greatest lower bound and least upper bound). Let $(X, \preceq)$ be a poset and $S \subseteq X$. The *greatest lower bound (GLB) of* S, denoted $\bigwedge S$, is the unique $x \in X$ such that $x \in S^l_\preceq$ and, for all $y \in S^l_\preceq$, $y \preceq x$. The *least upper bound (LUB) of* S, denoted $\bigvee S$, is the unique $x \in X$ such that $x \in S^u_\preceq$ and, for all $y \in S^u_\preceq$, $x \preceq y$.

If $S = \{x, y\}$, then $\bigwedge S$ is also written as $x \wedge y$, and $\bigvee S$ is also written as $x \vee y$. In a partial order $(X, \preceq)$, GLBs and LUBs are not guaranteed to exist for every subset $S \subseteq X$.

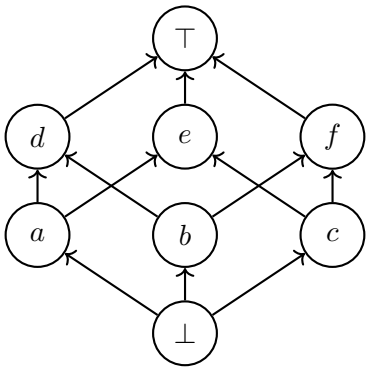


Figure 1.2: A lattice.

Definition 1.9 (Lattice). A partial order $(X, \preceq)$ is a *lattice* if the GLB $x \wedge y$ and the LUB $x \vee y$ exist for every pair of elements $x, y \in X$. A lattice is *complete* if the GLB $\bigwedge S$ and the LUB $\bigvee S$ exist for every subset $S \subseteq X$.

Note that every finite non-empty lattice is complete.

Example 1.10 (Lattice). The poset represented in Fig. 1.1b is not a lattice since, for example, the nodes e and f do not have a GLB. An example of a lattice is represented in Fig. 1.2. ▷

Although OSF logic formally requires its set of sort symbols to constitute a finite bounded lattice, it is often more practical to simply work with a partial order. For instance, when modeling a subsumption relation, the choice of a partial order may be convenient to avoid explicitly specifying a GLB for each pair of sort symbols. While two sorts may not have a GLB in a poset, it is nevertheless always possible to consider the set of their *maximal lower bounds* (MLBs).

Definition 1.11 (Minimal and maximal elements). Let $(X, \preceq)$ be a poset and $S \subseteq X$. The set of *minimal elements of* S is defined as $\lfloor S \rfloor \stackrel{\text{def}}{=} \{x \in S \mid \forall y \in S(\text{ if } y \preceq x, \text{ then } x = y)\}$. The set of *maximal elements of* S is defined as $\lceil S \rceil \stackrel{\text{def}}{=} \{x \in S \mid \forall y \in S(\text{ if } x \preceq y, \text{ then } x = y)\}$.

Definition 1.12 (Minimal upper bounds and maximal lower bounds). Let $(X, \preceq)$ be a poset and $S \subseteq X$. The set of *minimal upper bounds (MUBs) of* S is defined as $S^{mub}_{\preceq} \stackrel{\text{def}}{=} \left\lfloor S^{u}_{\preceq} \right\rfloor$ i.e., it is the set of upper bounds of all elements of S that are minimal with respect to $\preceq$. The set of *maximal lower bounds (MLBs) of* S is defined as $S^{mlb}_{\preceq} \stackrel{\text{def}}{=} \left\lceil S^{l}_{\preceq} \right\rceil$ i.e., it is the set of lower bounds of all elements of S that are maximal with respect to $\preceq$.

Example 1.13 (Maximal lower bounds). Consider the poset $(X, \preceq)$ represented in Fig. 1.1b. The set of maximal lower bounds of e and f is $\{c, d\}$. ▷

The idea of considering MLBs when working with posets rather than lattices is made formal by the construction of a lattice on the *antichains* of a partial order $(X, \preceq)$, where the GLB of two antichains of shape $\{s_0\}$ and $\{s_1\}$ corresponds exactly to the set of MLBs of s_0 and s_1 in $(X, \preceq)$. The following definition of this construction is based on [13].

Definition 1.14 (Antichain). Let $(X, \preceq)$ be a poset. Two elements $x, y \in X$ are said to be *incomparable* – denoted $x \parallel y$ – if $x \not\preceq y$ and $y \not\preceq x$. An *antichain* is a subset $C \subseteq X$ such that, for all $x, y \in C$, $x \parallel y$ if $x \neq y$. The set of all antichains of $(X, \preceq)$ is denoted $\mathit{Antichains}(X)$.

Note that, for any $S \subseteq X$, $\lfloor S \rfloor$ and $\lceil S \rceil$ are antichains.

The set of antichains of a partial order $(X, \preceq)$ can be endowed with an ordering that extends the one on $(X, \preceq)$.

Definition 1.15 (Antichain ordering). Let $(X, \preceq)$ be a poset. The partial order $\preceq$ on X can be extended to a *partial order* $\preceq$ *on* $\mathit{Antichains}(X)$ by letting, for all $C, C' \in \mathit{Antichains}(X)$:

$$C \preceq C' \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall x \in C \; \exists x' \in C' \text{ such that } x \preceq x'.$$

The same symbol $\preceq$ is used for the partial order on X and the partial order on $\mathit{Antichains}(X)$, since its meaning is always clear from context.

Equivalent definitions of the ordering on $\mathit{Antichains}(X)$ can be given as follows.

Proposition 1.16 (Equivalent definition of the antichain ordering). *Let $(X, \preceq)$ be a poset. For all $C, D \in \mathit{Antichains}(X)$:*

$$(i)\ C \preceq D \;\Leftrightarrow\; (ii)\ C{\downarrow} \subseteq D{\downarrow} \;\Leftrightarrow\; (iii)\ C \subseteq D{\downarrow}.$$

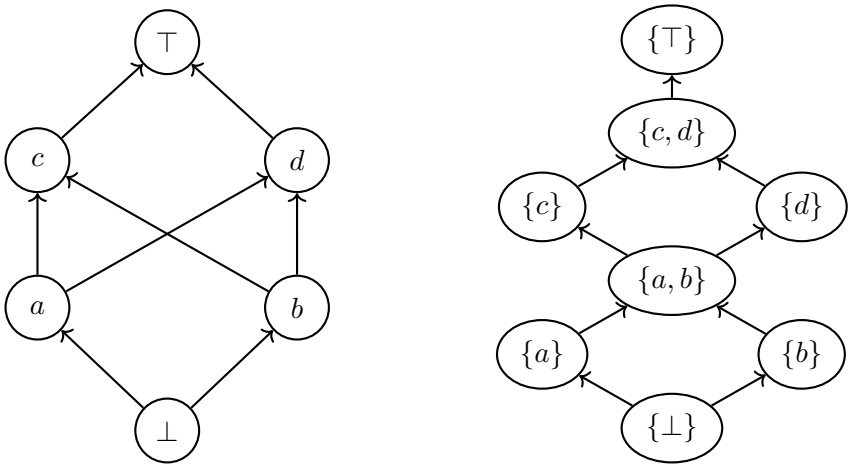
The following proposition states that $(\mathit{Antichains}(X), \preceq)$ is a *completion* of $(X, \preceq)$, i.e., $(\mathit{Antichains}(X), \preceq)$ is a complete lattice that extends $(X, \preceq)$ in the sense that there exists an *order embedding* of $(X, \preceq)$ into $(\mathit{Antichains}(X), \preceq)$, that is, a function $f : X \to \mathit{Antichains}(X)$ such that, for all $x, y \in X$, $x \preceq y \Leftrightarrow f(x) \preceq f(y)$.

Proposition 1.17 (Antichain lattice). *Let $(X, \preceq)$ be a (bounded) poset.*

1. *$(\mathit{Antichains}(X), \preceq)$ from Definition 1.15 is a (bounded) lattice, where GLBs and LUBs are defined by letting, for $C_0, C_1 \in \mathit{Antichains}(X)$: $C_0 \wedge C_1 \overset{\text{def}}{=} \lceil C_0{\downarrow} \cap C_1{\downarrow} \rceil$ and $C_0 \vee C_1 \overset{\text{def}}{=} \lfloor C_0{\uparrow} \cap C_1{\uparrow} \rfloor$.*

2. *$(\mathit{Antichains}(X), \preceq)$ is a completion of $(X, \preceq)$ as witnessed by the order embedding $f : X \to \mathit{Antichains}(X)$ defined by letting $f(x) = \{x\}$ for each $x \in X$.*

3. *Existing GLBs and LUBs in $(X, \preceq)$ are preserved in $(\mathit{Antichains}(X), \preceq)$: for all $x, y \in X$, if $z = x \wedge y$, then $\{z\} = \{x\} \wedge \{y\}$.*



(a) Partial order (b) Antichain completion

Figure 1.3: The partial order and its antichain completion of Example 1.18.

Example 1.18 (Antichain lattice). Consider the partial order $(X, \preceq)$ represented in Fig. 1.3a, which is not a lattice since the elements c and d do not have a GLB. Fig. 1.3b represent the completion $(\mathit{Antichains}(X), \preceq)$ of $(X, \preceq)$ as defined in Definition 1.15. Note that, in particular, the GLB of $\{c\}$ and $\{d\}$ in $(\mathit{Antichains}(X), \preceq)$ is $\{a, b\}$, which is the set of MLBs of c and d in $(X, \preceq)$. ▷

We conclude the section by recalling a well known construction of a partial order from a preorder $(X, \preceq)$, based on the idea of merging in an equivalence class $[x]$ all the elements $y \in X$ such that $x \preceq y$ and $y \preceq x$. A fuzzy generalization of this result will be a key component in the development of similarity-based reasoning with OSF logic in Chapter 5.

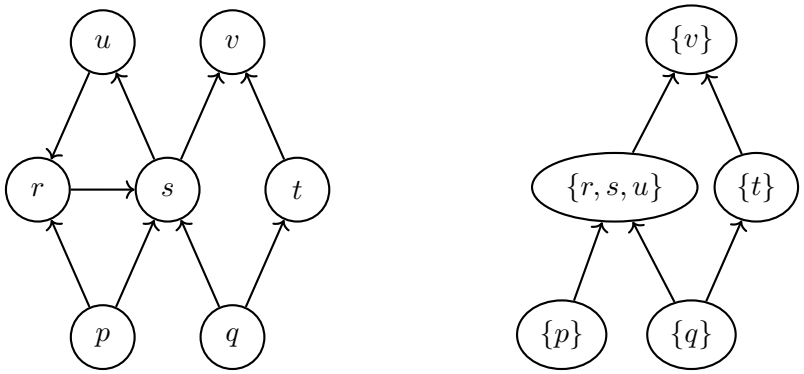
Definition 1.19 (Equivalence relation). A binary relation $\approx$ on a set X is called an *equivalence relation* if it satisfies (Reflexivity), (Transitivity) and

$$\forall x, y \in X, \text{ if } x \approx y, \text{ then } y \approx x. \qquad \text{(Symmetry)}$$

The *equivalence class* of $x \in X$ is the set $[x]^{\approx} \stackrel{\text{def}}{=} \{y \in X \mid x \approx y\}$. The superscript in $[x]^{\approx}$ is dropped when $\approx$ is clear from context. The set of equivalence classes $X_{/\approx} \stackrel{\text{def}}{=} \{[x]^{\approx} \mid x \in X\}$ forms a partition of X.

Proposition 1.20 (Partial order on the quotient set of a preorder). *Every preorder $(X, \preceq)$ induces an equivalence relation $\approx$ on X defined by letting, for all $x, y \in X$, $x \approx y \stackrel{\text{def}}{\Leftrightarrow} x \preceq y$ and $y \preceq x$. The preorder $\preceq \subseteq X \times X$ can be extended to partial order $\preceq \subseteq X_{/\approx} \times X_{/\approx}$ by letting, for all $x, y \in X$, $[x]^{\approx} \preceq [y]^{\approx} \stackrel{\text{def}}{\Leftrightarrow} \exists x' \in [x]^{\approx}, \exists y' \in [y]^{\approx}$ s.t. $x' \preceq y'$.*

The same symbol $\preceq$ is used for the preorder on X and the partial order on $X_{/\approx}$, since its meaning is always clear from context.



(a) Preorder (b) Partial order

Figure 1.4: The preorder and partial order of Example 1.21.

Example 1.21 (Partial order on the quotient set of a preorder). Fig. 1.4b represents the partial order constructed on the equivalence classes of the elements of the preorder represented in Fig. 1.4a according to Proposition 1.20. ▷

1.2 Fuzzy set theory and fuzzy orders

In this section we recall the basic definitions of fuzzy set theory [44, 68] that are necessary for the development of fuzzy OSF logic and similarity-based OSF logic in Chapter 4 and Chapter 5. Whenever possible, we employ the same notation for fuzzy sets and fuzzy orders as the one used for crisp sets and orders, but with the addition of a small dot ($\cdot$) to avoid ambiguity. For example, the symbol for the intersection of two crisp sets is $\cap$, while the symbol for the intersection of two fuzzy sets is $\cdot\cap$.

1.2.1 Fuzzy sets

Every subset C of a set X is associated with a characteristic function $\mathbf{1}_C : X \to \{0, 1\}$ defined by letting

$$\mathbf{1}_C(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x \in C, \\ 0 & \text{otherwise.} \end{cases}$$

A fuzzy subset generalizes this definition by letting the value of a *membership function* range in the unit interval $[0, 1]$.

Definition 1.22 (Fuzzy subset, support and β-cut). A *fuzzy subset* F of a (crisp) set X is determined by its membership function $\mu_F : X \to [0, 1]$. A fuzzy subset F of X can also be represented as

$$F = \{\beta/x \mid \mu_F(x) = \beta\}.$$

Name	t-norm	t-conorm
Gödel	$\beta_0 \wedge \beta_1 \stackrel{\text{def}}{=} \min(\beta_0, \beta_1)$	$\beta_0 \vee \beta_1 \stackrel{\text{def}}{=} \max(\beta_0, \beta_1)$
Product	$\beta_0 \wedge_p \beta_1 \stackrel{\text{def}}{=} \beta_0 \cdot \beta_1$	$\beta_0 \vee_p \beta_1 \stackrel{\text{def}}{=} \beta_0 + \beta_1 - \beta_0 \cdot \beta_1$
Łukasiewicz	$\beta_0 \wedge_l \beta_1 \stackrel{\text{def}}{=} \max(\beta_0 + \beta_1 - 1, 0)$	$\beta_0 \vee_l \beta_1 \stackrel{\text{def}}{=} \min(\beta_0 + \beta_1, 1)$

Table 1.1: Examples of t-norms and t-conorms.

The *support* of F is $|F| \stackrel{\text{def}}{=} \{x \in X \mid \mu_F(x) > 0\}$, while the *$\beta$-cut*[1] *of* F ($\beta \in [0,1]$) is $|F|_\beta \stackrel{\text{def}}{=} \{x \in X \mid \mu_F(x) \geq \beta\}$.

The fuzzy generalizations of the intersection and the union of crisp subsets are based on operations known as triangular norms (t-norms) and triangular conorms (t-conorms). Common examples of these operations are given in Table 1.1. In this thesis we adopt the Gödel t-norm $\wedge$ (minimum) and t-conorm $\vee$ (maximum).

Definition 1.23 (Intersection and union of fuzzy subsets). Let X be a set and $\mathcal{F}$ be a set of fuzzy subsets of X. The *intersection* $\bigcap\!\!\!\cdot\, \mathcal{F}$ *and the union* $\bigcup\!\!\!\cdot\, \mathcal{F}$ *of* $\mathcal{F}$ are defined by letting

$$\mu_{\bigcap\!\!\!\cdot\, \mathcal{F}}(x) \stackrel{\text{def}}{=} \inf(\{\mu_F(x) \mid F \in \mathcal{F}\}) \quad \text{and} \quad \mu_{\bigcup\!\!\!\cdot\, \mathcal{F}}(x) \stackrel{\text{def}}{=} \sup(\{\mu_F(x) \mid F \in \mathcal{F}\}).$$

Note that the latter definition employs inf and sup rather than min and max in order to be applicable to sets of fuzzy subsets of arbitrary cardinality. If $\mathcal{F}$ is finite, then clearly

$$\mu_{\bigcap\!\!\!\cdot\, \mathcal{F}}(x) = \bigwedge_{F \in \mathcal{F}} \mu_F(x) = \min(\{\mu_F(x) \mid F \in \mathcal{F}\}), \text{ and}$$
$$\mu_{\bigcup\!\!\!\cdot\, \mathcal{F}}(x) = \bigvee_{F \in \mathcal{F}} \mu_F(x) = \max(\{\mu_F(x) \mid F \in \mathcal{F}\}).$$

The literature on fuzzy sets also provides several definitions of the *complement* of a fuzzy set, based on a *fuzzy negation* operator. Two examples are the *standard (or Łukasiewicz) negation* and the *Gödel negation*, which are defined, respectively by letting

$$\ominus_l \beta \stackrel{\text{def}}{=} 1 - \beta \quad \text{and} \quad \ominus_g \beta \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \beta = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 1.24 (Complement of a fuzzy subset). Let X be a set, let F be a fuzzy subset of X, and let $\ominus$ be a fuzzy negation operator (e.g., $\ominus_g$ or $\ominus_l$). The *complement* $X \setminus F$ of F is defined by letting, for all $x \in X$, $\mu_{X \setminus F}(x) \stackrel{\text{def}}{=} \ominus \mu_F(x)$.

[1] Note that in the literature this is usually refereed to as an *α-cut*, but we prefer to use the name β-cut so as to avoid notational clashes in later chapters, where α is used as the symbol for variable assignments.

Name	Implication
Gödel	$\beta_0 \Rightarrow_g \beta_1 \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \beta_0 \leq \beta_1 \\ \beta_1 & \text{otherwise} \end{cases}$
Goguen (or product)	$\beta_0 \Rightarrow_p \beta_1 \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \beta_0 \leq \beta_1 \\ \frac{\beta_1}{\beta_0} & \text{otherwise} \end{cases}$
Łukasiewicz	$\beta_0 \Rightarrow_l \beta_1 \stackrel{\text{def}}{=} \min(1 - \beta_0 + \beta_1, 1)$
Kleene Dienes	$\beta_0 \Rightarrow_{kd} \beta_1 \stackrel{\text{def}}{=} \max(1 - \beta_0, \beta_1)$

Table 1.2: Examples of fuzzy implication operators.

Finally, an *implication operator* can be defined starting from a t-norm, or from a t-conorm and a fuzzy negation operator, generalizing analogous formulas from crisp propositional logic.

Definition 1.25 (R-implication and s-implication). Let $\wedge_t$ be a t-norm, let $\vee_t$ be a t-conorm, and let $\ominus$ be a fuzzy negation operator. The *r-implication* $\Rightarrow$ with respect to $\wedge_t$ is defined by letting, for all $\beta_0, \beta_1 \in [0, 1]$:

$$\beta_0 \Rightarrow \beta_1 \stackrel{\text{def}}{=} \sup\{\beta \mid \beta_0 \wedge_t \beta \leq \beta_1\}.$$

The *s-implication* $\Rightarrow$ with respect to $\vee_t$ and $\ominus$ is defined by letting, for all $\beta_0, \beta_1 \in [0, 1]$: $\beta_0 \Rightarrow \beta_1 \stackrel{\text{def}}{=} \ominus\beta_0 \vee_t \beta_1$.

A few examples of fuzzy implication operators are shown in Table 1.2. The implication $\Rightarrow_{kd}$ is an s-implication based on $\ominus_l$ and $\vee$, while $\Rightarrow_l$ is an s-implication based on $\ominus_l$ and $\vee_l$. The implications $\Rightarrow_g$, $\Rightarrow_l$, and $\Rightarrow_p$ are r-implications based on $\wedge$, $\wedge_l$, and $\wedge_p$, respectively.

We adopt the convention of identifying a fuzzy subset F with its membership function μ_F in order to make the notation less cumbersome.

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

1.2.2 Fuzzy orders

Fuzzy OSF logic is based on a fuzzy subsumption relation, i.e., a *fuzzy binary relation* that associates a subsumption degree in $[0, 1]$ with every pair of sort symbols.

Definition 1.26 (Fuzzy binary relation). A *fuzzy binary relation* R between the sets X and Y is a fuzzy subset of $X \times Y$, i.e., it is a function $R : X \times Y \to [0, 1]$.

Fuzzy preorders and fuzzy partial orders are generalization of preorders (Definition 1.1) and partial orders (Definition 1.2).

Definition 1.27 (Fuzzy preorder). A fuzzy binary relation R on a set X is a *fuzzy preorder* if it satisfies:

$$\forall x \in X,\ R(x,x) = 1, \qquad \text{(Fuzzy Reflexivity)}$$
$$\forall x,y,z \in X,\ R(x,z) \geq R(x,y) \wedge R(y,z). \qquad \text{(Max-Min Transitivity)}$$

Equivalently, (Max-Min Transitivity) can also be defined as follows:

$$\forall x,z \in X,\ R(x,z) \geq \bigvee_{y \in X} (R(x,y) \wedge R(y,z)).$$

Definition 1.28 (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

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

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

A fuzzy preorder (X,R) can be represented as a weighted directed graph (X,E,λ_E) (where $E \subseteq X^2$ is a set of edges and $\lambda_E : E \to [0,1]$ labels each edge with a value in $[0,1]$) such that R is the reflexive and transitive closure of λ_E. Analogously, a fuzzy poset (X,R) can be represented as a weighted DAG (X,E,λ_E) such that R is the reflexive and transitive closure of λ_E.

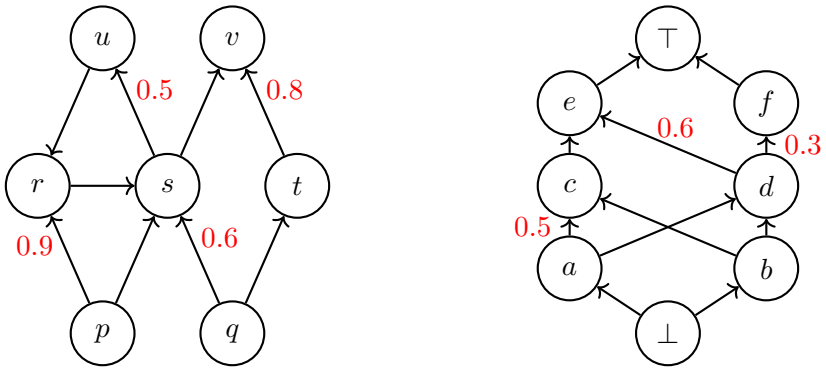
Definition 1.29 (Composition of fuzzy binary relations). The *(max-min) composition* of two fuzzy binary relations R and Q on a finite set X is the fuzzy binary relation $R \circ Q$ defined by the membership function

$$R \circ Q(x,z) \stackrel{\text{def}}{=} \bigvee_{y \in X} (R(x,y) \wedge Q(y,z)).$$

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 1.30 (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.

Example 1.31 (Fuzzy preorder and fuzzy partial order). Consider the weighted directed graph (X,E,λ_E) of Fig. 1.5a. Edges with no weight are assumed to be implicitly



(a) Fuzzy preorder (b) Fuzzy partial order

Figure 1.5: A fuzzy preorder and a fuzzy partial order.

labeled with a 1. Let $R : X \times X \to [0, 1]$ be defined by letting, for all $x, y \in X$:

$$R(x, y) \stackrel{\text{def}}{=} \begin{cases} \lambda_E^{\circledast}(x, y) & \text{if } (x, y) \in E^*, \\ 0 & \text{otherwise.} \end{cases}$$

The pair (X, R) is a fuzzy preorder. Note that antisymmetry is not satisfied, since, for instance, $R(r, u) = 0.5$ and $R(u, r) = 1$, but $r \neq u$. This can also be seen from the graph representation of (X, R), since u and r are part of a cycle together with the node s.

In a similar way, the weighted graph of Fig. 1.5b represents a fuzzy partial order. ▷

We introduce the following conventions regarding infix notation for fuzzy binary relations.

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 $R(x, y) > 0$.

1.2.3 Fuzzy similarity relations

Several approaches to approximate reasoning based on fuzzy relations such as *proximities* and *similarities* have been proposed in the fuzzy logic programming literature. The goal of this research, which is briefly reviewed in Section 2.2, is to make query answering from knowledge bases more flexible, by allowing the retrieval of approximate (not exact, but similar) solutions to a query. In Chapter 5 we develop an extension of OSF logic based on a similarity relation on the set of sort symbols, which will enable approximate reasoning with this language. Proximity relations and similarity relations are defined next.

Definition 1.32 (Proximity relation). A fuzzy binary relation $\sim$ on a set X is a *proximity*

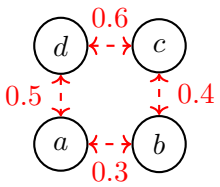


Figure 1.6: Representation of a proximity or a similarity relation.

if it satisfies (Fuzzy Reflexivity) and

$$\forall x, y \in X, {\sim}(x, y) = {\sim}(y, x). \qquad \text{(Fuzzy Symmetry)}$$

Definition 1.33 (Similarity relation). A fuzzy binary relation $\sim$ on a set X is a *similarity* if it satisfies (Fuzzy Reflexivity), (Fuzzy Symmetry) and (Max-Min Transitivity).

A proximity relation $\sim$ can be represented as a weighted directed graph of which $\sim$ is the reflexive closure. A similarity relation $\sim$ can be represented as a weighted directed graph, of which $\sim$ is the reflexive and transitive closure.

Example 1.34 (Proximity and similarity relations). Consider the weighted directed graph (X, E, λ_E) of Fig. 1.6. Let $\sim\colon X \times X \to [0, 1]$ be defined by letting, for all $(x, y) \in X$:

$${\sim}(x, y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = y, \\ \lambda_E(x, y) & \text{else, if } (x, y) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Then $\sim$ is a proximity relation on X. Note that $\sim$ does not satisfy transitivity since, for instance, ${\sim}(a, b) = 0.3$ and ${\sim}(b, c) = 0.4$, but ${\sim}(a, c) = 0$. The transitive closure ${\sim}^{\oplus}$ of $\sim$ is a similarity relation such that, for example, ${\sim}^{\circledast}(a, c) = 0.5$. ▷

The following propositions provide a connection between the fuzzy binary relations defined in this section and their crisp counterparts of Section 1.1.

Proposition 1.35 (Crisp and fuzzy properties of binary relations). *Let R be a fuzzy binary relation on a set X.*

1. *If R satisfies* (Fuzzy Reflexivity)*, then $|R|$ satisfies* (Reflexivity)*.*
2. *If R satisfies* (Max-Min Transitivity)*, then $|R|$ satisfies* (Transitivity)*.*
3. *If R satisfies* (Strong Fuzzy Antisymmetry)*, then $|R|$ satisfies* (Antisymmetry)*.*
4. *If R satisfies* (Fuzzy Symmetry)*, then $|R|$ satisfies* (Symmetry)*.*

Corollary 1.36 (Support of fuzzy binary relations). *Let R be a fuzzy binary relation on a set X.*

1. *If R is a fuzzy preorder, then $|R|$ is a preorder.*
2. *If R is a fuzzy partial order, then $|R|$ is a partial order.*
3. *If R is a fuzzy similarity, then $|R|$ is an equivalence relation.*

1.2.4 Fuzzy lattices

Finally, we recall the definition of a fuzzy lattice [36, 79, 80], which will be central in our fuzzy generalization of OSF logic in Chapter 4.

Definition 1.37 (Lower and upper bounds in a fuzzy poset). Let $(X, \preceq)$ be a fuzzy poset and $S \subseteq X$. The set of *lower bounds of S* is defined as $S^{fl}_{\preceq} \stackrel{\text{def}}{=} \{x \in X \mid \forall x' \in S, x \preceq x'\}$, and the set of *upper bounds of S* is defined as $S^{fu} \stackrel{\text{def}}{=} \{x \in X \mid \forall x' \in S, x' \preceq x\}$[2].

Definition 1.38 (Maximal elements and down-sets in a fuzzy poset). Let $(X, \preceq)$ be a fuzzy poset and $S \subseteq X$. The set of *maximal elements of S* is defined as $\lceil S \rceil^{\cdot} \stackrel{\text{def}}{=} \{x \in S \mid \forall x' \in S(\text{ if } x \preceq x', \text{ then } x = x')\}$. The *(fuzzy) down-set of S* is defined as the set $S{\downarrow} \stackrel{\text{def}}{=} \{x \in X \mid \exists x' \in S, x \preceq x'\}$.

Definition 1.39 (Fuzzy GLB and LUB). Let $(X, \preceq)$ be a fuzzy poset and $S \subseteq X$. The *GLB of S* is the unique $x \in S^{fl}_{\preceq}$ such that, for all $x' \in S^{fl}_{\preceq}$, $x' \preceq x$. The *LUB of S* is the unique $x \in S^{fu}$ such that, for all $x' \in S^{fu}$, $x \preceq x'$. If the GLB of S exists, it is denoted $\bigwedge S$, or simply $x \wedge x'$ in case $S = \{x, x'\}$. Similarly, if the LUB of S exists, it is denoted $\bigvee S$, or simply $x \vee x'$ in case $S = \{x, x'\}$.

Definition 1.40 (Fuzzy lattice and bounded lattice). A fuzzy poset $(X, \preceq)$ is a *fuzzy lattice* if every pair of its elements has a GLB and a LUB. A fuzzy lattice $(X, \preceq)$ is *bounded* if there are elements $\bot, \top \in X$ such that, for all $x \in X$, $\preceq(\bot, x) = 1$ and $\preceq(x, \top) = 1$.

Example 1.41 (Fuzzy lattice). Fig. 1.7 depicts the weighted graph representation (X, E, λ_E) of a fuzzy bounded lattice $(X, \preceq)$, where edges with no weight are assumed to be implicitly labeled with a 1. The fuzzy lattice can be formally defined by letting, for all $x, y \in X$,

$$\preceq(x, y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = \bot \text{ or } y = \top, \\ \lambda^{\circledast}_E(x, y) & \text{else, if } (x, y) \in E^*, \\ 0 & \text{otherwise.} \end{cases}$$

[2] Recall that $x \preceq x'$ is an abbreviation for $\preceq(x, x') > 0$.

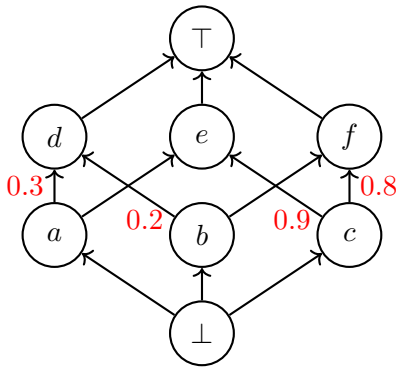


Figure 1.7: A fuzzy lattice.

This definition ensures that the boundedness condition is satisfied by $\preceq$, without the need to add extra edges to its graph representation, as simply taking the transitive closure of λ_E might not be sufficient in general. In this case, for instance, $\lambda_E^{\circledast}(\bot, d) < 1$. ▷

The following proposition states that the support $(X, |\preceq|)$ of a fuzzy lattice $(X, \preceq)$ is a lattice. Additionally, if x is the GLB of x_0 and x_1 in $(X, \preceq)$, then x is also the GLB of x_0 and x_1 in $(X, |\preceq|)$, and vice versa. This property ensures that the computation of GLBs in a fuzzy lattice can be reduced to the crisp setting. For instance, it would be possible to employ the techniques of [5, 13] on the weighted graph representation of a fuzzy lattice simply by ignoring the edge weights, thus preserving the same computational complexity. The computation of GLBs in a fuzzy lattice and of the subsumption degree $\preceq(x, x')$ will be discussed in Section 4.8, in the context of fuzzy OSF logic.

Proposition 1.42 (Fuzzy and crisp lattices). *If $(X, \preceq)$ is a fuzzy lattice, then $(X, |\preceq|)$ is a (crisp) lattice. Moreover, if $\wedge$ is the GLB operation for $(X, |\preceq|)$, then $\bigwedge S = \bigwedge S$ for every subset $S \subseteq X$.*

Chapter 2

Related Work

Order-Sorted Feature (OSF) logic is one of many logical formalisms that have been researched in Knowledge Representation and Reasoning (KRR), the area of Artificial Intelligence concerned with how knowledge can be explicitly represented by symbols and manipulated in an automated way by reasoning algorithms in order to solve problems [31, 57]. This chapter provides a brief overview of a few KRR languages and approaches that are closely related to OSF logic and the research presented in this thesis. The presentation is meant to provide the necessary context for Chapters 3 to 5 and to facilitate comparisons, and it is not intended as a comprehensive survey.

Closely related to OSF logic are Description Logics (DLs) [18], as both languages are descendants of Ron Brachman's structured inheritance networks and were developed to overcome the lack of a formal semantics in earlier KRR formalisms such as frames and semantic networks [4, 18, 19, 31]. DLs allow to represent knowledge through *concept descriptions*, expressions built by applying *concept and role constructors* starting from unary predicates called *atomic concepts*, and from binary relations called *atomic roles*. One advantage of DLs is the possibility of choosing among a variety of constructors to build new concepts and roles starting from the atomic ones in order to achieve the desired trade-off between the reasoning complexity and the expressivity needed for the intended application. A brief overview of DLs is provided in Section 2.1.1.

In addition to addressing challenges related to complexity and scalability, several real-world applications need to handle uncertain, imprecise, or ambiguous knowledge, or could benefit significantly from this capability. Classical two-valued logic is inherently inadequate for addressing inconsistencies, uncertainty, or fuzziness. Extensions of DLs that can handle imperfect knowledge are being researched extensively (e.g., [30, 77] are two surveys on the matter), including probabilistic, possibilistic, and fuzzy DLs. Fuzzy DLs are briefly discussed in Section 2.1.2 in order to aid the comparison with our fuzzy generalization of OSF logic introduced in Chapter 4.

Another approach to approximate reasoning that is related to our research of Chapter 5

is *similarity-based reasoning* in logic programming. In general terms, this approach consists in enhancing the ordinary (crisp) representation of a Prolog knowledge base with the addition of a similarity relation between symbols. This allows, for instance, to relax the *equality* constraint on two functor symbols to a flexible constraint of similarity when unifying two first-order terms (FOTs). This kind of relaxed unification is generally referred to as *weak unification* (e.g., [101]). A similarity relation can also be considered between *predicate* symbols in order to perform *similarity-based SLD resolution* [101]. A few approaches to approximate reasoning in logic programming based on fuzzy relations are recalled in Section 2.2.

2.1 Description logics and fuzzy description logics

2.1.1 Description logics

DLs are a family of KRR languages that can be employed to represent the terminological knowledge of an application domain in a structured and formally well-understood way [18]. The knowledge of a domain can be captured in DL through *concept descriptions*, expressions built from unary predicates called *atomic concepts* (like *Person* or *Animal*) and binary relations called *atomic roles* (like *hasChild* or *hasParent*). More complex concepts and roles are built from the atomic ones by applying *concept and role constructors*. For instance, the expression $\exists \mathit{hasChild}.\top$ denotes the class of entities who have a child.

Depending on which constructors are allowed by a specific DL, a different trade-off between expressivity and computational complexity can be achieved. Indeed, most DLs restrict the language of FOL in order to be decidable [76]. The DL $\mathcal{AL}$ was introduced as a minimal language that is of practical interest [19, 100]. Negation in this DL is restricted to atomic concepts, and only the top concept $\top$, representing the whole application domain, is allowed under the scope of an existential quantification.

Definition 2.1 (The DL $\mathcal{AL}$). The *concept descriptions of* $\mathcal{AL}$ (denoted $C, D, \ldots$) are defined inductively as follows.

Top and bottom The top concept $\top$ and the bottom concept $\bot$ are $\mathcal{AL}$ concepts.

Atomic negation If A is an atomic concept, then A and $\neg A$ are $\mathcal{AL}$ concepts.

Conjunction If C and D are $\mathcal{AL}$ concepts, then so is $C \sqcap D$.

Universal restriction If R is an atomic role and C is an $\mathcal{AL}$ concept, then $\forall R.C$ is an $\mathcal{AL}$ concept.

Limited existential restriction If R is an atomic role, then $\exists R.\top$ is an $\mathcal{AL}$ concept.

For instance, in $\mathcal{AL}$ it is possible to represent the class of people who have a child, with the expression $Person \sqcap \exists hasChild.\top$.

The meaning of the concept descriptions of $\mathcal{AL}$ is provided by structures called *interpretations*. A DL *interpretation* is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that $\Delta^{\mathcal{I}}$ is a non-empty set (the domain of the interpretation) and $\cdot^{\mathcal{I}}$ is an interpretation function that assigns a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ to each atomic concept A, and a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each atomic role R [19].

Definition 2.2 (Semantics of $\mathcal{AL}$ [19]). Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. The interpretation function $\cdot^{\mathcal{I}}$ is extended to $\mathcal{AL}$ concept descriptions according to the following inductive definitions:

- $\top^{\mathcal{I}} \stackrel{\text{def}}{=} \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} \stackrel{\text{def}}{=} \emptyset$,
- $(\neg A)^{\mathcal{I}} \stackrel{\text{def}}{=} \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$,
- $(C \sqcap D)^{\mathcal{I}} \stackrel{\text{def}}{=} C^{\mathcal{I}} \cap D^{\mathcal{I}}$,
- $(\forall R.C)^{\mathcal{I}} \stackrel{\text{def}}{=} \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x) \subseteq C^{\mathcal{I}}\}$, and
- $(\exists R.\top)^{\mathcal{I}} \stackrel{\text{def}}{=} \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x) \neq \emptyset\}$,

where $R^{\mathcal{I}}(x) \stackrel{\text{def}}{=} \{y \in \Delta^{\mathcal{I}} \mid (x, y) \in R^{\mathcal{I}}\}$.

In DLs, the knowledge about a domain is usually organized in an *ontology* (or knowledge base) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, consisting of a terminological component $\mathcal{T}$ and an assertional component $\mathcal{A}$. The terminological component, also called the T-Box, involves axioms of shape $C \sqsubseteq D$ (concept inclusion) or $C \equiv D$ (concept equivalence or definition). A concept inclusion $C \sqsubseteq D$ and a concept equivalence $C \equiv D$ are satisfied in an interpretation $\mathcal{I}$ if, respectively, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $C^{\mathcal{I}} = D^{\mathcal{I}}$.

The assertional component of a DL knowledge base, or the A-Box, involves assertions about specific entities of the domain of interpretation. Assertions are expressions of shape $C(a)$ (concept assertion) and $R(a, b)$ (role assertion), where C is a concept, R is a role, and a and b are individuals. Individuals are interpreted as objects of the domain, i.e., $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and $b^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies* a concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it satisfies a role assertion $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Example 2.3 ($\mathcal{AL}$ knowledge base). Consider the atomic concepts *Movie*, *Horror*, *Comedy* and *ComedyHorror*, the individuals *alinda*, *celeste*, *psycho*, and *city_lights*, and the atomic role *likesMovie*. Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be the $\mathcal{AL}$ knowledge base defined as follows. The assertional component $\mathcal{A}$ of $\mathcal{K}$ contains the facts

$$likesMovie(alinda, psycho), likesMovie(celeste, city\_lights),$$
$$Horror(psycho), Comedy(city\_lights), \neg Horror(city\_lights),$$

while the T-Box $\mathcal{T}$ contains the concept inclusions $Horror \sqsubseteq Movie$ and $Comedy \sqsubseteq Movie$, and the concept definition $ComedyHorror = Horror \sqcap Comedy$.

An example of an $\mathcal{AL}$ concept description constructed with the language of this knowledge base is $\forall likesMovie.Horror$. The denotation of this concept under any interpretation $\mathcal{I}$ that satisfies $\mathcal{K}$ is the set containing all the elements of the domain that only like horror movies. For instance, it could contain the element $alinda^{\mathcal{I}}$, but it may never contain the element $celeste^{\mathcal{I}}$, since $(celeste^{\mathcal{I}}, city\_lights^{\mathcal{I}}) \in likesMovie^{\mathcal{I}}$ and $city\_lights^{\mathcal{I}} \notin Horror^{\mathcal{I}}$. ▷

Reasoning with DLs allows one to infer implicit knowledge from the knowledge that is explicitly expressed in the knowledge base [19]. The inference patterns supported by DLs are related to the *classification* of concepts and individuals, i.e., they consist in determining whether a concept C is a subconcept of another concept D, or verifying whether an individual a belongs to a given concept C. Concept classification allows to structure the terminology in a subsumption hierarchy, while individual classification provides information on the properties of an individual [19]. More specifically, the following are a few of the most common reasoning tasks related to DLs [18, 19].

- Deciding the *consistency of a concept* C with respect to a knowledge base $\mathcal{K}$, which amounts to deciding whether there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$.
- Deciding the *subsumption* of two concepts C and D with respect to $\mathcal{K}$, which amounts to deciding if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{K}$.
- Deciding the *equivalence* of two concepts C and D with respect to $\mathcal{K}$, which amounts to deciding their subsumption in the two directions.
- Deciding whether an individual a is an instance of a concept C with respect to $\mathcal{K}$, which amounts to deciding whether $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.
- Deciding whether a pair of individuals (a, b) is an instance of a role R with respect to $\mathcal{K}$, which amounts to deciding whether $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.

The complexity of these reasoning tasks varies depending on the concept constructors supported by a DL. For instance, in $\mathcal{ALC}$ – the extension of $\mathcal{AL}$ that supports the complement of any concept, not just atomic ones, and that also allows any concept under the scope of an existential restriction – concept satisfiability, concept subsumption, and A-Box consistency are PSPACE-complete problems [18]. In the same DL, deciding satisfiability with respect to a general T-Box is EXPTIME-complete, while the problem becomes PSPACE-complete if the T-Box is *definitorial*, i.e., it contains only unique and acyclic definitions [18]. As an example of how adding constructors to a DL can affect its reasoning complexity, deciding concept satisfiability in $\mathcal{ALC}$ with either *transitive roles* or *role hierarchies* is in PSPACE, but the problem becomes EXPTIME-hard with both constructors [18].

DLs have been implemented in several reasoners – such as FaCT++ [116], HermiT [52, 102], Pellet [105], TrOWL [115], RacerPro [56], and SnoRocket [75]. The most notable application of DLs is possibly the Semantic Web [18], an extension of the World Wide Web in which "information is given well-defined meaning, better enabling computers and people to work in cooperation" [25]. Indeed, by design, DLs provide the formal semantics for the Web Ontology Language (OWL), the ontology language developed by the World Wide Web Consortium [18]. An OWL ontology describes a domain of interest in terms of *classes* and *properties*, which correspond to DL concepts and roles, respectively, together with a set of *axioms* that assert, for example, inclusions between classes, like in a DL T-Box [18]. DLs and OWL have found applications in various fields, such as software engineering, medicine, biology, natural language processing, and database management [18, 19]. Several biomedical ontologies have been developed using DLs [18], such as the Biological Pathways Exchange ontology [98], the Galen ontology [92], the Foundational Model of Anatomy [53], and the National Cancer Institute thesaurus [58]. More details regarding the applications of DLs and OWL can be found, for instance, in [18, 19, 21].

2.1.2 Fuzzy description logics

Fuzzy DLs are extensions of DLs whose purpose is to model vagueness and imprecision in the real world, and to characterize notions that cannot be properly defined with a crisp predicate, such as the class of tall people [18, 30]. The following presentation of fuzzy DLs is based on [18, 30, 77, 113].

Fuzzy DLs maintain the same basic syntax of DLs, but the semantics of DL expressions is now defined through t-norms, t-conorms, fuzzy negations and implications (see Section 1.2). A *fuzzy interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ assigns a fuzzy subset $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0,1]$ to each atomic concept A, and a fuzzy binary relation $R^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0,1]$ to each atomic role R. The interpretation of an individual a is the same as in crisp DLs, i.e., $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The semantics is extended to the concept descriptions of the DL $\mathcal{ALC}$ as follows.

Definition 2.4 (Fuzzy semantics of $\mathcal{ALC}$). Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a fuzzy interpretation, let $\wedge_t$ be a t-norm, let $\ominus$ be a fuzzy negation, and let $\Rightarrow$ be a fuzzy implication. The interpretation function $\cdot^{\mathcal{I}}$ is extended to $\mathcal{ALC}$ concept descriptions according to the following inductive definitions[1]:

- $\top^{\mathcal{I}} \stackrel{\text{def}}{=} \mathbf{1}_{\Delta^{\mathcal{I}}}$ and $\bot^{\mathcal{I}} \stackrel{\text{def}}{=} \mathbf{1}_{\emptyset}$,
- $(\neg C)^{\mathcal{I}}(x) \stackrel{\text{def}}{=} \ominus C^{\mathcal{I}}(x)$ for all $x \in \Delta^{\mathcal{I}}$,
- $(C \sqcap D)^{\mathcal{I}}(x) \stackrel{\text{def}}{=} C^{\mathcal{I}}(x) \wedge_t D^{\mathcal{I}}(x)$ for all $x \in \Delta^{\mathcal{I}}$,

[1]The symbol $\mathbf{1}_D$ denotes the characteristic function $\mathbf{1}_D : \Delta^{\mathcal{I}} \to \{0,1\}$ of the set $D \subseteq \Delta^{\mathcal{I}}$, which is defined by letting, for all $d \in \Delta^{\mathcal{I}}$, $\mathbf{1}_D(d) \stackrel{\text{def}}{=} 1$ if $d \in D$, and $\mathbf{1}_D(d) \stackrel{\text{def}}{=} 0$ otherwise.

- $(\forall R.C)^{\mathcal{I}}(x) \stackrel{\text{def}}{=} \inf_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \Rightarrow C^{\mathcal{I}}(y)$ for all $x \in \Delta^{\mathcal{I}}$, and
- $(\exists R.C)^{\mathcal{I}}(x) \stackrel{\text{def}}{=} \sup_{y \in \Delta^{\mathcal{I}}} R^{\mathcal{I}}(x,y) \wedge_t C^{\mathcal{I}}(y)$ for all $x \in \Delta^{\mathcal{I}}$.

The terminological component $\mathcal{T}$ of a fuzzy $\mathcal{ALC}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is composed of *fuzzy concept inclusion* axioms of shape $(C \sqsubseteq D, \beta)$, i.e., concept inclusions associated with a truth degree $\beta \in [0,1]$. Such an axiom $(C \sqsubseteq D, \beta)$ is satisfied by a fuzzy interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ if, for every $x \in \Delta^{\mathcal{I}}$, it holds that $C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x) \geq \beta$. The assertional component $\mathcal{A}$ of $\mathcal{K}$ is made up of fuzzy concept assertions of shape $(C(a), \beta)$, and fuzzy role assertions of shape $(R(a,b), \beta)$, where C is a concept, R is a role, a and b are individuals, and $\beta \in [0,1]$. A fuzzy concept assertion $(C(a), \beta)$ is satisfied in a fuzzy interpretation $\mathcal{I}$ if $C^{\mathcal{I}}(a^{\mathcal{I}}) \geq \beta$, and a fuzzy role assertion $(R(a,b), \beta)$ is satisfied in $\mathcal{I}$ if $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \beta$.

Example 2.5 (Fuzzy DL concept inclusion). Consider the atomic concepts *Movie*, *Horror*, *Slasher*, and *Thriller*, and the individuals *halloween*, *memento*, and *psycho*. Suppose that $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a fuzzy interpretation such that $\Delta^{\mathcal{I}} = \{h, p, m\}$, $halloween^{\mathcal{I}} = h$, $psycho^{\mathcal{I}} = p$, $memento^{\mathcal{I}} = m$, and

- $Thriller^{\mathcal{I}}(h) = 0.5$, and $Horror^{\mathcal{I}}(h) = Slasher^{\mathcal{I}}(h) = 1$;
- $Thriller^{\mathcal{I}}(p) = Horror^{\mathcal{I}}(p) = 1$ and $Slasher^{\mathcal{I}}(p) = 0.7$;
- $Thriller^{\mathcal{I}}(m) = 1$;
- $Movie^{\mathcal{I}}(x) = 1$ for every $x \in \Delta^{\mathcal{I}}$; and
- all the remaining membership degrees are equal to 0.

Adopting $\Rightarrow_g$ (see Table 1.2) as the fuzzy inclusion operator, the interpretation $\mathcal{I}$ satisfies the following fuzzy concept inclusion axioms:

- $(C \sqsubseteq Movie, 1)$ for $C \in \{Thriller, Slasher, Horror\}$, since $C^{\mathcal{I}}(x) \Rightarrow_g Movie^{\mathcal{I}}(x) = 1$ for all $x \in \Delta^{\mathcal{I}}$;
- $(Slasher \sqsubseteq Horror, 1)$, since $Slasher^{\mathcal{I}}(x) \Rightarrow_g Horror^{\mathcal{I}}(x) = 1$ for all $x \in \Delta^{\mathcal{I}}$;
- $(Slasher \sqsubseteq Thriller, 0.5)$, since in particular $Slasher^{\mathcal{I}}(h) \Rightarrow_g Thriller^{\mathcal{I}}(h) = 0.5$
- $(Horror \sqsubseteq Thriller, 0.5)$, since in particular $Horror^{\mathcal{I}}(h) \Rightarrow_g Thriller^{\mathcal{I}}(h) = 0.5$.
- $(Thriller \sqsubseteq C, 0)$ for $C \in \{Slasher, Horror\}$, since $Thriller^{\mathcal{I}}(m) \Rightarrow_g C^{\mathcal{I}}(m) = 0$. ▷

Fuzzy DLs support several reasoning problems, including the following generalizations of decision problems of crisp DLs [113].

- Deciding the *consistency of a concept* C with respect to a knowledge base $\mathcal{K}$, which amounts to deciding whether there is a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{K}$ such that $C^{\mathcal{I}}(x) > 0$ for some $x \in \Delta^{\mathcal{I}}$.
- Deciding whether the *subsumption degree* of two concepts C and D with respect to $\mathcal{K}$ is always greater than a given truth degree β, which amounts to deciding if $(C \sqsubseteq D, \beta)$ is satisfied in all models $\mathcal{I}$ of $\mathcal{K}$.
- Deciding whether the degree of membership of an individual a to a concept C with respect to $\mathcal{K}$ is always greater than a given truth degree β, which amounts to deciding whether $C^{\mathcal{I}}(a^{\mathcal{I}}) \geq \beta$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.
- Deciding whether the degree of membership of a pair of individuals (a, b) to a role R with respect to $\mathcal{K}$ is always greater than a given truth degree β, which amounts to deciding whether $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \beta$ holds for all models $\mathcal{I}$ of $\mathcal{K}$.

Other reasoning problems that have been introduced for fuzzy DLs include finding the best entailment degree of an axiom with respect to a knowledge base $\mathcal{K}$, or computing the best satisfiability degree of an axiom with respect to $\mathcal{K}$ [113].

The complexity of deciding these reasoning problems depends not only on the expressivity of a given fuzzy DL (i.e., on which concept constructors are supported), but also on the adopted t-norms, t-conorms, fuzzy negation and fuzzy implication operators. For instance, deciding the consistency of a knowledge base in the fuzzy DL $\mathfrak{N}\mathcal{ALC}$ (where $\mathfrak{N}$ stands for an additional concept constructor called *residual negation*) with the Gödel t-norm $\wedge$ and implication $\Rightarrow_g$ is ExpTime-complete. However, the problem in general becomes undecidable if the product t-norm $\wedge_p$ and implication $\Rightarrow_p$, or the Łukasiewicz t-norm $\wedge_l$ and implication $\Rightarrow_l$, are employed instead [30].

Fuzzy DLs have been implemented in a number of reasoners, such as Fire [104, 110], SoftFacts [114], DeLorean [26], fuzzyDL [28, 29], and LiFR [117]. Several fuzzy extensions of OWL have been proposed (e.g, [50, 111, 112]), along with a method to represent fuzzy DL ontologies using OWL 2 annotation properties [27].

2.2 Similarity-based reasoning in logic programming

Approximate reasoning based on fuzzy relations, similarities in particular, has been researched extensively in fuzzy logic programming. Early work includes Ying's logic for approximate reasoning [118] and the first papers on similarity-based logic programming [17, 48, 51, 101]. One motivation behind the similarity-based approaches is to model a form of reasoning that may be referred to as *reasoning by analogy or similarity* [101]. For instance, this may be achieved by relaxing the *equality* constraint on two functor symbols to a flexible constraint of *similarity*, when unifying two first-order terms (FOTs). For example, if

the functors `thriller` and `horror` are assumed to denote similar concepts, then the term `thriller(X)` can unify with the term `horror("Psycho")` to some extent (degree), leading to approximate (not exact but similar) solutions to a query posed to a knowledge base. This kind of relaxed unification is generally referred to as *weak unification* (e.g., [101]).

This research line has been extended in several ways, including approaches that support multiple similarity relations [45], proximity relations [61, 64, 74], or related operations like matching and anti-unification [71, 72, 73]. Moreover, weak unification has been implemented in fuzzy logic programming systems such as FASILL [60] and Bousi∼Prolog [62, 63], which has been employed in several applications such as text classification [94, 99], linguistic feedback in computer games [97], decision making [32, 33], and knowledge discovery [96]. Aït-Kaci and Pasi [9] have presented a procedure for weak unification that, besides tolerating different (but similar) functor symbols, also allows the unification of FOTs with a different number and possibly a different order of arguments. This work has been generalized to proximity relations [90], and a possible incorporation in Bousi∼Prolog has been proposed [38].

In this section we briefly review weak FOT unification and similarity-based SLD resolution, following [9, 62, 101]. The presentation – which mainly serves as the background and as a means for comparison with Chapter 5, where similarity-based reasoning with OSF logic is introduced – is mostly kept informal and carried out through examples.

For the rest of this section, it is assumed that $\mathcal{V}$ is a countably infinite set of variables denoted X, Y, X_1, and so on. For each $n \geq 0$, Σ_n is a set of n-ary functor symbols (denoted f, g, f_i, and so on), and $\Sigma \stackrel{\text{def}}{=} \bigcup_{n\geq 0} \Sigma_n$. If a functor symbol has arity 0, it is called a constant. For each $n \geq 0$, Π_n is a set of n-ary predicate symbols (denoted p, q, p_i, and so on), and $\Pi \stackrel{\text{def}}{=} \bigcup_{n\geq 0} \Pi_n$. A 0-ary predicate symbol is also called a propositional symbol. The set $\mathcal{T}_{\Sigma,\mathcal{V}}$ of FOTs is defined as usual: variables in $\mathcal{V}$ are terms, constants in Σ_0 are terms, and $f(t_1, \ldots, t_n)$ is a term whenever $f \in \Sigma_n$ and $t_1, \ldots, t_n$ are terms, for each $n \geq 1$. A FOT is denoted by t or t_i for some $i \in \mathbb{N}$. If $p \in \Pi_n$ and $t_1, \ldots, t_n$ are FOTs, then $p(t_1, \ldots, t_n)$ is an atom. The set of variables occurring in a term t is denoted $\mathit{Vars}(t)$. A substitution is a function from $\mathcal{V}$ to $\mathcal{T}_{\Sigma,\mathcal{V}}$ that is the identity except for a finite subset of $\mathcal{V}$, and it can also be represented as a set $\{t_1/X_1, \ldots, t_n/X_n\}$. Substitutions are denoted by θ, θ_1, and so on, and the result of applying a substitution θ to a FOT t is denoted $t\theta$.

2.2.1 Weak unification

Unification is a fundamental operation upon which many methods for automated reasoning are based [20]. Given two FOTs t_1 and t_2, their unification consists in finding the most general substitution θ such that $t_1\theta = t_2\theta$, called their most general unifier (MGU). Several FOT unification algorithms have been proposed in the literature (e.g., see [69] for a survey on the matter). Along the lines of [9], a unification procedure for FOTs consisting of

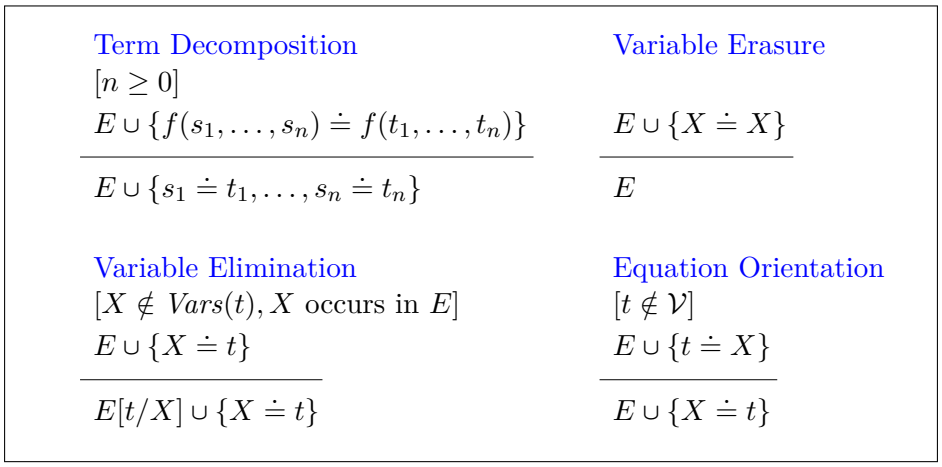


Figure 2.1: Herbrand-Martelli-Montanari FOT unification rules.

transformations of sets of equations is presented in Fig. 2.1. This procedure is dubbed *Herbrand-Martelli-Montanari* unification, as [9] tracks the origin of these rules back to Herbrand's PhD thesis [59] and Martelli and Montanari's work [78]. Each unification rule is of the form

Rule name
[Side condition]
Premise

Conclusion

and it expresses that, whenever the optional side condition is true, the set of equations in the premise can be transformed into the set of equations in the conclusion. A set of equations is denoted by E, and the notation $E[t/X]$ stands for the set of equations resulting after every occurrences of the variable X in E has been replaced with the term t. The unification of two FOTs t_1 and t_2 can be performed by applying the rules of Fig. 2.1 starting from the set $\{t_1 \doteq t_2\}$, until no rule applies, resulting in a set of equations E'. If all equations in E' are of the form $X \doteq t$, with X occurring nowhere else in E', then these equalities provide a most general unifying substitution (modulo variable renaming) $\theta = \{t/X \mid X \doteq t \in E'\}$ such that $t_1\theta = t_2\theta$, otherwise there is no solution. As a simple example, according to the rules of Fig. 2.1, the unification of the FOTs

$$t_1 = movie(hitchcock, psycho) \quad \text{and} \quad t_2 = film(hitchcock, X)$$

immediately fails, as no rule is applicable to the set of equations $\{movie(hitchcock, psycho) \doteq film(hitchcock, X)\}$, since the functors *movie* and *film* are different.

The *weak unification* procedure presented in [101] is a generalization of the standard

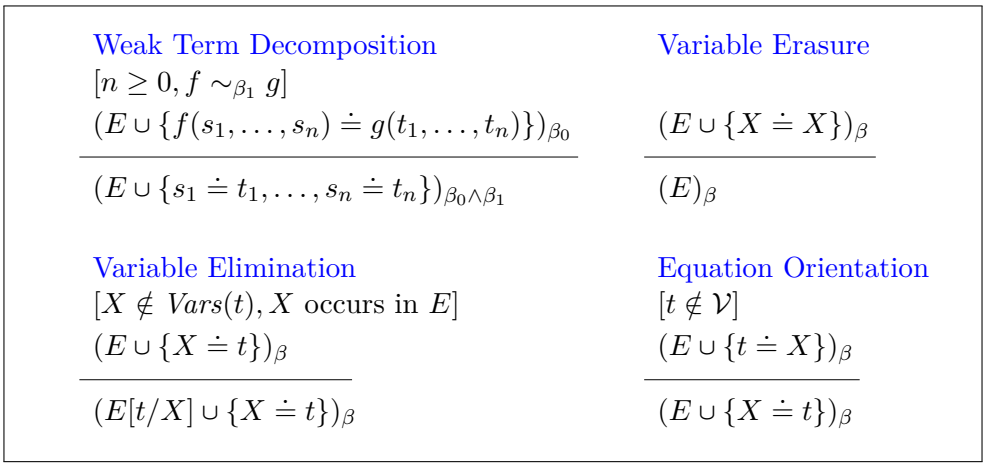


Figure 2.2: Weak FOT unification rules [101].

unification algorithm for FOTs based on a similarity relation[2] $\sim\colon \Sigma \times \Sigma \to [0,1]$ between the functor symbols of a first-order signature. The similarity relation enables to perform approximate reasoning at a syntactic level rather than at a rule level, by allowing to match different but similar functor symbols. In other words, the equality constraint on functor symbols is replaced by a flexible constraint of similarity. The similarity on functor symbols is extended to a similarity relation on FOTs by letting [101]:

- $\sim(X, X) \stackrel{\text{def}}{=} 1$ for all $X \in \mathcal{V}$;
- $\sim(X, t) \stackrel{\text{def}}{=} 0$ and $\sim(t, X) \stackrel{\text{def}}{=} 0$ for all $X \in \mathcal{V}$ and $t \in \mathcal{T}_{\Sigma,\mathcal{V}}$ such that $t \neq X$;
- For all $f, g \in \Sigma_n$ and $s_i, t_i \in \mathcal{T}_{\Sigma,\mathcal{V}}$ such that $f \sim_{\beta} g$ and $s_i \sim_{\beta_i} t_i$ (for $1 \leq i \leq n$):

$$\sim(f(s_1, \ldots, s_n), g(t_1, \ldots, t_n)) \stackrel{\text{def}}{=} \beta \wedge \bigwedge_{i=1}^{n} \beta_i.$$

Fig. 2.2 shows the weak unification rules of Sessa [101]. The main differences with respect to the standard FOT unification rules are that (i) the weak unification rules associate an approximation degree $\beta \in [0,1]$ to each set of equations E, and (ii) the rule Weak Term Decomposition allows the simplification of the equation $f(s_1, \ldots, s_n) \doteq g(t_1, \ldots, t_n)$ even in cases where f and g are different, provided that they are similar to some degree β. The unification process starts from the set of equations $(\{t_1 \doteq t_2\})_1$ with associated approximation degree 1, and it terminates when no more rules are applicable. The approximation degree associated with a set of equations can possibly only decrease before the end of the unification process. If the procedure is successful, the result of applying the

[2]Similarity relations are defined in Section 1.2.3.

rules of Fig. 2.2 to two FOTs t_1 and t_2 is a substitution θ such that $\sim(t_1\theta, t_2\theta) = \beta$ for some $\beta \in [0, 1]$.

Assuming, for instance, that $\sim(\mathit{movie}, \mathit{film}) = 0.8$, then the weak unification procedure of [101] allows to unify the terms $t_1 = \mathit{movie}(\mathit{hitchcock}, \mathit{psycho})$ and $t_2 = \mathit{film}(\mathit{hitchcock}, X)$ with the substitution θ mapping X to *psycho*. The terms $t_1\theta$ and $t_2\theta$ are deemed to be similar with degree 0.8.

As preliminary work towards the definition of similarity-based reasoning with OSF logic, Aït-Kaci and Pasi [9] have proposed a generalization of Sessa's weak unification that allows matching two FOTs even if they have a different arity or if their argument positions are in a different order, besides tolerating different, but similar, functor symbols. Indeed, relaxing these constraints makes the setting more similar to that of OSF logic, where terms do not have a fixed arity or argument order (see Chapter 3). For instance, with the approach of [9] it is possible to weakly unify the FOTs

$$t_3 = \mathit{movie}(\mathit{hitchcock}, \mathit{psycho}, 1960) \quad \text{and} \quad t_4 = \mathit{film}(X, \mathit{hitchcock}).$$

While the unification rules of Fig. 2.1, or the weak unification rules of Fig. 2.2 would clearly fail to unify these two terms, the approach of [9] is to consider, for each $f \in \Sigma_m$ and $g \in \Sigma_n$ with $m \leq n$, an injective function $\mu_{f,g} : \{1, \ldots, m\} \to \{1, \ldots, n\}$ (satisfying a few consistency conditions), called an *argument alignment mapping*. A similarity relation $\approx$ between functor symbols can then be extended to a similarity $\approx$ on $\mathcal{T}_{\Sigma,\mathcal{V}}$. The definition of $\approx$ on variables is the same as Sessa's [101], while the definition on complex terms becomes:

- For all $f \in \Sigma_m$, $g \in \Sigma_n$ and $s_i, t_j \in \mathcal{T}_{\Sigma,\mathcal{V}}$ such that $f \approx_\beta g$ and $s_i \approx_{\beta_i} t_{\mu_{f,g}(i)}$ (with $1 \leq i \leq m$ and $1 \leq j \leq n$):

$$\approx(f(s_1, \ldots, s_m), g(t_1, \ldots, t_n)) \stackrel{\text{def}}{=} \beta \wedge \bigwedge_{i=1}^{m} \beta_i.$$

The weak unification procedure of [9] consists of the rules of Fig. 2.3, together with the rules Variable Erasure, Variable Elimination and Equation Orientation of Fig. 2.2, yielding a unification procedure that tolerates functor arity and argument position mismatches, besides different but similar functor symbols. For instance, consider the terms $t_3 = \mathit{movie}(\mathit{hitchcock}, \mathit{psycho}, 1960)$ and $t_4 = \mathit{film}(X, \mathit{hitchcock})$, and assume that $\mathit{movie} \approx_{0.9} \mathit{film}$ and that the argument alignment mapping $\mu_{\mathit{film},\mathit{movie}}$ is such that $\mu_{\mathit{film},\mathit{movie}}(1) = 2$ and $\mu_{\mathit{film},\mathit{movie}}(2) = 1$. The weak FOT unification rules with non-aligned arguments of [9] would find the substitution θ mapping X to *psycho* as the unifier of the two terms. The terms $t_3\theta$ and $t_4\theta$ are deemed to be similar with degree 0.9.

Fuzzy Non-Aligned-Argument Term Decomposition

$$\frac{[0 \leq m \leq n, f \approx_{\beta_1} g] \quad (E \cup \{f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n)\})_{\beta_0}}{(E \cup \{s_1 \doteq t_{\mu_{f,g}(1)}, \ldots, s_m \doteq t_{\mu_{f,g}(m)}\})_{\beta_0 \wedge \beta_1}}$$

Fuzzy Equation Orientation

$$\frac{[0 \leq m < n] \quad (E \cup \{g(t_1, \ldots, t_n) \doteq f(s_1, \ldots, s_m)\})_{\beta}}{(E \cup \{f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n)\})_{\beta}}$$

Figure 2.3: Weak FOT unification rules with non-aligned arguments [9].

```
horror(X) :- slasher(X).
slasher("Psycho").
thriller("Memento").
```

Figure 2.4: The logic program of Example 2.6.

2.2.2 Similarity-based SLD resolution

The weak unification rules of [101] are instrumental in defining similarity-based SLD resolution, a generalization of the SLD resolution procedure where the classical unification algorithm is replaced by the weak unification procedure of Fig. 2.2. One of the motivations behind the definition of similarity-based SLD resolution is to model a form of reasoning that may be referred to as *reasoning by analogy or similarity* [101]. An example would be inferring "Alinda likes thriller movies" from the premises "Alinda likes horror movies" and "horror movies are similar to thrillers". An advantage of this approach is that it enables *flexible query answering*: besides exact solutions, a query may return similar solutions, both in a semantic sense (for instance, by returning movies of a genre that is similar to the requested one) or syntactically (for example, by tolerating spelling mistakes).

Example 2.6 (Similarity-based SLD resolution). Consider, for example, the Prolog program of Fig. 2.4 and assume that `horror` and `thriller` are similar to degree 0.5. If similarity-based SLD resolution is adopted, the query `?- thriller(X)` will return, besides `X = "Memento"`, also the solution `X = "Psycho"` with approximation degree 0.5. This is due to the fact that, thanks to the similarity relation, the query will resolve with the first clause of the program, leading to an approximate solution. Intuitively, since `"Psycho"` is a `horror` movie as a consequence of the first rule, and `horror` movies are similar to `thrillers`, then, to some degree, `"Psycho"` is also a `thriller`. ▷

```
% Subsumption rules
person(X) :- director(X).
horror(X) :- slasher(X).
movie(X) :- horror(X).
movie(X) :- thriller(X).
% Similarity relation
thriller ~ horror = 0.5.
% Instances
person(alinda).
director(hitchcock).
director(carpenter).
director(nolan).
horror(psycho).
slasher(halloween).
thriller(memento).
% Facts
director_of(hitchcock, psycho).
director_of(carpenter, halloween).
director_of(nolan, memento).
% Rules
likes(alinda, Y) :- thriller(Y), director_of(X, Y).
```

Figure 2.5: The Bousi~Prolog program of Example 2.7.

Weak unification and similarity-based SLD resolution have been implemented, for instance, in the fuzzy logic programming language Bousi~Prolog. Bousi~Prolog extends the syntax of Prolog by allowing to specify similarity declarations of shape $a \sim b = \beta$, where a and b are functor (or predicate) symbols, and $\beta \in [0, 1]$. Bousi~Prolog also generalizes Sessa's approach by supporting proximity relations [62, 63] (i.e., reflexive and symmetric fuzzy binary relations, which can be more appropriate in some modeling contexts [61, 67, 103]), and linguistic variables [95]. The following example shows a Bousi~Prolog program and the computation of approximate solutions to a query through similarity-based SLD resolution.

Example 2.7 (Bousi~Prolog program). Consider the Bousi~Prolog program of Fig. 2.5. The syntax is essentially the same as that of Prolog, except for a similarity declaration stating that the predicate symbols `horror` and `thriller` are similar to degree 0.5. The query `?- likes(alinda, Y)` will return, for instance, a solution mapping `Y` to `halloween` with approximation degree 0.5, through the following steps:

- By resolving against the last rule, the query `?- likes(alinda, Y)` simplifies into `?- thriller(Y), director_of(X, Y)`.
- Since `horror` and `thriller` are similar, by similarity-based SLD resolution the first

goal `thriller(Y)` resolves against the rule `horror(X) :- slasher(X)`, so that the query becomes `?- slasher(Y), director_of(X, Y)`. The approximation degree associated with the computation becomes 0.5.

- The first goal `slasher(Y)` now resolves with the fact `slasher(halloween)`, so that `Y` is mapped to `halloween`, and the query becomes `?- director_of(X, halloween)`.
- The remaining goal resolves with the fact `director_of(carpenter, halloween)`, and the computation terminates successfully.

Through similar steps, the program will also return a solution mapping `Y` to `psycho` with approximation degree 0.5, and a solution mapping `Y` to `memento`. ▷

Chapter 5 will deal with the development of similarity-based reasoning with OSF logic, by extending its language with a similarity relation on sort symbols. One challenge that arises in this context is how to reconcile the similarity relation with the sort subsumption ordering on which OSF logic is grounded. Our solution will be to combine these two relations into a single *fuzzy subsumption relation*, shifting the setting to that of *fuzzy OSF logic*, which is developed in Chapter 4. This strategy allows to seamlessly integrate the sort similarity relation into the unification algorithm of fuzzy OSF logic. As will be discussed in Chapter 5, an advantage of this approach is that a single fuzzy OSF term unification can replace several similarity-based SLD resolution steps (Example 5.3).

Chapter 3

Order-Sorted Feature Logic

Order-Sorted Feature (OSF) logic is a Knowledge Representation and Reasoning (KRR) language that originates in Hassan Aït-Kaci's work [2]. At the core of OSF logic are two kinds of symbols that are employed to represent concepts and properties: *sort symbols* (or *sorts*) are used to denote conceptual classes such as $person$ or $director$, and *feature symbols* (or *features*) are used to describe functional attributes of objects, like $directed\_by$, $written\_by$ or $name$. Sorts are ordered in a *subsumption relation* $\preceq$ that denotes inclusion between classes. For example, the subsumption $director \preceq person$ means that every director is a person.

Together with *variables*, also named *coreference tags*, sorts and features can be used to construct record-like structures called *OSF terms* that can represent complex concepts, similarly to the defined concepts of Description Logic (DLs). The following OSF term, for example, denotes the class of movies that are written and directed by the same person:

$$t_1 = X_1 : movie \left(\begin{array}{lll} directed\_by & \rightarrow & Y_1 : person, \\ written\_by & \rightarrow & Y_1 \end{array} \right).$$

Reasoning in OSF logic is based on the unification of OSF terms, a procedure that aims to combine in a single term the constraints expressed by two terms. For instance, the unification of the term t_1 with the term

$$t_2 = X_2 : movie \left(\begin{array}{lll} directed\_by & \rightarrow & Y_2 : director, \\ genre & \rightarrow & Z_2 : thriller \end{array} \right)$$

results in the term

$$t_3 = X_3 : movie \left(\begin{array}{lll} directed\_by & \rightarrow & Y_3 : director, \\ written\_by & \rightarrow & Y_3, \\ genre & \rightarrow & Z_3 : thriller \end{array} \right).$$

The unification procedure takes the sort subsumption relation into account: for example, assuming that *director* $\preceq$ *person*, since the feature *directed_by* in t_1 points to the sort *person*, and the same feature in t_2 points to the sort *director*, then the value of this feature in the unifier t_3 must be of sort *director*, i.e., the most specific of the two sorts. Moreover, the sort subsumption ordering $\preceq$ can be extended to an ordering between OSF terms, and the unification algorithm for such structures provides an efficient way to decide whether two OSF terms are subsumed by each other (for instance, t_3 is subsumed by t_1 and t_2), and, in general, for finding the most general OSF term that is subsumed by both terms, thus offering an efficient calculus of partially ordered types [8].

Similarly to DLs, OSF logic was initially meant as a formalization of Ron Brachman's structured inheritance networks [4, 18]. Indeed, due to their common origin, the two languages share a few similarities, as well as several distinguishing aspects. For instance, both formalisms are subsets of first-order logic designed to simplify its language in order to achieve computational tractability, while still providing enough expressive power for effective knowledge representation and reasoning. DLs and OSF logic are both based on set-denoting symbols (concepts and sorts, respectively), and on symbols for expressing attributes: *relational roles* for DL, and *functional features* for OSF Logic. While the feature symbols of OSF logic denote *total functions*, which may appear less versatile than the relational roles of DLs, versions of this language that support partial functions or relations have also been defined [4, 34, 106]. The OSF sort subsumption relation is also comparable to the inclusion axioms of a DL T-Box. One of the most significant differences with respect to DLs is that the semantics of OSF logic is based on the *closed world assumption*: for instance, if two sorts do not share a common subsort, then they are assumed to be disjoint. The relationship between OSF logic and DLs has been explored thoroughly in works such as [4, 70, 87, 88].

OSF logic and related formalisms, like feature logic [106] or the logic of typed feature structures [34], have been applied in computational linguistics [34] and implemented in constraint logic programming languages such as LOGIN [8], LIFE [10] and CIL [86]. On the other hand, DLs have found one of their primary applications in the Semantic Web, as they provide, by design, the formal semantics for the Web Ontology Language (OWL), the ontology language developed by the World Wide Web Consortium [18]. More recently, OSF logic was proposed by Aït-Kaci as an alternative formalism for the Semantic Web [4], and the language has been implemented in the CEDAR Semantic Web reasoner [6, 15].

The goal of this chapter is to provide a comprehensive overview of OSF logic, setting the foundations for our development of fuzzy OSF logic in Chapter 4 and of similarity-based OSF logic in Chapter 5. We start by presenting the main syntactic objects of this language in Section 3.1, namely sorts, features, OSF terms and OSF clauses. These objects are then given a meaning in structures called OSF algebras, which are discussed in Section 3.2.

Besides terms and clauses, OSF logic also enjoys a third syntactic representation, namely

OSF graphs. These are rooted directed graphs whose nodes are labeled by sorts and whose edges are labeled by features. OSF graphs are the elements of the domain of an OSF algebra that is of core importance for the development of OSF logic, the OSF graph algebra, which is presented in Section 3.3. In Section 3.4 we explore OSF algebra homomorphisms, structure-preserving mappings between OSF algebras that are essential for proving several results regarding the satisfiability of OSF clauses. OSF algebra homomorphisms are also central in Section 3.5, which is devoted to proving that the subsumption ordering between sort symbols can be extended to the other syntactic objects of OSF logic and, in particular, we show that OSF terms form a lattice. The greatest lower bound of two OSF terms can be computed through a unification procedure which is presented in Section 3.6.

The integration of OSF term unification in the logic programming language LOGIN is then outlined in Section 3.7.1, followed by an overview of the Semantic Web reasoner CEDAR based on OSF logic in Section 3.7.2. The implementation of OSF logic in LOGIN and CEDAR is based on techniques that exploit the specificity of concept taxonomies [15], and in particular on *graph encoding techniques* that allow to efficiently perform lattice operations. These techniques are discussed in Section 3.8.

While our presentation of OSF logic closely follows Aït-Kaci's development of this language [10], we have taken the opportunity to make a few adjustments. For example, Definition 3.12 redefines the procedure to translate OSF constraints into OSF terms, as the original one can lead to an infinite loop. We also refine a few theorems and definitions, such as Theorem 3.49 and Definition 3.50, since the original ones are susceptible to counterexamples, as discussed in Remark 4. Moreover, we define a syntactic notion of OSF term subsumption and prove that it is equivalent to the semantic one of [10].

3.1 Syntax

In this section we provide the definitions of OSF signature and of two formal languages that are used to represent knowledge with OSF logic: *OSF terms* and *OSF clauses*. As discussed in the introduction, OSF terms are comparable to the defined concepts of DLs, and they are interpreted as subsets of the domain of an interpretation. An OSF clause is an equivalent representation that can be seen as a logical reading of an OSF term, and for which a notion of satisfiability is defined. In OSF logic, both syntactic representations are important from an implementation perspective, as OSF terms are the abstract syntax employed by an user, while OSF clauses are used in the constraint normalization rules needed for OSF term unification [10].

We begin our overview of the syntax of OSF logic by introducing the concept of an *OSF signature*, a tuple that specifies the sets of sort and feature symbols that may be used to construct OSF terms. Additionally, an OSF signature also prescribes a lattice ordering for

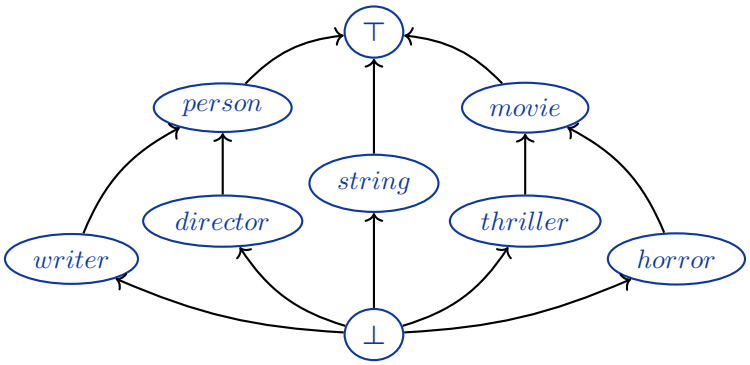


Figure 3.1: The OSF signature of Example 3.2.

the sort symbols.

Definition 3.1 (OSF signature [10]). An *OSF signature* is a tuple $(\mathcal{S}, \mathcal{F}, \preceq)$ such that

1. $\mathcal{S}$ is a finite set of *sort symbols*,
2. $\mathcal{F}$ is a finite set of *feature symbols*,
3. $(\mathcal{S}, \preceq)$ is a bounded lattice with least element $\bot$ and greatest element $\top$.

Elements of $\mathcal{S}$ and $\mathcal{F}$ are also simply called *sorts* and *features*, respectively. For a sort $s \in \mathcal{S}$, the set $s{\uparrow} \stackrel{\text{def}}{=} \{s' \in \mathcal{S} \mid s \preceq s'\}$ is the set of *supersorts of* s, while the set $s{\downarrow} \stackrel{\text{def}}{=} \{s' \in \mathcal{S} \mid s' \preceq s\}$ is the set of *subsorts of* s. The greatest lower bound (GLB) $s \wedge s'$ of two sorts s and s' is also called their *greatest common subsort*. The least upper bound (LUB) $s \vee s'$ of two sorts s and s' is also called their *least common supersort*.

Example 3.2 (OSF signature). An example of an OSF signature in the domain of movies is the tuple $(\mathcal{S}, \mathcal{F}, \preceq)$ such that $(\mathcal{S}, \preceq)$ is the lattice represented in Fig. 3.1, and $\mathcal{F} \stackrel{\text{def}}{=} \{\mathit{written\_by}, \mathit{directed\_by}, \mathit{title}\}$ · ▷

Starting from sorts and features it is possible to construct record-like syntactic structures called *OSF terms*, which serve as representations for complex concepts.

Definition 3.3 (OSF term [10]). Let $\mathcal{V}$ be a countably infinite set of variables (or *coreference tags*, or simply *tags*), and $(\mathcal{S}, \mathcal{F}, \preceq)$ be an OSF signature. Let $X \in \mathcal{V}$, $s \in \mathcal{S}$ and $f_1, \ldots, f_n \in \mathcal{F}$. An *OSF term* is defined recursively as follows.

- A sorted variable $X : s$ is an OSF term.
- An attributed sorted variable $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$ is an OSF term whenever $t_1, \ldots, t_n$ are OSF terms.

We let $\mathit{Tags}(t) \stackrel{\text{def}}{=} \{X\} \cup \bigcup_{1 \leq i \leq n} \mathit{Tags}(t_i)$. The variable X is called the *root tag* of t and is denoted $\mathit{RootTag}(t)$, and the sort s is called the *root sort* of t.

Example 3.4 (OSF term). The following OSF term denotes the class of movies that are written and directed by the same person:

$$X : \mathit{movie} \left(\begin{array}{ll} \mathit{directed\_by} & \to Y : \mathit{person}, \\ \mathit{written\_by} & \to Y \end{array} \right).$$

The variable Y is used as a coreference tag, i.e., it specifies that the values of the features *directed_by* and *written_by* should be the same. The variables in an OSF term are often left implicit unless they are necessary to express this property, as in the following OSF term:

$$t = \mathit{movie} \left(\begin{array}{lll} \mathit{title} & \to & \mathit{string}, \\ \mathit{directed\_by} & \to & X : \mathit{director} \left(\begin{array}{ll} \mathit{name} & \to \mathit{string}, \\ \mathit{spouse} & \to Y \end{array} \right), \\ \mathit{written\_by} & \to & Y : \mathit{writer} \left(\mathit{spouse} \to X \right) \end{array} \right). \qquad \triangleright$$

OSF terms generalize the terms of first-order logic in several ways. First of all, they lack a fixed number of arguments, so that they are convenient to represent partial information. For example, an OSF logic knowledge base could include terms of shape $\mathit{movie}(\mathit{title} \to$ "*Psycho*", $\mathit{year} \to 1960)$ and also of shape $\mathit{movie}(\mathit{title} \to$ "*Halloween*"$)$.[1] Moreover, the arguments of an OSF terms are identified by features rather than positions, which helps the interpretability of a term. For instance, consider the OSF term $\mathit{movie}(\mathit{title} \to$ "*Adaptation*", $\mathit{directed\_by} \to$ *"Spike Jonze"*, $\mathit{written\_by} \to$ *"Charlie Kaufman"*$)$ and the first-order term *movie*(*"Adaptation"*, *"Spike Jonze"*, *"Charlie Kaufman"*). Features can also be left implicit in an OSF term by considering $s(X, Y, s')$ as an abbreviation for $s(1 \to X, 2 \to Y, 3 \to s')$, where the features 1, 2 and 3 take on a positional meaning. An additional feature of OSF terms is their capability to express cycles, as exemplified by the term $X : \mathit{person}\,(\mathit{name} \to$ *"Celeste"*, $\mathit{spouse} \to Y : \mathit{person}\,(\mathit{spouse} \to X))$, thus enhancing the flexibility of this language.

The definition of OSF terms given above does not rule out the presence of redundant or even contradictory information (e.g., consider the OSF term $s(f \to s_0, f \to s_1)$, which is contradictory if $s_0 \wedge s_1 = \bot$). OSF terms that are well-behaved to this regard are called *normal OSF terms* and are defined as follows [10].

Definition 3.5 (Normal OSF term, or ψ-term [10]). An OSF term $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$ is in *normal form* (or *normal*) if the following conditions are satisfied.

[1] Strings such as "*Psycho*" and "*Halloween*", and integers such as 1960, are treated as *singleton sorts*, i.e., sorts that denote a single element.

1. The root sort s is different from $\bot$.
2. The features $f_1, \ldots, f_n \in \mathcal{F}$ are pairwise distinct.
3. Each t_i is in normal form.
4. For all $Y \in \mathit{Tags}(t)$, there is at most one occurrence of Y in t such that Y is the root variable of an OSF term different from $Y : \top$.

OSF terms in normal form are also called *ψ-terms* and denoted ψ, ψ_i, and so on. For an OSF term ψ in normal form and $X \in \mathit{Tags}(\psi)$, we let $\mathit{Sort}_\psi(X)$ be the most specific sort s such that $X : s$ appears in ψ. The notation $X \xrightarrow{f}_\psi Y$ indicates that there is a feature f pointing from a subterm of ψ with root tag X to a subterm of ψ with root tag Y. We let Ψ denote the set of all normal OSF terms.

A method for transforming an OSF term into a normal one is more easily presented as a constraint normalization procedure for *OSF constraints*, which are defined next.

Definition 3.6 (OSF constraints and clauses [10]). Let $\mathcal{V}$ be a countably infinite set of variables, and $(\mathcal{S}, \mathcal{F}, \preceq)$ be an OSF signature. An *OSF constraint* is an expression of the form $X : s$, $X \doteq X'$, or $X.f \doteq X'$, where $X, X' \in \mathcal{V}$, $s \in \mathcal{S}$ and $f \in \mathcal{F}$. If $\phi_1, \phi_2, \ldots, \phi_n$ are OSF constraints, then their conjunction $\phi = \phi_1 \,\&\, \phi_2 \,\&\, \ldots \,\&\, \phi_n$ is an *OSF clause*. The set of variables occurring in ϕ is denoted $\mathit{Tags}(\phi)$, while $\phi[X/Y]$ is the OSF clause obtained by replacing all occurrences of the variable Y with X.

Informally, the constraint $X : s$ means that the value assigned to X is of sort s, $X \doteq X'$ means that the same value is assigned to the variables X and X', while $X.f \doteq X'$ means that applying the feature f to the value assigned to X returns the value assigned to X'.

Example 3.7 (OSF clause). Let ϕ be the following OSF clause.

$$\begin{array}{llllll} X_0 : \mathit{movie} & \& & X_0.\mathit{title} \doteq X_1 & \& & X_1 : \mathit{string} & \& \\ X_0.\mathit{directed\_by} \doteq X & \& & X : \mathit{director} & \& & X.\mathit{name} \doteq X_2 & \& \\ X_2 : \mathit{string} & \& & X.\mathit{spouse} \doteq Y & \& & X_0.\mathit{written\_by} \doteq Y & \& \\ Y : \mathit{writer} & \& & Y.\mathit{spouse} \doteq X. & & & \end{array}$$

Note that ϕ is simply a translation of the term t from Example 3.4 into an OSF clause. The variables that were left implicit in Example 3.4 must be written explicitly in the OSF clause. ▷

We thus specify formally a way to translate an OSF term into an OSF clause. An analogous mapping will be provided for the opposite direction, and, in Section 3.3, procedures will be defined to translate OSF terms and OSF clauses into OSF graphs, and back. The definition of these mapping is necessary, as they allow us to move effortlessly between the three representations, depending on which one is more convenient at a given moment.

Definition 3.8 (Mapping from normal OSF terms to OSF clauses [10]). The mapping ϕ from normal OSF terms to OSF clauses is defined as follows [10]: if $\psi = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$, then $\phi(\psi) \stackrel{\text{def}}{=} X : s \;\&\; \mathop{\&}_{1 \le i \le n} (X.f_i \doteq \mathit{RootTag}(t_i) \;\&\; \phi(t_i))$.

Note that the same symbol ϕ is used to denote an OSF clause and the function mapping an OSF term to a clause, as its meaning is always clear from context.

While every OSF term can be rewritten as an OSF clause, the converse is in general only true for the class of *rooted solved OSF clauses* [10], which are defined next.

Definition 3.9 (Rooted OSF clause and maximal subclause rooted in X[10]). Given an OSF clause ϕ, the binary reachability relation $\overset{\phi}{\rightsquigarrow} \subseteq \mathit{Tags}(\phi)^2$ is defined as follows, for all $X, Y \in \mathit{Tags}(\phi)$: (i) $X \overset{\phi}{\rightsquigarrow} X$, and (ii) $X \overset{\phi}{\rightsquigarrow} Y$ if there is a constraint $X.f \doteq Z$ in ϕ and $Z \overset{\phi}{\rightsquigarrow} Y$. A *rooted OSF clause* ϕ_X is an OSF clause ϕ together with a distinguished variable X (its root) such that every variable Y occurring in ϕ is explicitly sorted (possibly as $Y : \top$) and reachable from X (i.e., $X \overset{\phi}{\rightsquigarrow} Y$). Given an OSF clause ϕ and a variable $X \in \mathit{Tags}(\phi)$, the *maximal subclause of ϕ rooted in X* is denoted $\phi(X)$.

Definition 3.10 (Solved OSF clause [10]). An OSF clause ϕ is called *solved* if, for each variable X, ϕ contains (i) at most one sort constraint of the form $X : s$ (with $s \neq \bot$), (ii) at most one feature constraint of the form $X.f \doteq Y$ for each f, and (iii) no equality constraint of the form $X \doteq Y$. The set of all OSF clauses in solved form is denoted Φ, and the subset of rooted solved OSF clauses is denoted Φ_R.

Example 3.11 (Rooted and solved OSF clauses). The OSF clause ϕ of Example 3.7 is a solved clause rooted in the variable X. On the other hand, the clause

$$\begin{array}{rlllllll} \phi' = & X_0 : \mathit{movie} & \& & X_0.\mathit{title} \doteq X_1 & & \& & X_1 : \mathit{string} & \& \\ & Y_0 : \mathit{movie} & \& & Y_0.\mathit{directed\_by} \doteq Y_1 & & \& & Y_1 : \mathit{person} & \end{array}$$

is solved, but not rooted. The maximal subclause of ϕ' rooted in X_0 is $\phi'(X_0) = X_0 : \mathit{movie} \;\&\; X_0.\mathit{title} \doteq X_1 \;\&\; X_1 : \mathit{string}$. The clause

$$\begin{array}{rlllll} \phi'' = & X : \mathit{movie} & \& & X.\mathit{directed\_by} \doteq X_1 & \& & X_1 : \mathit{person} \\ & & \& & X.\mathit{directed\_by} \doteq Y_1 & \& & Y_1 : \mathit{director} \end{array}$$

is rooted, but not solved. ▷

We are thus ready to define the syntactic mapping that translates rooted solved OSF clauses into normal OSF terms.

Definition 3.12 (Mapping from rooted solved OSF clauses to normal OSF terms). Assume without loss of generality a total ordering on $\mathcal{F}$, which induces a lexicographic

ordering on $\mathcal{F}^*$ (the set of all sequences of elements of $\mathcal{F}$). The mapping $\psi_\phi : \Phi_R \to \Psi$ from rooted solved OSF clauses to normal OSF terms is defined as follows. Suppose that ϕ_X contains the constraint $X : s$[2], and that $X.f_1 \doteq Y_1, \ldots, X.f_n \doteq Y_n$ are all the other constraints in ϕ with an occurrence of the variable X on the left-hand side. The OSF term $\psi = \psi_\phi(\phi_X)$ is constructed as follows:

1. X and s are the root tag and the root sort of ψ, respectively;
2. for each $1 \leq i \leq n$, if the variable Y_i in the constraint $X.f_i \doteq Y_i$ has not yet occurred in the construction of ψ (in the predetermined ordering of $\mathcal{F}^*$), then the subterm $f_i \to \psi_\phi(\phi(Y_i))$ is added to ψ, and the construction continues recursively from $\psi_\phi(\phi(Y_i))$. Otherwise, the subterm $f_i \to Y_i : \top$ is added to ψ instead.

The lexicographic ordering on $\mathcal{F}$ is needed to ensure that $\psi_\phi(\phi_X)$ is unique.

Example 3.13 (Mapping OSF terms to and from OSF clauses). Let $\phi = X : s \,\&\, X.f \doteq Y \,\&\, Y : \top \,\&\, Y.f \doteq X$ and note that $\phi(X) = \phi(Y) = \phi$.

Following Definition 3.12 we construct $\psi = \psi_\phi(\phi_X)$ as follows.

- The root tag and the root sort of ψ are X and s, respectively.
- Because ϕ_X contains the constraint $X.f \doteq Y$ and the variable Y has not appeared in the construction yet, then we add the subterm $f \to \psi_\phi(\phi(Y))$ to ψ. So far, the construction has yielded the term $X : s(f \to \psi_\phi(\phi(Y)))$.
- The construction continues recursively from $\psi' = \psi_\phi(\phi(Y))$. The root tag and root sort of ψ' must be Y and $\top$, respectively. Next, because we have the constraint $Y.f \doteq X$, but X has already appeared in the construction of ψ, then ψ' only contains the subterm $f \to X : \top$. Thus $\psi' = Y : \top(f \to X : \top)$.

This concludes the construction, yielding the term

$$\psi_\phi(\phi(X)) = X : s(f \to \psi_\phi(\phi(Y))) = X : s(f \to Y : \top(f \to X : \top)). \qquad \triangleright$$

Remark 3. Our definition of the mapping ψ_ϕ differs from the original one [10], where $\psi_\phi(\phi_X)$ is instead given as $\psi_\phi(\phi_X) \stackrel{\text{def}}{=} X : s(f_1 \to \psi_\phi(\phi(Y_1)), \ldots, f_n \to \psi_\phi(\phi(Y_n)))$. The issue with this definition is that it can lead to infinite loops. For example, applying this mapping to the clause ϕ_X of Example 3.13 results in

$$\begin{aligned}
\psi_\phi(\phi_X) &= X : s(f \to \psi_\phi(\phi(Y))) \\
&= X : s(f \to Y : \top(f \to \psi_\phi(\phi(X)))) \\
&= X : s(f \to Y : \top(f \to X : s(f \to \psi_\phi(\phi(Y))))) \\
&= \ldots
\end{aligned}$$

[2]Otherwise, we can assume the implicit existence of $X : \top$.

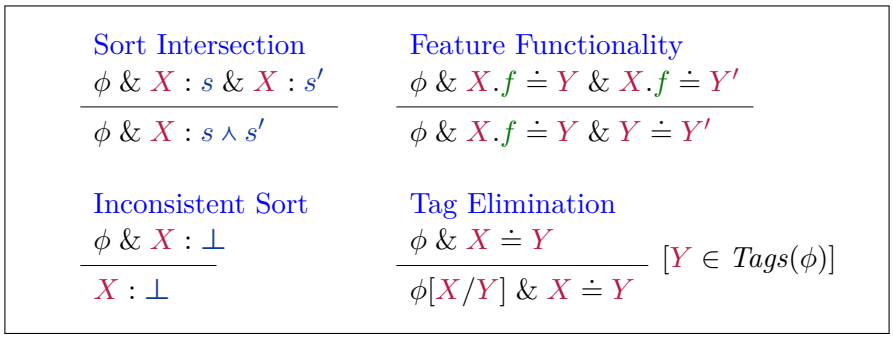


Figure 3.2: OSF constraint normalization rules.

Definition 3.12 is inspired by the mapping that translates OSF graphs to OSF terms from [10] (also see Definition 3.33).

Proposition 3.14 (Bijections between normal OSF terms and rooted solved OSF clauses [10]). *The mappings* $\phi : \Psi \to \Phi_R$ *and* $\psi_\phi : \Phi_R \to \Psi$ *are bijections, i.e.:* $id_{\Phi_R} = \phi \circ \psi_\phi$ *and* $id_\Psi = \psi_\phi \circ \phi$.

Finally, we report the constraint normalization rules that are needed to transform an OSF clause into a solved form [10]. Each rule is of the form

$$\frac{\begin{array}{l}\text{Rule name}\\ \text{Premise } \phi\end{array}}{\text{Conclusion } \phi'} \;[\text{Side condition}]$$

and it expresses that, whenever the (optional) side condition holds, the premise ϕ can be simplified into the conclusion ϕ'.

Proposition 3.15 (OSF clause normalization [10]). *The rules of Fig. 3.2 are finite terminating and confluent (modulo variable renaming). Furthermore, they always result in a normal form that is either the inconsistent clause or an OSF clause in solved form together with a conjunction of equality constraints.*

The constraint normalization rules also provide a procedure for transforming an OSF term t into an equivalent normal form, by applying the constraint normalization rules to $\phi(t)$ and translating the result back into an OSF term.

Example 3.16 (OSF clause normalization). Consider the clause ϕ'' from Example 3.11. An application of the rule Feature Functionality to this clause leads to the clause

$$\begin{array}{lllll} X : \mathit{movie} & \& & X.\mathit{directed\_by} \doteq X_1 & \& & X_1 : \mathit{person} \\ & \& & X_1 \doteq Y_1 & \& & Y_1 : \mathit{director} \end{array}$$

Applying the rule Tag Elimination then yields the clause

$$X : \mathit{movie} \;\&\; X.\mathit{directed\_by} \doteq X_1 \;\&\; X_1 : \mathit{person} \;\&\; X_1 \doteq Y_1 \;\&\; X_1 : \mathit{director}$$

Finally, an application of the rule Sort Intersection results in the OSF clause in solved form $X : \mathit{movie} \;\&\; X.\mathit{directed\_by} \doteq X_1 \;\&\; X_1 : \mathit{director}$ together with the equality constraint $X_1 \doteq Y_1$. ▷

3.2 Semantics

The meaning of the symbols of OSF logic is given by an OSF algebra, a structure that specifies the denotation of sort symbols as sets and of feature symbols as functions. Moreover, a sort subsumption lattice is interpreted in an OSF algebra as a set inclusion relation that must ensure that the denotation of the GLB of two sorts corresponds to the intersection of their respective sets.

Definition 3.17 (OSF algebra [10]). An *OSF algebra (or interpretation)* for a signature $(\mathcal{S}, \mathcal{F}, \preceq)$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that

1. $\Delta^{\mathcal{I}}$ is a non-empty set, called the *domain* or *universe* of the algebra,
2. for each $s \in \mathcal{S}$, $s^{\mathcal{I}}$ is a subset of $\Delta^{\mathcal{I}}$,
3. for each $s_0, s_1 \in \mathcal{S}$: if $s_0 \preceq s_1$, then $s_0^{\mathcal{I}} \subseteq s_1^{\mathcal{I}}$,
4. for each $s_0, s_1 \in \mathcal{S}$: $(s_0 \wedge s_1)^{\mathcal{I}} = s_0^{\mathcal{I}} \cap s_1^{\mathcal{I}}$, and
5. for each $f \in \mathcal{F}$: $f^{\mathcal{I}}$ is a function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}$.

Going forward, we refrain from explicitly specifying the OSF signature and instead assume that the syntactic objects of OSF logic are interpreted in OSF algebras for the relevant signature.

Example 3.18 (OSF algebra). Consider the signature of Example 3.2. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be the OSF algebra for this signature defined as follows.

- The domain is

$$\Delta^{\mathcal{I}} = \{\mathit{psycho}, \mathit{vertigo}, \mathit{hitchcock}, \mathit{coppel}, \mathit{stefano}, \text{“}\mathit{Psycho}\text{”}, \text{“}\mathit{Vertigo}\text{”}, \mathit{null}\}.$$

- The interpretation of the sort symbols is defined by letting
  - $\mathit{string}^{\mathcal{I}} = \{\text{“}\mathit{Psycho}\text{”}, \text{“}\mathit{Vertigo}\text{”}\}$, $\bot^{\mathcal{I}} = \emptyset$, $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$;

  - $\mathit{director}^{\mathcal{I}} = \{\mathit{hitchcock}\}$, $\mathit{writer}^{\mathcal{I}} = \{\mathit{coppel}, \mathit{stefano}\}$, $\mathit{person}^{\mathcal{I}} = \mathit{director}^{\mathcal{I}} \cup \mathit{writer}^{\mathcal{I}}$; and
  - $\mathit{thriller}^{\mathcal{I}} = \{\mathit{vertigo}\}$, $\mathit{horror}^{\mathcal{I}} = \{\mathit{psycho}\}$, $\mathit{movie}^{\mathcal{I}} = \mathit{thriller}^{\mathcal{I}} \cup \mathit{horror}^{\mathcal{I}}$.

- The interpretation of the feature symbols is defined by letting
  - $\mathit{directed\_by}^{\mathcal{I}}(\mathit{psycho}) = \mathit{directed\_by}^{\mathcal{I}}(\mathit{vertigo}) = \mathit{hitchcock}$;
  - $\mathit{written\_by}^{\mathcal{I}}(\mathit{psycho}) = \mathit{stefano}$, $\mathit{written\_by}^{\mathcal{I}}(\mathit{vertigo}) = \mathit{coppel}$;
  - $\mathit{title}^{\mathcal{I}}(\mathit{psycho}) =$ "*Psycho*" and $\mathit{title}^{\mathcal{I}}(\mathit{vertigo}) =$ "*Vertigo*"; and
  - all the remaining feature applications are equal to *null*. ▷

Since features are interpreted as *total* functions, in the last example the feature *title* had to be defined also for elements of sort *person* such as *hitchcock*. While we can circumvent this issue by assigning $\mathit{title}^{\mathcal{I}}(\mathit{hitchcock}) = \mathit{null}$, there are versions of OSF logic that interpret features as *partial* functions instead [4, 34, 106].

In Section 3.1 we presented OSF terms and OSF clauses as two alternative, but syntactically equivalent, data structures for representing knowledge with OSF logic. We thus define the *denotation* of an OSF term as a subset of the domain and the *satisfaction* of an OSF clause, and show that these two notions are also *semantically* equivalent.

The denotation of an OSF term in an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is derived from the interpretation of the sorts, features and variables it contains. The meaning of variables is given by a variable assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$, and the set of all variable assignments is referred to as $\mathit{Val}(\mathcal{I})$.

Definition 3.19 (Denotation of an OSF term [10]). Let $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$ be an OSF term and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an OSF algebra. Let $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ be a variable assignment. The *denotation of t in the algebra $\mathcal{I}$ under the assignment α* is defined as

$$[\![t]\!]^{\mathcal{I},\alpha} \stackrel{\text{def}}{=} \{\alpha(X)\} \cap s^{\mathcal{I}} \cap \bigcap_{1 \le i \le n} (f_i^{\mathcal{I}})^{-1}([\![t_i]\!]^{\mathcal{I},\alpha})$$

where, for a subset $D \subseteq \Delta^{\mathcal{I}}$, $(f_i^{\mathcal{I}})^{-1}(D) \stackrel{\text{def}}{=} \{d \in \Delta^{\mathcal{I}} \mid f_i^{\mathcal{I}}(d) \in D\}$. The *denotation of t in the algebra $\mathcal{I}$* is defined as

$$[\![t]\!]^{\mathcal{I}} \stackrel{\text{def}}{=} \bigcup_{\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}} [\![t]\!]^{\mathcal{I},\alpha}.$$

Note that, for any OSF algebra $\mathcal{I}$, assignment α and OSF term t, the set $[\![t]\!]^{\mathcal{I},\alpha}$ is always a singleton or the empty set.

Example 3.20 (Denotation of an OSF term). Continuing from Example 3.18, consider the term $t = X : \mathit{movie}\,(\mathit{directed\_by} \to X_0 : \mathit{director})$ and the assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ such

that $\alpha(X) = \mathit{psycho}$ and $\alpha(X_0) = \mathit{hitchcock}$. The denotation of the term t in $\mathcal{I}$ according to α is

$$\begin{aligned}
[\![t]\!]^{\mathcal{I},\alpha} &= \{\alpha(X)\} \cap \mathit{movie}^{\mathcal{I}} \cap (\mathit{directed_by}^{\mathcal{I}})^{-1}[\![X_0 : \mathit{director}]\!]^{\mathcal{I},\alpha} \\
&= \{\mathit{psycho}\} \cap \{\mathit{psycho}, \mathit{vertigo}\} \cap (\mathit{directed_by}^{\mathcal{I}})^{-1}(\{\alpha(X_0)\} \cap \mathit{director}^{\mathcal{I}}) \\
&= \{\mathit{psycho}\} \cap (\mathit{directed_by}^{\mathcal{I}})^{-1}(\{\mathit{hitchcock}\}) \\
&= \{\mathit{psycho}\} \cap \{\mathit{psycho}, \mathit{vertigo}\} = \{\mathit{psycho}\}.
\end{aligned}$$

It is also easy to see that $[\![t]\!]^{\mathcal{I}} = \{\mathit{psycho}, \mathit{vertigo}\}$.

As another example, let $\mathcal{I}$ be an arbitrary OSF algebra, let $t_1 = X : s(f_0 \to Y : \top, f_1 \to Y : \top)$, and let $t_2 = X : s(f \to X)$. Then $[\![t_1]\!]^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid d \in s^{\mathcal{I}} \text{ and } f_0^{\mathcal{I}}(d) = f_1^{\mathcal{I}}(d)\}$, and $[\![t_2]\!]^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid d \in s^{\mathcal{I}} \text{ and } f^{\mathcal{I}}(d) = d\}$. ▷

We now provide the definition of the satisfaction of an OSF clause in an OSF algebra $\mathcal{I}$ under a variable assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$.

Definition 3.21 (Satisfaction of an OSF clause in an OSF algebra [10]). The *satisfaction of an OSF clause ϕ in a OSF algebra $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ under the assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$* (notation: $\mathcal{I}, \alpha \models \phi$) is defined recursively by letting

$$\begin{aligned}
\mathcal{I}, \alpha \models X : s &\quad\Leftrightarrow\quad \alpha(X) \in s^{\mathcal{I}}, \\
\mathcal{I}, \alpha \models X \doteq Y &\quad\Leftrightarrow\quad \alpha(X) = \alpha(Y), \\
\mathcal{I}, \alpha \models X.f \doteq Y &\quad\Leftrightarrow\quad f^{\mathcal{I}}(\alpha(X)) = \alpha(Y), \text{ and} \\
\mathcal{I}, \alpha \models \phi \,\&\, \phi' &\quad\Leftrightarrow\quad \mathcal{I}, \alpha \models \phi \text{ and } \mathcal{I}, \alpha \models \phi'.
\end{aligned}$$

If $\mathcal{I}, \alpha \models \phi$, then α is called a *solution* for the clause ϕ, and ϕ is said to be *satisfiable* in $\mathcal{I}$.

Example 3.22 (OSF clause satisfaction). Continuing from Example 3.18, consider the OSF clause $\phi = X : \mathit{movie} \,\&\, X.\mathit{directed\_by} \doteq X_0 \,\&\, X_0 : \mathit{director}$. Let α be an assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ such that $\alpha(X) = \mathit{psycho}$ and $\alpha(X_0) = \mathit{hitchcock}$. It follows that $\mathcal{I}, \alpha \models \phi$, since every constraint of ϕ is satisfied in $\mathcal{I}$ under the assignment α.

- $\mathcal{I}, \alpha \models X : \mathit{movie}$, since $\alpha(X) = \mathit{psycho} \in \mathit{movie}^{\mathcal{I}}$.
- $\mathcal{I}, \alpha \models X.\mathit{directed\_by} \doteq X_0$, since $\mathit{directed\_by}^{\mathcal{I}}(\alpha(X)) = \mathit{directed\_by}^{\mathcal{I}}(\mathit{psycho}) = \mathit{hitchcock} = \alpha(X_0)$.
- $\mathcal{I}, \alpha \models X_0 : \mathit{director}$, since $\alpha(X_0) = \mathit{hitchcock} \in \mathit{director}^{\mathcal{I}}$. ▷

Besides being syntactically equivalent, an OSF term t and its corresponding OSF clause $\phi(t)$ are also semantically equivalent, in the sense that the denotation of t in an OSF algebra $\mathcal{I}$ can be derived from the solutions for the clause $\phi(t)$ in $\mathcal{I}$.

Proposition 3.23 (Equivalence of term denotation and constraint satisfaction [10]). *For every OSF term $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$, for every interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ and for every assignment $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$:*

$$[\![t]\!]^{\mathcal{I},\alpha} = \begin{cases} \{\alpha(X)\} & \text{if } \mathcal{I}, \alpha \models \phi(t) \\ \emptyset & \text{otherwise} \end{cases}$$

and therefore

$$[\![t]\!]^\mathcal{I} = \{\alpha(X) \mid \alpha \in \mathit{Val}(\mathcal{I}) \text{ s.t. } \mathcal{I}, \alpha \models \phi(t)\}.$$

An important property of the constraint normalization rules is that the clause ϕ' resulting from the application of a rule of Fig. 3.2 to an OSF clause ϕ is semantically equivalent to the original clause ϕ, meaning that they share the same solutions across all OSF algebras. As we will illustrate in Chapter 4, this property is only partially retained in the fuzzy setting.

Proposition 3.24 (Solution-preservation of OSF clause normalization [10]). *The rules of Fig. 3.2 are solution-preserving, i.e., for any rule with premise ϕ and conclusion ϕ', for any OSF interpretation $\mathcal{I}$ and assignment α: $\mathcal{I}, \alpha \models \phi$ if and only if $\mathcal{I}, \alpha \models \phi'$.*

We conclude this section by exploring subalgebras of OSF algebras.

Definition 3.25 (Subalgebra of an OSF algebra). An OSF algebra $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is a *subalgebra* of an OSF algebra $\mathcal{J} = (\Delta^\mathcal{J}, \cdot^\mathcal{J})$ if $\Delta^\mathcal{I} \subseteq \Delta^\mathcal{J}$ and for all $d \in \Delta^\mathcal{I}$, $s \in \mathcal{S}$ and $f \in \mathcal{F}$: $s^\mathcal{I} = s^\mathcal{J} \cap \Delta^\mathcal{I}$ and $f^\mathcal{I}(d) = f^\mathcal{J}(d)$.

It is easy to see that, if $\mathcal{I}$ is a subalgebra of $\mathcal{J}$, then the denotation of an OSF term t under a valuation $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$ is the same in the two algebras. Similarly, an assignment $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$ is a solution for a clause ϕ in $\mathcal{I}$ if and only if it is a solution for ϕ in $\mathcal{J}$.

Proposition 3.26 (Denotation and satisfaction in subalgebras). *Let $\mathcal{I}$ be a subalgebra of an OSF algebra $\mathcal{J}$. For every OSF term t, every OSF clause ϕ, and every assignment $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$: (i) $[\![t]\!]^{\mathcal{I},\alpha} = [\![t]\!]^{\mathcal{J},\alpha}$ and (ii) $\mathcal{I}, \alpha \models \phi$ if and only if $\mathcal{J}, \alpha \models \phi$.*

In the next sections we will often consider specific subalgebras of a given algebra $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, namely those that can be generated by applying every possible feature composition to the elements of a subset D of $\Delta^\mathcal{I}$.

Definition 3.27 ($\mathcal{F}$-closure [10]). Let $\mathcal{I}$ be an OSF interpretation. For each sequence $w = f_1 \ldots f_n \in \mathcal{F}^*$ of features let $w^\mathcal{I} = f_n^\mathcal{I} \circ \ldots \circ f_1^\mathcal{I}$ be the corresponding function composition on $\Delta^\mathcal{I}$. For any non-empty subset D of $\Delta^\mathcal{I}$ the *$\mathcal{F}$-closure of D* is the set

$$\mathcal{F}^*(D) \stackrel{\text{def}}{=} \bigcup_{w \in \mathcal{F}^*} w^\mathcal{I}(D) = \bigcup_{w \in \mathcal{F}^*} \{w^\mathcal{I}(d) \mid d \in D\}.$$

In other words $\mathcal{F}^*(D)$ is the smallest set containing D and closed under feature applications.

Definition 3.28 (OSF subalgebra generated by a set [10]). Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an OSF interpretation and $D \subseteq \Delta^{\mathcal{I}}$ be nonempty. The *OSF subalgebra generated by* D is the structure $\mathcal{I}[D] = (\mathcal{F}^*(D), \cdot^{\mathcal{I}[D]})$ such that, for each $s \in \mathcal{S}$, $s^{\mathcal{I}[D]} \stackrel{\text{def}}{=} s^{\mathcal{I}} \cap \mathcal{F}^*(D)$, and for each $f \in \mathcal{F}$, $f^{\mathcal{I}[D]}$ is defined as the restriction of $f^{\mathcal{I}}$ to $\mathcal{F}^*(D)$.

When $D = \{d\}$ is a singleton we write $\mathcal{I}[d]$ instead of $\mathcal{I}[\{d\}]$.

Proposition 3.29 (OSF subalgebra generated by a set [10]). *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an OSF interpretation. For any non-empty subset $D \subseteq \Delta^{\mathcal{I}}$ the structure $\mathcal{I}[D]$ is the least subalgebra of $\mathcal{I}$ containing D.*

3.3 The OSF graph algebra

This section covers the OSF graph algebra $\mathcal{G}$, an OSF algebra that is essential for proving several semantic properties of OSF logic. The elements of this algebra are rooted directed labeled graphs called OSF graphs.

Definition 3.30 (OSF graph [10]). An *OSF graph* is a directed labeled graph $g = (N, E, \lambda_N, \lambda_E, X)$ such that

- $N \subseteq \mathcal{V}$ and $X \in N$ is a distinguished node called the *root* of g;
- $\lambda_N : N \to \mathcal{S}$ is a node labeling function such that each node of g is labeled by a non-bottom sort, i.e., $\lambda_N(N) \subseteq \mathcal{S} \setminus \{\bot\}$;
- $\lambda_E : E \to \mathcal{F}$ is an edge labeling function that assigns a feature to each edge $(Y, Z) \in E$ in such a way that no two edges outgoing from the same node are labeled by the same feature, i.e., if $\lambda_E(Y, Z) = \lambda_E(Y, Z')$ then $Z = Z'$; and
- every node lies on a directed path starting at the root.

The set of all OSF graphs is denoted $\Delta^{\mathcal{G}}$.

Example 3.31 (OSF graph). Let $g = (N, E, \lambda_N, \lambda_E, X_0)$ be the OSF graph such that

- $N = \{X_0, X_1, X_2, X, Y\}$;
- $E = \{(X_0, X_1), (X_0, X), (X_0, Y), (X, Y), (X, X_2), (Y, X)\}$;
- $\lambda_N = \{(X_0, \mathit{movie}), (X_1, \mathit{string}), (X_2, \mathit{string}), (X, \mathit{director}), (Y, \mathit{writer})\}$; and
- $\lambda_E = \left\{ \begin{array}{lll} ((X_0, X_1), \mathit{title}), & ((X_0, X), \mathit{directed\_by}), & ((X_0, Y), \mathit{written\_by}), \\ ((X, Y), \mathit{spouse}), & ((X, X_2), \mathit{name}), & ((Y, X), \mathit{spouse}) \end{array} \right\}$.

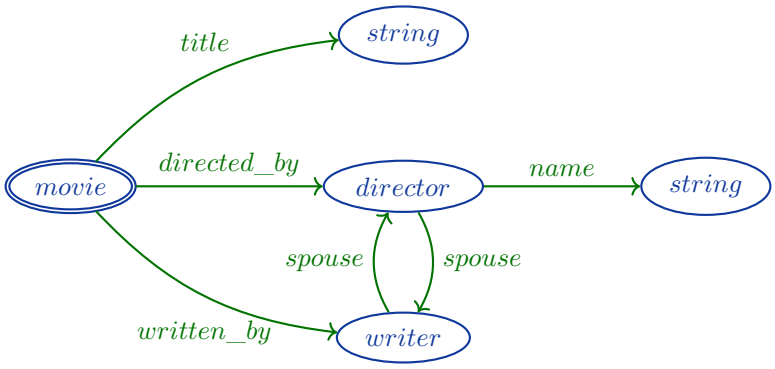


Figure 3.3: The OSF graph of Example 3.31.

The OSF graph g is depicted in Fig. 3.3, where the root node is identified with a double ellipse, and the node identifiers (i.e., the variables) are omitted. ▷

OSF graphs can also be seen as an alternative syntax for normal OSF terms, and consequently for rooted solved OSF clauses. Each of the three syntactic representations is valuable for different purposes, and being able to switch between them depending on the context is highly convenient. For instance, OSF terms offer a more concise representation, OSF clauses are essential for the constraint normalization procedure, and OSF graphs are often more practical for establishing results about this language. From an implementation perspective, Aït-Kaci argues that "the three views are important since the term view is the abstract syntax used by the user; the clausal view is the syntax used in the normalization rules presenting the operational semantics of constraint-solving; and, the graph view is the canonical representation used for implementation" [10].

The mappings between normal OSF terms and OSF graphs are defined as follows.

Definition 3.32 (Mapping from normal OSF terms to OSF graphs [10]). The mapping $G : \Psi \to \Delta^{\mathcal{G}}$ from normal OSF terms to OSF graphs is defined as follows.

- If $\psi = X : s$, then $G(\psi) \stackrel{\text{def}}{=} (\{X\}, \emptyset, \{(X, s)\}, \emptyset, X)$, i.e., $G(\psi)$ is the graph consisting of a single node X labeled s.

- If $\psi = X : s(f_1 \to \psi_1, \ldots, f_n \to \psi_n)$ and, for each $1 \leq i \leq n$, $G(\psi_i) = (N_i, E_i, \lambda_{N_i}, \lambda_{E_i}, X_i)$, then $G(\psi) \stackrel{\text{def}}{=} (N, E, \lambda_N, \lambda_E, X)$ where:

  - $N = \{X\} \cup \bigcup_{1 \leq i \leq n} N_i$,
  - $E = \{(X, X_i) \mid 1 \leq i \leq n\} \cup \bigcup_{1 \leq i \leq n} E_i$,
  - $\lambda_N(Y) = \begin{cases} s & \text{if } Y = X, \\ \lambda_{N_i}(Y) & \text{if } Y \in N_i \setminus (\{X\} \cup \bigcup_{1 \leq j < i} N_j), \end{cases}$

– $\lambda_E(e) = \begin{cases} f_i & \text{if } e = (X, X_i), \\ \lambda_{E_i}(e) & \text{if } e \in E_i. \end{cases}$

Definition 3.33 (Mapping from OSF graphs to normal OSF terms [10]). Assume without loss of generality a total ordering on $\mathcal{F}$, which induces a lexicographic ordering on $\mathcal{F}^*$. The mapping $\psi_G : \Delta^{\mathcal{G}} \to \Psi$ from OSF graphs to normal OSF terms is defined as follows. Let $g = (N, E, \lambda_N, \lambda_E, X)$ be such that $\lambda_N(X) = s$ and let $f_1, \ldots, f_n$, with $n \geq 0$, be the (pairwise distinct) features in $\mathcal{F}$ labeling all the edges outgoing from X. Then $\psi_G(g)$ can be constructed as the OSF term $X : s(f_1 \to \psi_1, \ldots, f_n \to \psi_n)$ where, for each $1 \leq i \leq n$:

- if the root Y_i of $g_i = f_i^{\mathcal{G}}(g)$ has already occurred earlier during the construction (in the predetermined ordering of $\mathcal{F}^*$), then $\psi_i \stackrel{\text{def}}{=} Y_i : \top$,
- otherwise, $\psi_i \stackrel{\text{def}}{=} \psi_G(g_i) = \psi_G(f_i^{\mathcal{G}}(g))$.

Proposition 3.34 (Bijections [10]). *The mappings*

$$\psi_\phi : \Phi_R \to \Psi, \quad \phi : \Psi \to \Phi_R, \quad \psi_G : \Delta^{\mathcal{G}} \to \Psi, \quad \textit{and} \quad G : \Psi \to \Delta^{\mathcal{G}}$$

are bijections between the sets Ψ, $\Delta^{\mathcal{G}}$ and Φ_R. More precisely,

$$\mathbf{1}_{\Phi_R} = \phi \circ \psi_\phi, \quad \mathbf{1}_\Psi = \psi_\phi \circ \phi = \psi_G \circ G, \quad \textit{and} \quad \mathbf{1}_{\Delta^{\mathcal{G}}} = G \circ \psi_G.$$

To improve readability, we simplify the notation by using expressions like $\phi(g)$ instead of $\phi(\psi_G(g))$, or $G(\phi)$ instead of $G(\psi_\phi(\phi))$. We will also simply write $\psi(g)$ for $\psi_G(g)$, and $\psi(\phi)$ for $\psi_\phi(\phi)$.

Example 3.35 (Mappings between OSF graphs, normal OSF terms and rooted solved OSF clauses). Consider the OSF term t from Example 3.4, the OSF clause ϕ from from Example 3.7, and the OSF graph g graph from Example 3.31: then $g = G(\psi) = G(\phi)$, $\psi = \psi(g) = \psi(\phi)$ and $\phi = \phi(g) = \phi(\psi)$. ▷

Having introduced OSF graphs and explored how they are syntactically related to OSF terms and clauses, we can proceed to examine how this collection of graphs can be endowed with an OSF algebraic structure.

Definition 3.36 (OSF graph algebra [10]). The *OSF graph algebra* is the structure $\mathcal{G} = (\Delta^{\mathcal{G}}, \cdot^{\mathcal{G}})$, where $\Delta^{\mathcal{G}}$ is the set of all OSF graphs, and where the interpretation of sorts and features is defined as follows.

- For each $s \in \mathcal{S}$: $s^{\mathcal{G}} \stackrel{\text{def}}{=} \{g = (N, E, \lambda_N, \lambda_E, X) \mid \lambda_N(X) \preceq s\}$.

- For each $f \in \mathcal{F}$ and $g = (N, E, \lambda_N, \lambda_E, X) \in \Delta^{\mathcal{G}}$, the function $f^{\mathcal{G}} : \Delta^{\mathcal{G}} \to \Delta^{\mathcal{G}}$ is defined as follows:

$$f^{\mathcal{G}}(g) \stackrel{\text{def}}{=} \begin{cases} g\big|_Y & \text{if } \exists Y \in N \text{ such that } \lambda_E(X, Y) = f, \\ G(Z_{f,g} : \top) & \text{otherwise,} \end{cases}$$

  where $g\big|_Y$ is the maximally connected subgraph of g rooted in Y, and $G(Z_{f,g} : \top)$ denotes the trivial OSF graph $(\{Z_{f,g}\}, \emptyset, \{(Z_{f,g}, \top)\}, \emptyset, Z_{f,g})$ whose only node is the new variable $Z_{f,g} \in \mathcal{V} \setminus N$ – labeled $\top$ – uniquely determined by the feature f and the graph g (i.e., if $f \neq f'$ or $g \neq g'$, then $Z_{f,g} \neq Z_{f',g'}$).

Example 3.37 (Interpretation of sorts in the OSF graph algebra). The interpretation of the sort symbols in the OSF graph algebra is straightforward: for each $s \in \mathcal{S}$, the set $s^{\mathcal{G}}$ contains any graph whose root is labeled by a sort s' such that $s' \preceq s$. Considering the sort signature of Example 3.2, for instance, the denotation of the sort *movie* is $movie^{\mathcal{G}} = \{movie(title \to string), horror(directed\_by \to director), horror(written\_by \to writer, directed\_by \to person), \ldots\}$. ▷

Example 3.38 (Interpretation of features in the OSF graph algebra). Consider a signature where $\mathcal{S} = \{s_0, \ldots, s_3\}$ and $\mathcal{F} = \{f_0, \ldots, f_4\}$. Fig. 3.4 shows how features are interpreted in the OSF graph algebra for this signature. OSF graphs are represented inside boxes. For a feature f, the application of $f^{\mathcal{G}}$ to a graph g is represented as an arrow originating from the box containing g and pointing at the box containing $f^{\mathcal{G}}(g)$. The figure shows the iterative application of a few features to the graph g corresponding to the term

$$s_0 \left(\begin{array}{lcl} f_2 & \to & s_1, \\ f_0 & \to & X : s_2 \left(\begin{array}{lcl} f_4 & \to & Y, \\ f_3 & \to & s_1 \end{array} \right), \\ f_1 & \to & Y : s_3 \left(\begin{array}{lcl} f_4 & \to & X \end{array} \right) \end{array} \right).$$

Note that the result of applying the function $f^{\mathcal{G}}$ to a graph that does not contain this feature results in a trivial graph (e.g., the application of $f_3^{\mathcal{G}}$ to g or to $f_1^{\mathcal{G}}(g)$). ▷

Proposition 3.39 (OSF graph algebra). *The OSF graph algebra $\mathcal{G}$ is an OSF algebra, i.e., it satisfies the conditions of Definition 3.17.*

A key property of the OSF graph algebra is that every solved OSF clause ϕ is satisfiable in a subalgebra of $\mathcal{G}$, namely the subalgebra generated by the graphs corresponding to the maximal rooted subclauses of ϕ.

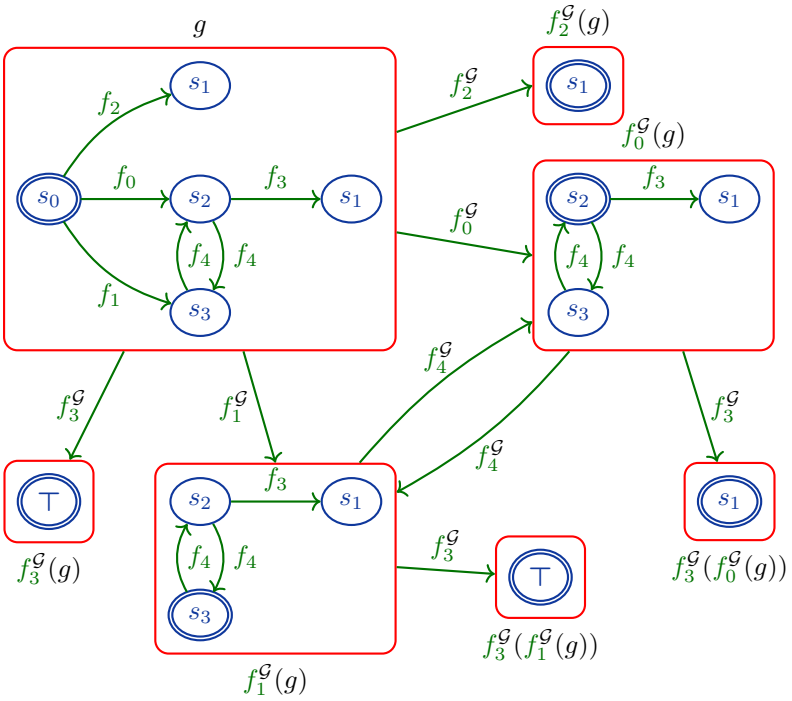


Figure 3.4: Feature applications in the OSF graph algebra.

Definition 3.40 (Canonical graph algebra [10]). Let ϕ be a solved OSF clause. The subalgebra $\mathcal{G}[\Delta^{\mathcal{G},\phi}]$ of the OSF algebra $\mathcal{G}$ generated by the set $\Delta^{\mathcal{G},\phi} \stackrel{\text{def}}{=} \{G(\phi(X)) \mid X \in \mathit{Tags}(\phi)\}$ is called the *canonical graph algebra* induced by ϕ.

Theorem 3.41 (Satisfiability in the canonical graph algebra [10]). *Any solved clause ϕ is satisfiable in $\mathcal{G}[\Delta^{\mathcal{G},\phi}]$ under any assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]}$ such that $\alpha(Y) = G(\phi(Y))$ for all $Y \in \mathit{Tags}(\phi)$.*

As a corollary of Proposition 3.26, we obtain that every solved OSF clause ϕ is satisfiable in $\mathcal{G}$ under any assignment that maps each variable $Y \in \mathit{Tags}(\phi)$ to the graph $G(\phi(Y))$. For this reason, such a mapping is called a *canonical solution* for the clause ϕ in $\mathcal{G}$.

Corollary 3.42 (Canonical solution in the OSF graph algebra [10]). *Every solved OSF clause ϕ is satisfiable in the OSF graph algebra $\mathcal{G}$ under any assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{G}}$ such that, for each $Y \in \mathit{Tags}(\phi)$, $\alpha(Y) = G(\phi(Y))$.*

3.4 OSF algebra homomorphisms

In this section we present OSF algebra homomorphisms, which constitute an essential tool for proving several results regarding OSF algebras.

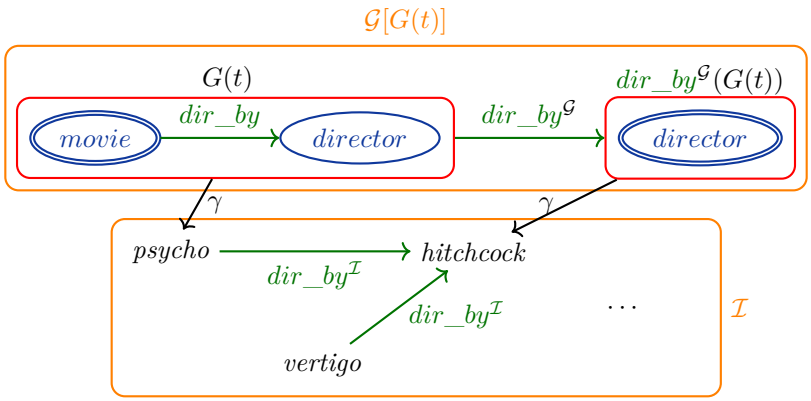


Figure 3.5: The OSF algebra morphism of Example 3.44.

Definition 3.43 (OSF algebra homomorphism [10]). An *OSF algebra homomorphism* $\gamma : \mathcal{I} \to \mathcal{J}$ between two OSF algebras $\mathcal{I}$ and $\mathcal{J}$ is a function $\gamma : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ such that

- for all $f \in \mathcal{F}$ and $d \in \Delta^{\mathcal{I}}$: $\gamma(f^{\mathcal{I}}(d)) = f^{\mathcal{J}}(\gamma(d))$;
- for all $s \in \mathcal{S}$: $\gamma(s^{\mathcal{I}}) \subseteq s^{\mathcal{J}}$, i.e., for all $d \in s^{\mathcal{I}}$, $\gamma(d) \in s^{\mathcal{J}}$.

In other words, an OSF algebra homomorphism is a function from an OSF algebra $\mathcal{I}$ into another OSF algebra $\mathcal{J}$ that preserves the structure of $\mathcal{I}$. The first condition of Definition 3.43 means that the morphism γ must commute with $f^{\mathcal{I}}$ and $f^{\mathcal{J}}$ for each $f \in \mathcal{F}$, while the second one states that γ must preserve the sort of each element of the algebra $\mathcal{I}$.

Example 3.44 (OSF algebra homomorphism). Consider the fuzzy interpretation $\mathcal{I}$, the assignment α and the term $t = X : \mathit{movie}\,(\mathit{directed\_by} \to X_0 : \mathit{director})$ from Examples 3.18 and 3.20. Consider the subalgebra of $\mathcal{G}$ generated from the element $G(t)$ and define a function $\gamma : \Delta^{\mathcal{G}[G(t)]} \to \Delta^{\mathcal{I}}$ by setting, for $g = w^{\mathcal{G}}(G(t)) \in \Delta^{\mathcal{G}[G(t)]}$, $\gamma(g) \stackrel{\text{def}}{=} w^{\mathcal{I}}(\mathit{psycho})$, where $w \in \mathcal{F}^*$. In particular $\gamma(G(t)) = \mathit{psycho}$ and $\gamma(G(Y : \mathit{director})) = \gamma(\mathit{directed\_by}^{\mathcal{G}}(G(t))) = \mathit{directed\_by}^{\mathcal{I}}(\mathit{psycho}) = \mathit{hitchcock}$. This is easily verified to be an OSF algebra homomorphism: for instance, $G(t) \in \mathit{movie}^{\mathcal{G}}$, and $\gamma(G(t)) = \mathit{psycho} \in \mathit{movie}^{\mathcal{I}}$. The two algebras and the homomorphism are depicted in Fig. 3.5 (where trivial graphs are not shown, and some names have been shortened). ▷

Since an OSF algebra morphism $\gamma : \mathcal{I} \to \mathcal{J}$ preserves the structure of $\mathcal{I}$ in $\mathcal{J}$, it seems intuitive that any clause that is satisfiable in $\mathcal{I}$ must also be satisfiable in $\mathcal{J}$.

Theorem 3.45 (Extending solutions through homomorphisms [10]). *Let $\mathcal{I}$ and $\mathcal{J}$ be two OSF algebras and $\gamma : \mathcal{I} \to \mathcal{J}$ be a homomorphism between them. For every OSF clause ϕ and assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$, if $\mathcal{I}, \alpha \models \phi$, then $\mathcal{J}, \alpha' \models \phi$, where $\alpha' = \gamma \circ \alpha$.*

More interestingly, it is possible to show that any solution for a clause ϕ in any OSF algebra $\mathcal{I}$ can be obtained as the homomorphic image of the canonical solution for ϕ in $\mathcal{G}$ (see Corollary 3.42).

Theorem 3.46 (Extracting solutions through homomorphisms [10]). *For any solved OSF clause ϕ, OSF algebra $\mathcal{I}$ and assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ such that $\mathcal{I}, \alpha \models \phi$ there exists an OSF algebra homomorphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ such that $\alpha(X) = \gamma(G(\phi(X)))$ for each $X \in Tags(\phi)$.*

As stated in the following theorem, it is always possible to define a homomorphism γ from any OSF algebra $\mathcal{I}$ into the OSF graph algebra $\mathcal{G}$. Thus, as a corollary of Theorem 3.45 we obtain that any solution α for an OSF clause ϕ in some OSF algebra $\mathcal{I}$ can be extended to a solution for ϕ in $\mathcal{G}$, meaning that any OSF clause is satisfiable if and only if it is satisfiable in the OSF graph algebra.

Theorem 3.47 (Weak finality of $\mathcal{G}$ [10]). *There exists a homomorphism γ from any OSF algebra $\mathcal{I}$ into the OSF graph algebra $\mathcal{G}$.*

Corollary 3.48 (Canonicity of the OSF graph algebra [10]). *An OSF clause is satisfiable if and only if it is satisfiable in the OSF graph algebra.*

Because of the semantic equivalence between OSF terms and OSF clauses (Proposition 3.23), the denotation of an OSF term in an OSF algebra $\mathcal{I}$ can be characterized through the existence of homomorphisms from the canonical graph algebra induced by $G(\psi)$ into $\mathcal{I}$.

Theorem 3.49 (Interpretability of canonical solutions [10]). *Let ψ be a normal OSF term, let $\phi = \phi(\psi)$, and let $\mathcal{I}$ be an OSF interpretation. Then*

$$[\![\psi]\!]^{\mathcal{I}} = \{\gamma(G(\psi)) \mid \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \text{ is an OSF algebra homomorphism}\}.$$

Remark 4. The original version of Theorem 3.49 from [8] states that, for any normal OSF term ψ and algebra $\mathcal{I}$:

$$[\![\psi]\!]^{\mathcal{I}} = \{\gamma(G(\psi)) \mid \gamma : \mathcal{G} \to \mathcal{I} \text{ is an OSF algebra homomorphism}\}.$$

While this statement is less cumbersome to read, it is also imprecise, as the next counterexample shows.

Consider $\mathcal{S} = \{\bot, s_0, s_1, \top\}$ such that $s_0 \wedge s_1 = \bot$, $s_0 \vee s_1 = \top$, and $\mathcal{F} = \{f\}$. Let $\psi = X : s_0(f \to X)$ and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be such that $\Delta^{\mathcal{I}} = \{a\}$, $f^{\mathcal{I}}(a) = a$, $s_0^{\mathcal{I}} = \{a\}$, and $s_1^{\mathcal{I}} = \emptyset$. It follows that $a \in [\![\psi]\!]^{\mathcal{I}}$ (consider any assignment α such that $\alpha(X) = a$).

Clearly there is no homomorphism $\gamma : \mathcal{G} \to \mathcal{I}$ such that $a = \gamma(G(\psi))$. If such a homomorphism γ existed, then it would also be defined on the graph $g = G(Y : s_1) \in \Delta^{\mathcal{G}}$, and, since $g \in s_1^{\mathcal{G}}$, then by Definition 3.43 it should hold that $\gamma(g) \in s_1^{\mathcal{I}}$, which is impossible since $s_1^{\mathcal{I}} = \emptyset$.

On the other hand, the function $\gamma : \Delta^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]} \to \Delta^{\mathcal{I}}$ (where $\phi = \phi(\psi)$) defined by letting $\gamma(g) = a$ constitutes a homomorphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$. To see this, note that

$\Delta^{\mathcal{G},\phi} = \{G(\psi)\}$, where $G(\psi)$ is an OSF graph composed of a single node labeled s_0 and a self loop labeled f, and thus $\Delta^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]} = \mathcal{F}^*(\Delta^{\mathcal{G},\phi}) = \{G(\psi)\}$.

Theorem 3.49 and Definition 3.50 have been rephrased to account for this kind of counterexamples.

3.5 Subsumption

In this section we show that the set Ψ of normal OSF terms, the set Φ_R of rooted solved OSF clauses, and the set $\Delta^{\mathcal{G}}$ of OSF graphs can each be endowed with a partial order[3]. More specifically, we define an approximation ordering on OSF graphs, an implication ordering on OSF clauses, and a subsumption ordering on OSF terms.

Establishing a decidable subsumption ordering between OSF terms is crucial for effective reasoning within OSF logic. This point is also underscored by Carpenter [34] with respect to the related *logic of typed feature structures*:

> "The primary goal in constructing our feature structures is to allow for the representation of partial information. We do not think of a feature structure as representing all that can be known about a domain object, but rather what is known at some particular stage in a computation. When dealing with partial information it is important to be able to tell when one piece of partial information is more informative or specific than another."

We start by presenting, for each OSF algebra $\mathcal{I}$, an approximation ordering on $\Delta^{\mathcal{I}}$ which is defined according to the existence of homomorphisms relating subalgebras of $\mathcal{I}$ generated by singletons.

Definition 3.50 (Endomorphic approximation [10]). On each OSF interpretation $\mathcal{I}$ a preorder $\sqsubseteq^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ is defined by letting, for all $d, d' \in \Delta^{\mathcal{I}}$:

$$d \sqsubseteq^{\mathcal{I}} d' \text{ if } \gamma(d) = d' \text{ for some homomorphism } \gamma : \mathcal{I}[d] \to \mathcal{I}[d']$$

If $d \sqsubseteq^{\mathcal{I}} d'$ we say that d *approximates* d'.

Proposition 3.51 (Endomorphic approximation preorder). *For each OSF interpretation $\mathcal{I}$, the relation $\sqsubseteq^{\mathcal{I}}$ is a preorder on $\Delta^{\mathcal{I}}$.*

The intuition behind this definition is more easily understood if we consider the approximation ordering defined on the OSF graph algebra $\mathcal{G}$.

[3] More precisely, we define *preorders* rather than partial orders on these sets, and show that antisymmetry holds modulo an equivalence relation. This situation is analogous to the subsumption ordering of first-order terms [91, 93], which is antisymmetric modulo variable renaming.

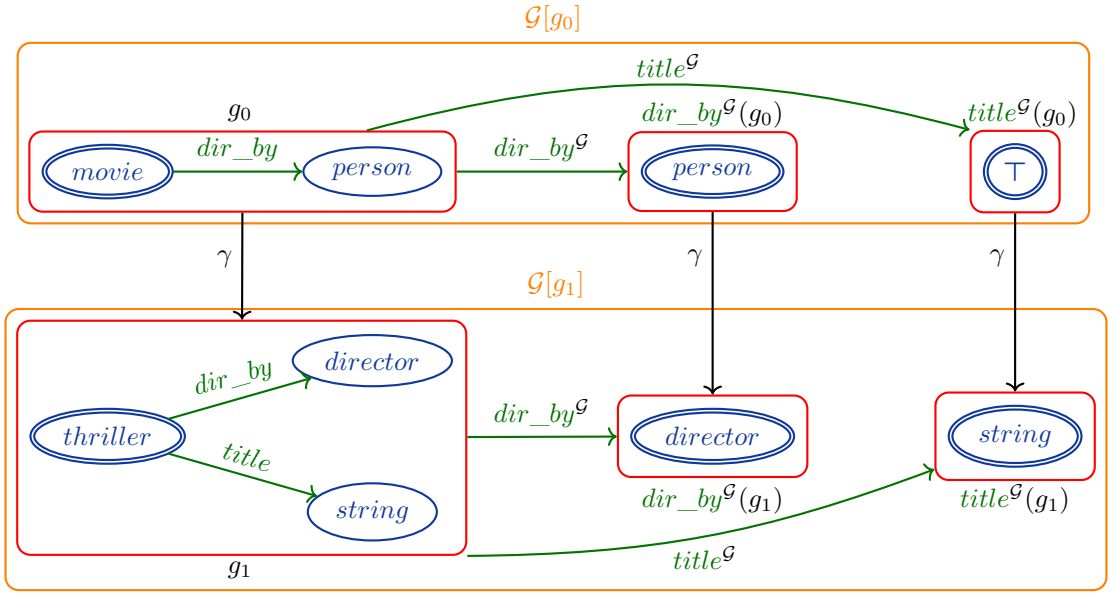


Figure 3.6: The OSF algebra morphism of Example 3.52.

Example 3.52 (OSF graphs approximation ordering). Consider the OSF signature of Example 3.2 and let

$$g_0 = G(X_0 : movie\,(directed\_by \to Y_0 : person)) \text{ and}$$
$$g_1 = G(X_1 : thriller\,(directed\_by \to Y_1 : director, title \to Z_1 : string))$$

Define $\gamma : \Delta^{\mathcal{G}[g_0]} \to \Delta^{\mathcal{G}[g_1]}$ by letting, for each $g = w^{\mathcal{G}}(g_0)$ in $\Delta^{\mathcal{G}[g_0]}$ (with $w \in \mathcal{F}^*$): $\gamma(g) \stackrel{\text{def}}{=} w^{\mathcal{G}}(g_1)$. For example, $\gamma(g_0) = g_1$, and $\gamma(directed\_by^{\mathcal{G}}(g_0)) = directed\_by^{\mathcal{G}}(g_1)$. This is depicted in Fig. 3.6 (where not all trivial graphs are shown, and a few feature names are abbreviated). The function γ is a morphism witnessing $g_0 \sqsubseteq^{\mathcal{G}} g_1$. In particular, $g_0 \in movie^{\mathcal{G}}$ and $g_1 = \gamma(g_0) \in thriller^{\mathcal{G}} \subseteq movie^{\mathcal{G}}$, and the feature applications commute with the function γ. Intuitively, the graph g_0 approximates g_1 as it represents information that is consistent with g_1 in a more general way: g_0 is a *movie* rather than a *thriller*, it is directed by a *person* rather than by a *director*, and it does not constrain the feature *title* to be of any sort more specific than $\top$. ▷

Thanks to Proposition 1.20, we know that $\sqsubseteq^{\mathcal{G}}$ can be extended to a partial order among equivalence classes of OSF graphs. As the next proposition illustrates, the elements of these equivalence classes are OSF graphs that are essentially the same except for different variable names or possibly the presence of trivial subgraphs.

Proposition 3.53 (OSF graph equivalence). *Let g_0 and g_1 be two OSF graphs. If $g_0 \sqsubseteq^{\mathcal{G}} g_1$ (as witnessed by a morphism $\gamma_0 : \mathcal{G}[g_0] \to \mathcal{G}[g_1]$) and $g_1 \sqsubseteq^{\mathcal{G}} g_0$ (as witnessed by a morphism $\gamma_1 : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$), then*

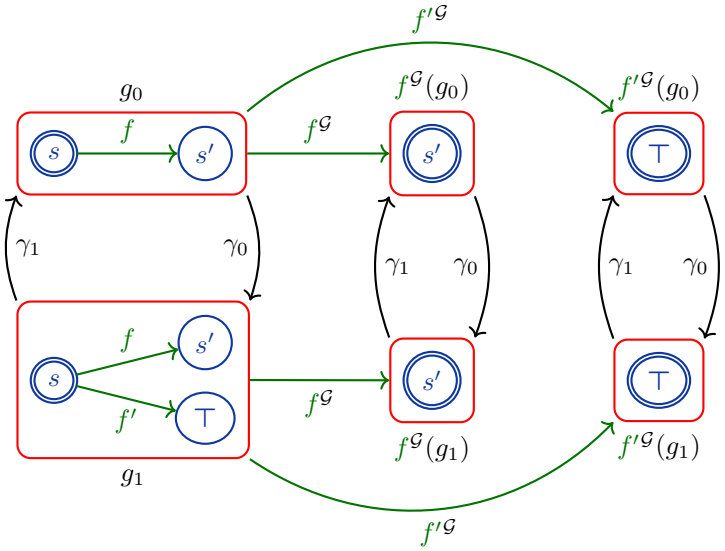


Figure 3.7: The OSF graphs and morphisms of Example 3.54.

1. *for all* $g \in \mathcal{G}[g_0]$, g *and* $\gamma_0(g)$ *are labeled by the same sort;*
2. *for all* $g \in \mathcal{G}[g_1]$, g *and* $\gamma_1(g)$ *are labeled by the same sort;*
3. $\gamma_0 \circ \gamma_1 = id_{\Delta^{\mathcal{G}[g_1]}}$ *and* $\gamma_1 \circ \gamma_0 = id_{\Delta^{\mathcal{G}[g_0]}}$*, and thus* γ_0 *and* γ_1 *are bijections.*

Example 3.54 (OSF graph equivalence). Consider the OSF terms

$$\psi_0 = X_0 : s(f \to Y_0 : s') \text{ and } \psi_1 = X_1 : s(f \to Y_1 : s', f' \to Z_1 : \top)$$

and let $g_0 = G(\psi_0)$ and $g_1 = G(\psi_1)$. Let $\gamma_0 : \mathcal{G}[g_0] \to \mathcal{G}[g_1]$ be defined by letting, for all $g = w^{\mathcal{G}}(g_0) \in \mathcal{G}[g_0]$ with $w \in \mathcal{F}^*$, $\gamma_0(g) \stackrel{\text{def}}{=} w^{\mathcal{G}}(g_1)$. Similarly, let $\gamma_1 : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$ be defined by letting, for all $g = w^{\mathcal{G}}(g_1) \in \mathcal{G}[g_1]$ with $w \in \mathcal{F}^*$, $\gamma_1(g) \stackrel{\text{def}}{=} w^{\mathcal{G}}(g_0)$. In particular $\gamma_0(g_0) = g_1$ and $\gamma_1(g_1) = g_0$. These are easily verified to be homomorphisms witnessing $g_0 \sqsubseteq^{\mathcal{G}} g_1$ and $g_1 \sqsubseteq^{\mathcal{G}} g_0$. The OSF graphs and morphisms are depicted in Fig. 3.7. ▷

It is now possible to characterize the denotation of a normal OSF term ψ in the OSF graph algebra $\mathcal{G}$ as the set of OSF graphs that are approximated by $G(\psi)$.

Corollary 3.55 (Denotation of an OSF term in $\mathcal{G}$ [10]). *Let* ψ *be a normal OSF term. The denotation of* ψ *in the OSF graph algebra* $\mathcal{G}$ *can be characterized as follows:*

$$[\![\psi]\!]^{\mathcal{G}} = \{g \in \Delta^{\mathcal{G}} \mid G(\psi) \sqsubseteq^{\mathcal{G}} g\}.$$

It can also be proved that, for any OSF algebra $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and $d, d' \in \Delta^{\mathcal{I}}$ such that $d \sqsubseteq^{\mathcal{I}} d'$, whenever d belongs to the denotation a term ψ, then d' must also belong to the

denotation of the same term. Intuitively, if an object d satisfies the constraints expressed by an OSF term, then the same must also hold for any object d' that is more specific than d. Moreover, the opposite direction is also true, that is, if two objects d and d' are such that $d' \in [\![\psi]\!]^{\mathcal{I}}$ whenever $d \in [\![\psi]\!]^{\mathcal{I}}$ for any $\psi \in \Psi$, then it must be the case that $d \sqsubseteq^{\mathcal{G}} d'$. As stated in [10], this result was originally established by Dörre and Rounds [43].

Corollary 3.56 (Dörre-Rounds [10, 43]). *For all OSF interpretations $\mathcal{I}$ and elements $d, d' \in \Delta^{\mathcal{I}}$:*

$$d \sqsubseteq^{\mathcal{I}} d' \text{ iff for all OSF terms } \psi,\ d' \in [\![\psi]\!]^{\mathcal{I}} \text{ whenever } d \in [\![\psi]\!]^{\mathcal{I}}.$$

By Definition 3.17 we know that, whenever s_0 is subsumed by s_1 (i.e., $s_0 \preceq s_1$), then $s_0^{\mathcal{I}} \subseteq s_1^{\mathcal{I}}$ holds in any OSF algebra $\mathcal{I}$. The subsumption ordering on OSF terms is an extension of this ordering on sorts: a term t_0 is subsumed by a term t_1 if the denotation of t_0 is always included in the denotation of t_1 in any OSF algebra.

Definition 3.57 (Semantic OSF term subsumption [10]). *The OSF term t_0 is subsumed by the OSF term t_1* (denoted $t_0 \preceq t_1$) if, for all OSF interpretations $\mathcal{I}$, $[\![t_0]\!]^{\mathcal{I}} \subseteq [\![t_1]\!]^{\mathcal{I}}$.

OSF clauses are instead ordered by logical consequence or implication: the clause ϕ_1 is a logical consequence of ϕ_0 if, for any OSF algebra $\mathcal{I}$ and assignment α that satisfies ϕ_0 it is possible to define an assignment α' that agrees with α on the variables shared by ϕ_0 and ϕ_1 and such that $\mathcal{I}, \alpha' \models \phi_1$.

Definition 3.58 (OSF clause implication [10]). *The OSF clause ϕ_0 implies the OSF clause ϕ_1* (denoted $\phi_0 \models \phi_1$) if, for all OSF interpretations $\mathcal{I}$ and assignments α such that $\mathcal{I}, \alpha \models \phi$, there is an assignment α' such that

1. $\alpha'(X) = \alpha(X)$ for all $X \in \mathit{Tags}(\phi_0) \cap \mathit{Tags}(\phi_1)$, and
2. $\mathcal{I}, \alpha' \models \phi_1$.

The implication ordering for rooted OSF clauses is defined similarly, and it requires that the roots of the two clauses are assigned the same value.

Definition 3.59 (Rooted OSF clause implication [10]). Let ϕ_X and ϕ'_Y be two rooted OSF clauses with no common variables. *The OSF clause ϕ_X implies the clause ϕ'_Y* (denoted $\phi_X \models \phi'_Y$) if $\phi \models \phi'[X/Y]$.

The following theorem establishes that the three orderings are equivalent.

Theorem 3.60 (Semantic transparency of orderings [10]). *If the normal OSF terms ψ and ψ' (with roots Y and X, respectively, and no common variables), the OSF graphs g and g', and the rooted solved OSF clauses ϕ_Y and ϕ'_X respectively correspond to one another*

though the syntactic mappings, then the following are equivalent: (1) $g \sqsubseteq^{\mathcal{G}} g'$*, (2)* $\psi' \preceq \psi$*, (3)* $\phi'_X \models \phi_Y$*, and (4)* $[\![\psi']\!]^{\mathcal{G}} \subseteq [\![\psi]\!]^{\mathcal{G}}$*.*

Similarly to the approximation ordering on OSF graph, the subsumption ordering on OSF terms and the implication ordering on OSF clauses are in fact preorders, which can be lifted to partial orders between equivalence classes of these objects as per Proposition 1.20.

Example 3.61 (OSF term and OSF clause equivalence). Consider the terms ψ_0 and ψ_1 and the graphs $g_0 = G(\psi_0)$ and $g_1 = G(\psi_1)$ from Example 3.54, and let $\phi_0 = \phi(\psi_0)$ and $\phi_1 = \phi(\psi_1)$. As shown in Example 3.54 it holds that $g_0 \sqsubseteq^{\mathcal{G}} g_1$ and $g_1 \sqsubseteq^{\mathcal{G}} g_0$, which by Theorem 3.60 implies that $\psi_0 \preceq \psi_1$, $\psi_1 \preceq \psi_0$, $\phi_0 \models \phi_1$ and $\phi_1 \models \phi_0$. Indeed, because features are interpreted as *total functions*, the terms ψ_0 and ψ_1 are equivalent, i.e., they have the same denotation in every OSF algebra. Analogously, the two OSF clauses ϕ_0 and ϕ_1 are equivalent, since for any assignment α_0 that satisfies ϕ_0 in an OSF algebra $\mathcal{I}$ it is possible to define an assignment α_1 that satisfies ϕ_1 in $\mathcal{I}$, and vice versa. ▷

To conclude this section, we provide an additional characterization of subsumption among OSF terms, framing it as a homomorphic mapping between their tags.

Definition 3.62 (Syntactic OSF term subsumption). *The (normal) OSF term* ψ_0 *is (syntactically) subsumed by the (normal) OSF term* ψ_1 (denoted $\psi_0 \trianglelefteq \psi_1$) if there is a mapping $h : \mathit{Tags}(\psi_1) \to \mathit{Tags}(\psi_0)$ such that:

1. $h(\mathit{RootTag}(\psi_1)) = \mathit{RootTag}(\psi_0)$;
2. $\mathit{Sort}_{\psi_0}(h(X)) \preceq \mathit{Sort}_{\psi_1}(X)$ for each $X \in \mathit{Tags}(\psi_1)$;
3. If $X \xrightarrow{f}_{\psi_1} Y$, then $h(X) \xrightarrow{f}_{\psi_0} h(Y)$.

The equivalence of the semantic (Definition 3.57) and syntactic definitions of OSF term subsumption is given next.

Proposition 3.63 (Semantic and syntactic subsumption). *If* ψ_0 *and* ψ_1 *are consistent OSF terms, then* $\psi_0 \preceq \psi_1$ *if and only if there are two (normal) OSF terms* ψ'_0 *and* ψ'_1 *such that*

- $[\![\psi_0]\!]^{\mathcal{I}} = [\![\psi'_0]\!]^{\mathcal{I}}$ *for all* $\mathcal{I}$*,*
- $[\![\psi_1]\!]^{\mathcal{I}} = [\![\psi'_1]\!]^{\mathcal{I}}$ *for all* $\mathcal{I}$*,*
- $\psi'_0 \trianglelefteq \psi'_1$*.*

3.6 Unification

Unification is an essential operation in automated reasoning, and is at the core of the application of OSF logic and its variants in computational linguistics [34] and the implementation of logic programming languages based on OSF terms [8, 10]. More recently, it has enabled the implementation of the CEDAR Semantic Web reasoner [6, 15]. Informally, the unifier of two OSF terms ψ_1 and ψ_2 is a term ψ that consistently combines the constraints expressed by the two terms, without adding any extra constraints. In other words, it is the most general term that is subsumed by both ψ_1 and ψ_2.

Definition 3.64 (OSF term unification [10]). The *unifier* of two normal OSF terms ψ_1 and ψ_2 is their GLB in the OSF term subsumption ordering and is denoted $\psi_1 \wedge \psi_2$.

Recall that applying the OSF constraint normalization rules to an OSF clause results in a normal form ϕ that is either the inconsistent clause $X : \bot$, or an OSF clause in solved form together with a conjunction of equality constraints (Proposition 3.15). The subclause of ϕ in solved form is denoted $Solved(\phi)$.

Theorem 3.65 (OSF term unification [10]). *Let ψ_1 and ψ_2 be normal OSF terms, and let ϕ be the OSF clause obtained by non-deterministically applying any applicable constraint normalization rule (Fig. 3.2) to the clause*

$$\phi(\psi_1) \;\&\; \phi(\psi_2) \;\&\; RootTag(\psi_1) \doteq RootTag(\psi_2)$$

until none applies. Then, ϕ is the inconsistent clause iff the GLB of ψ_1 and ψ_2 is $X : \bot$. If ϕ is not the inconsistent clause, then $\psi_1 \wedge \psi_2 = \psi(Solved(\phi))$.

Note that two normal OSF terms always have a GLB (possibly $\bot$) in the OSF term subsumption ordering, which can be computed via their unification. Normal OSF terms are thus ordered in a subsumption lattice.

Example 3.66 (OSF term unification). Consider the OSF signature of Example 3.2 and the OSF terms

$$\psi_1 = X : movie \left(\begin{array}{lll} directed\_by & \rightarrow & X_0 : director, \\ title & \rightarrow & X_1 : string \end{array} \right)$$

and $\psi_2 = Y : horror\,(directed\_by \rightarrow Y_0 : person)$. Their unifier can be computed by considering the constraints

$$\begin{array}{lcllcl} \phi(\psi_1) & = & X : movie & \& & X.directed\_by \doteq X_0 & \& \quad X_0 : director \\ & & & \& & X.title \doteq X_1 & \& \quad X_1 : string \end{array}$$

and $\phi(\psi_2) = Y : horror \;\&\; X.directed\_by \doteq Y_0 \;\&\; Y_0 : person$ and applying the OSF constraint normalization rules of Fig. 3.2 to the clause $\phi(\psi_1)\;\&\;\phi(\psi_2)\;\&\;X \doteq Y$, resulting in the OSF clause

$$\begin{array}{lllll} X : horror & \& & X.directed\_by \doteq X_0 & \& & X_0 : director \\ & \& & X.title \doteq X_1 & \& & X_1 : string \\ & \& & X \doteq Y & \& & X_0 \doteq Y_0 \end{array}$$

or an equivalent clause. Translating this clause back into an OSF term yields

$$X : horror \left(\begin{array}{lll} directed\_by & \rightarrow & X_0 : director, \\ title & \rightarrow & X_1 : string \end{array} \right)$$

which is the unifier of ψ_1 and ψ_2, i.e., their GLB in the OSF term subsumption lattice. ▷

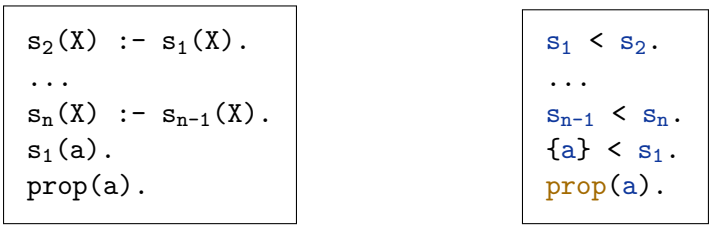
With respect to the complexity of computing the unifier of two OSF terms ψ_1 and ψ_2, the algorithm from [8] based on the union-find problem [1] has a worst-case complexity of $O(mG(m))$, where $m = |Tags(\psi_1) \cup Tags(\psi_2)|$ and the growth rate of the function G is of the order of an inverse of the Ackermann function ($G(m) \leq 5$ for all practical purposes) [8]. The computation of the unifier for two OSF terms hinges, in particular, on determining the GLB of two sorts within the sort subsumption lattice, as evidenced by the Sort Intersection rule. Various strategies for optimizing this operation are discussed in Section 3.8.

3.7 Applications

3.7.1 LOGIN: OSF logic and definite clauses

One of the primary applications of OSF logic involves its integration into Prolog, by replacing its first-order terms (FOTs) with OSF terms, resulting in the language LOGIN (Logic and Inheritance) [8]. There are several advantages to adopting OSF terms rather than FOTs in the context of a logic programming language, as highlighted in [8]. For instance, one limitation of FOTs is that its functor symbols have a fixed arity, while an OSF logic signature allows terms with the same root sort but with possibly a different number of arguments, such as $movie(directed\_by \rightarrow director, year \rightarrow 1960)$ and $movie(directed\_by \rightarrow person)$. Moreover, the arguments of an OSF term are identified by features rather than positions, aiding the interpretability of a term.

The main motivation behind the integration of OSF terms into Prolog is, however, a more natural, and possibly more efficient, implementation of inheritance (i.e., an is-a subsumption relation) directly into the unification process rather than through SLD resolution. Indeed, a key limitation of Prolog is that the unification of two FOTs fails whenever a mismatch of functor symbols occurs, which may not always be desirable. For instance, the unification of



(a) Prolog version (b) OSF version

Figure 3.8: The logic programs of Example 3.67.

`person(X)` and `student(alice)` fails in Prolog, even under the assumption that `student` is subtype of (or is subsumed by) `person`. While Prolog allows to express this subsumption through the rule `person(X) :- student(X)`, the authors of [8] argue that this approach is not satisfactory in practice, as inheritance is achieved by an inference step, lengthening proofs. Replacing the FOTs of Prolog with OSF terms offers the advantage of integrating inheritance into the unification process, since OSF term unification takes into account a subsumption relation on sorts, for instance allowing the unification of OSF terms such as $X : \mathit{person}$ and $\mathit{student}(\mathit{name} \to \text{“}\mathit{Alice}\text{”})$, provided that $\mathit{student} \preceq \mathit{person}$. In general, this property of OSF logic may lead a single unification step to replace several resolutions steps, as the following example from [8] shows.

Example 3.67 (Prolog resolution and OSF unification). Consider the Prolog program of Fig. 3.8a. The query `?- sn(X), prop(X)` will require n resolution steps before matching `X = a`. The OSF version of the same program is depicted in Fig. 3.8b. The program involves declarations of shape `s < s'` for the sort subsumption relation, and of shape `{a} < s` for the instances of the sort symbols. Constants such as `a` are treated as singleton sorts (sorts that denote a single element), and thus as OSF terms themselves. The symbol `prop` is a predicate symbol which takes OSF terms as arguments. Since `a` is subsumed by $\mathtt{s_n}$, now the query `prop(X:sn)` (which aims to retrieve individuals `X` of sort $\mathtt{s_n}$ and that satisfy the property `prop`) succeeds in a single unification step rather than n resolution steps. ▷

The advantage of performing a single unification step rather than several SLD resolution steps might not seem significant at first, especially considering that OSF term unification relies on the computation of the GLB of two sorts in a lattice, a potentially expensive operation. As will be seen in Section 3.8, however, this operation can be implemented very efficiently thanks to graph encoding techniques that reduce the computation of a GLB to a simple bitwise AND on binary strings, at the cost of a one-time preprocessing of the sort subsumption graph. As a result, replacing a number of SLD resolution steps with a single OSF term unification can indeed lead to more efficient computations.

To conclude the section, the following example illustrates a LOGIN program in which the computation of a solution to a given query relies on SLD resolution on predicate symbols

```
% Subsumption relation
director < person.
slasher < horror.
horror < movie.
thriller < movie.
% Instances
{ alinda } < person.
{ hitchcock, carpenter, nolan } < director.
{ psycho } < horror.
{ halloween } < slasher.
{ memento } < thriller.
% Facts
director_of(hitchcock, psycho).
director_of(carpenter, halloween(year -> 1979)).
director_of(nolan, memento(title -> "Memento")).
% Rules
likes(alinda, Y :thriller) :- director_of(X : person, Y).
```

Figure 3.9: A logic program with OSF terms.

and on OSF term unification. Predicate symbols in this context are distinct from the sorts and features of OSF terms. They work exactly as in Prolog, with the only difference that they accept OSF terms rather than first-order terms as arguments.

Example 3.68 (SLD resolution in LOGIN). Consider the program of Fig. 3.9. After the definition of the sort subsumption relation and of the instances of a few sorts, the program specifies a few facts regarding the binary predicate `director_of`. Finally, a rule involving the predicates `director_of` and `likes` states that `alinda` likes thriller movies. The query `?- likes(alinda, Y : movie)` is first reduced by resolution to the goal `director_of(X : person, Y : thriller)`, which is then resolved against the facts of the program, returning a single solution binding `Y` to `memento(title -> "Memento")`, through the unification of `thriller` and `memento(title -> "Memento")`. ▷

LOGIN was implemented as a part of LIFE (Logic, Inheritance, Functions, and Equations) a programming language which incorporates elements from functional, logic and object-oriented programming [7, 14].

3.7.2 CEDAR: OSF logic for the Semantic Web

Two versions of a Semantic Web reasoner based on OSF logic were implemented in the context of the *CEDAR project* [6, 15], an ANR Chair of Excellence Project led by Hassan

Aït-Kaci.[4] The goal of the project was to develop, implement and test a constraint-based approach to KRR based on the OSF framework in order to provide an effective, efficient and scalable technology that is able to represent knowledge and reason about data, aiming to prove that OSF logic provides a sound alternative to DL as a language for the Semantic Web, and that it is in fact able to overcome some of the limitations of DL as reported, for instance, in [42, 54, 109].

An extension of the CEDAR reasoner that leverages a similarity relation between sort symbols in order to provide approximate answers to a query will be discussed in Section 5.5.1, which deals with similarity-based OSF logic.

CEDAR reasoner V1 The first version of the CEDAR reasoner [5, 6] is a *taxonomic reasoner* that is capable of *classifying a taxonomy* and *answering Boolean queries*. In the context of the CEDAR project, a taxonomy is a partial order $(\mathcal{S}, \preceq)$, i.e., a set $\mathcal{S}$ of set-denoting sort symbols together with an is-a ordering $\preceq$ on $\mathcal{S}$ which represents set inclusion. The taxonomy $(\mathcal{S}, \preceq)$ is specified in CEDAR through *subsort declarations* of shape $s \lhd s'$. Assuming that they do not involve cycles, these declarations can be also be represented as a DAG $(\mathcal{S}, \lhd)$ such that $\preceq$ is equal to the reflexive and transitive closure of $\lhd$. It can be assumed without loss of generality that $\mathcal{S}$ contains a greatest sort $\top$ and a least sort $\bot$, which respectively denote the whole domain of interest and the empty set $\emptyset$.

The task of *classifying* a taxonomy consists in computing the reflexive and transitive closure of the subsort declarations $(\mathcal{S}, \lhd)$ given as input. After classifying a taxonomy, the CEDAR reasoner is able to decide whether a sort is subsumed by another sort, and to *answer Boolean queries* on the elements of the taxonomy, that is, queries involving conjunctions, disjunctions an negations on sort symbols. Formally, given two sorts $s_0, s_1 \in \mathcal{S}$, (i) a *conjunctive query* $s_0 \wedge s_1$ aims to find the GLB of s_0 and s_1 in $\mathcal{S}$, that is, the most general sort s that is subsumed by both s_0 and s_1, which corresponds to their intersection; (ii) a *disjunctive query* $s_0 \vee s_1$ aims to find the least upper bound (LUB) of s_0 and s_1 in $\mathcal{S}$, that is, the most specific sort s that subsumes both s_0 and s_1, which corresponds to their union; (iii) a *negated query* $\neg s_0$ aims to find the complement of the sort s_0.

The classification of a taxonomy is performed by the CEDAR reasoner by implementing a bottom-up DAG encoding algorithm originally proposed in [13], which consists in associating each sort $s \in \mathcal{S}$ with a vector of bits of length $O(|\mathcal{S}|)$. After the classification is complete, for instance, a conjunctive query $s_0 \wedge s_1$ can be answered simply by computing the bitwise AND of the bit vectors corresponding to s_0 and s_1. The classification algorithm is discussed in more detail in Section 3.8, which deals with the implementation of OSF logic.

The performance of the first version of the CEDAR reasoner was compared with that of state-of-the-art Semantic Web reasoners based on DL – including FaCT++ [116], HermiT

[4] More information about the CEDAR project can be found at `https://cedar.liris.cnrs.fr/`, including reports, papers, demos and software.

[52, 102], Pellet [105], TrOWL [115], RacerPro [56], and SnoRocket [75] – on several taxonomies of varying size. The evaluation involved the tasks of classifying a taxonomy and answering Boolean queries combining conjunctions, disjunctions and negations. The results showed that the CEDAR taxonomic reasoner was among the fastest with respect to classification, and orders of magnitude more efficient with respect to answering Boolean queries [5, 6].

As experimentally shown in [6], a key advantage of the CEDAR taxonomic reasoner is the efficiency of performing Boolean queries, which is virtually $O(1)$ irrespective of the size of the taxonomy [6]. Because each sort $s \in \mathcal{S}$ is encoded as a bit-vector of length $O(|\mathcal{S}|)$, however, there is a trade-off with respect to the space required by this representation, which is quadratic in the size of the taxonomy, and may thus become unfeasible in practice for very large taxonomies. Nevertheless, the CEDAR reasoner was applied to the NCBI (National Center for Biotechnology Information) taxonomy[5] consisting of 903617 sorts, which was, according to the authors, the largest taxonomy available at the time of the experiments [6]. Moreover, alternative encoding strategies that are more space efficient and that maintain efficient (logarithmic) Boolean operations are proposed in [13] and [6].

CEDAR reasoner V2 The second iteration of the CEDAR reasoner extends its capabilities by introducing support for reasoning not only with bare sorts but also with OSF features and DL roles. More specifically, in addition to performing the classification of an is-a taxonomy and answering Boolean queries, the enhanced CEDAR reasoner can carry out the following tasks [15, 16]:

- reasoning over OSF structures involving OSF feature symbols, domain and range specifications for such features, aggregate sorts representing collections of instances of specific sorts, and universal and existential restrictions;
- verifying the consistency of a query expressed as an OSF term by normalizing it with respect to the classified taxonomy; and
- applying the OSF constraint normalization rules to optimize a query before translating it into SPARQL[6] for the retrieval of RDF triples, an operation that can significantly reduce the retrieval search space.

Furthermore, [15, 16] define an RDF format for OSF structures, establishing a mapping between the two representations, and propose an indexing scheme for RDF triples that takes advantage of the semantic information provided by OSF sorts and attributes, facilitating efficient instance retrieval.

[5] `https://www.ncbi.nlm.nih.gov/Taxonomy/taxonomyhome.html/`.
[6] `https://www.w3.org/TR/sparql11-query/`.

An extended OSF logic syntax is also adopted in the updated CEDAR reasoner in order to supports a few DL constructs [3, 4]. For instance, one of the most notable differences between OSF logic and DLs lies in their interpretation of attributes. In the OSF formalism attributes are interpreted as functional features, whereas in DL they denote relational roles. However, because relations correspond to set-valued functions[7], the syntax of OSF logic can be enriched with the construct **set-of**(s), whose meaning in an OSF interpretation $\mathcal{I}$ is $[\![\textbf{set-of}(s)]\!]^{\mathcal{I}} = \mathcal{P}(s^{\mathcal{I}})$ [3, 4]. As a consequence, a relation R between two sorts s_0 and s_1 may be seen as a feature f_R from s_0 to **set-of**(s_1). Moreover, the OSF constraint normalization rules of Fig. 3.2 do not need to be altered in order to support the **set-of** construct [3, 4]. Other extensions of the OSF logic syntax, including the support for the existential and universal restrictions of DLs, are discussed in more detail in [4].

The updated version of the CEDAR reasoner also verifies the consistency of the taxonomy with respect to a given set of *domain and range declarations* for feature symbols. These consist in expressions of shape $f : s_1 \to s_2$, stating that the function denoted by f takes as input an object of sort s_1 and returns an object of sort s_2. The consistency check for feature declarations involves confirming that, for each feature f, there are no declarations $f : s_1 \to s_2$ and $f : s'_1 \to s'_2$ such that $s'_1 \preceq s_1$ and $s_2 \wedge s'_2 = \bot$. Otherwise, if $s_2 \wedge s'_2 \neq \bot$, then the initial declaration can be updated to $f : s'_1 \to s_2 \wedge s'_2$. This process is repeated for every declaration of shape $f : s_1 \to s_2$ and each subsort of s_1 [15, 16].

Another key feature of the CEDAR reasoner V2 lies in its query optimization step [15]. Before being executed, a query (expressed as an OSF term) is normalized according to the OSF constraint normalization rules and the knowledge expressed in the taxonomy, including subsort declarations and feature domain and range declarations. Moreover, this step also ensures the consistency of the input query, so that no answer is provided if the query is inconsistent. If consistent, the normalized OSF query is translated into a SPARQL query in order to retrieve instances. As argued in [15], the normalization of an OSF query before its translation to SPARQL can significantly reduce the retrieval search space by only focusing on the relevant instances, and it has the potential to improve the query evaluation by eliminating unnecessary joins.

Example 3.69 (CEDAR V2: query normalization and retrieval). Consider, for instance, the following query expressed as an OSF term (in which the variable of interest is marked by a question mark), which intends to retrieve the instances of *person* that study in some school and that also work in a restaurant:

$$t = ?X : person \left(\begin{array}{lll} studiesAt & \to & school, \\ workAt & \to & restaurant \end{array} \right).$$

[7] A relation $R \subseteq X \times Y$ can equivalently be seen as a set-valued function from X to $\mathcal{P}(Y)$ mapping each $x \in X$ to $R(x) = \{y \in Y \mid (x, y) \in R\}$.

```
SELECT ?x WHERE {
  ?x rdf:type :person.
  ?x :studiesAt ?y.
  ?y rdf:type :school.
  ?x :workAt ?z.
  ?z rdf:type :restaurant. }
```

(a) Without the normalization step

```
SELECT ?x WHERE {
  ?x rdf:type :student.
  ?x :workAt ?y.
  ?y rdf:type :restaurant. }
```

(b) After the normalization step

Figure 3.10: The SPARQL queries of Example 3.69.

Assuming, for instance, that the input to the CEDAR reasoner contains the feature declaration $\mathit{studiesAt} : \mathit{student} \rightarrow \mathit{school}$, specifying that the feature $\mathit{studiesAt}$ only applies to object of sort $\mathit{student}$, then the query t can be normalized to $t' = ?X : \mathit{student}(\mathit{workAt} \rightarrow \mathit{restaurant})$. Fig. 3.10 shows the translation into SPARQL of the queries t and t'. Note that the instance retrieval search space of the query of Fig. 3.10b is much smaller, resulting in a more efficient retrieval. ▷

The performance of the second version of the CEDAR reasoner was compared with that of state-of-the-art Semantic Web reasoners based on DL – including FaCT++ [116], HermiT [52, 102], Pellet [105], TrOWL [115], and RacerPro [56] – with respect to the tasks of classifying a taxonomy, T-Box reasoning, and A-Box query answering. The experimental evaluation showed that the CEDAR reasoner consistently ranked among the best three reasoners for taxonomy classification. As far as T-Box reasoning is concerned, the CEDAR reasoner systematically performed orders of magnitude more efficiently compared to the other reasoners. With respect to A-Box query answering, the CEDAR reasoner using Jena[8] as a SPARQL query evaluator systematically achieved the best performances of all tested systems in term of query answering time. Moreover, the CEDAR reasoner achieved even better results by taking advantage of an indexing scheme for RDF triples based on the OSF formalism instead of Jena [15, 16].

The efficiency of the CEDAR reasoner is one of the main motivations behind the development of extensions of OSF logic that are capable of performing approximate reasoning. A fuzzy generalization of OSF logic is introduced in Chapter 4, while Chapter 5 deals with defining similarity-based reasoning within the OSF framework.

3.8 Implementation

In Section 3.7 we have reviewed two applications of OSF logic: the logic programming language LOGIN [8], and the Semantic Web reasoner CEDAR [6, 15]. Both applications are

[8] `https://jena.apache.org/`.

based on OSF term unification. Indeed, one of the advantages of LOGIN compared to other logic programming languages is that a single OSF term unification step can replace several SLD resolution steps, possibly leading to more efficient computations (e.g., Example 3.67). The query normalization phase of CEDAR is also based on the OSF constraint normalization rules, and, at the time of the experiments, it allowed this reasoner to match the performance of state-of-the-art Semantic Web reasoners with respect to concept classification, while also demonstrating greater efficiency in terms of terminological reasoning [6, 15]. The efficiency of OSF logic is due to techniques that exploit the specificity of concept taxonomies [15], in particular to *graph encoding techniques* that allow to efficiently perform lattice operations.

In this section we recall a binary encoding strategy for graphs first presented in [13] and employed in the CEDAR Semantic Web reasoner [5, 6, 16], and that is also central in the implementation of fuzzy OSF logic (Section 4.8). We start by exploring how this encoding technique can be employed in order to compute GLBs in a lattice in Section 3.8.1, and then discuss in Section 3.8.2 how the same technique can be applied to partial orders. Section 3.8.3 deals with the issue of detecting cycles in a sort subsumption declaration, while Section 3.8.4 addresses the limitations of the encoding strategy of [13] with respect to space complexity.

The idea of preprocessing the DAG representation of a lattice or a partial order to enhance the efficiency of the GLB computation was initially introduced by [13], as also acknowledged in subsequent works such as [23, 24]. Although several alternative approaches have been proposed to perform this operation ([22, 23, 24, 35, 39, 40, 47, 49, 55] just to cite a few) this section primarily delves into the original method proposed by [13], since it has seen a practical application in the implementation of the CEDAR reasoner based on OSF logic.

3.8.1 Encoding lattices

One of the core operations behind OSF term unification is the computation of the GLB of two sorts in a sort subsumption lattice $(\mathcal{S}, \preceq)$. However, traversing the graph representation of $(\mathcal{S}, \preceq)$ to perform this computation would be too expensive in terms of time complexity, potentially negating the efficiency gains offered by OSF logic, such as the ability of a single unification step to replace several SLD resolution steps.

The main intuition behind the solution of [13] is that, rather than computing GLBs in $(\mathcal{S}, \preceq)$ directly, it can be more advantageous to consider a lattice $(B, \sqsubseteq)$ where GLBs can be computed efficiently and define *encoding* and *decoding* functions $\varepsilon : \mathcal{S} \to B$ and $\delta : B \to \mathcal{S}$ such that, for all $s, s' \in \mathcal{S}$, $\delta(\varepsilon(s)) = s$ and $\varepsilon(s \wedge s') = \varepsilon(s) \sqcap \varepsilon(s')$, where $\sqcap$ is the GLB operation in $(B, \sqsubseteq)$. Consequently, GLBs in $(\mathcal{S}, \preceq)$ can be computed as $s \wedge s' = \delta(\varepsilon(s) \sqcap \varepsilon(s'))$.

Concretely, $(B, \sqsubseteq)$ is a lattice of binary strings (or bit vectors), so that the elements of $\mathcal{S}$ are encoded as strings such as `101010` or `101`. The GLB of two binary strings is efficiently

Algorithm 1 Taxonomy classification.

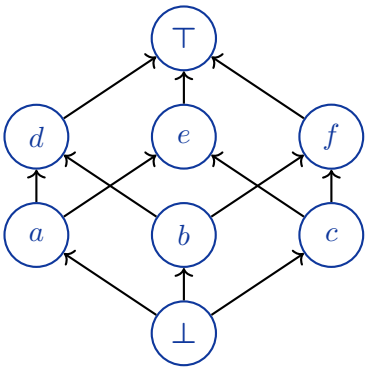
```
1:  procedure ClassifyTaxonomy
2:      ε(⊥) ← 0
3:      p ← 0
4:      L ← Parents(⊥)
5:      while L ≠ ∅ do
6:          for all s ∈ L do                                  ▷ Encode sort s
7:              ε(s) ← 2^p ⊔ ⨆_{s'∈Children(s)} ε(s')
8:              p ← p + 1
9:          L ← ⋃_{s∈L} Parents(s)                            ▷ Compute new layer
10:         for all s ∈ L do                  ▷ Remove unwanted elements from layer
11:             if ∃s' ∈ Children(x) such that ε(s') is undefined then
12:                 L ← L \ {s}
```

computed by taking their binary AND, e.g., `1011` AND `1101` = `1001`. The code $\varepsilon(s)$ of each sort $s \in \mathcal{S}$ is obtained through a bottom-up algorithm that traverses the sort subsumption lattice [13], which will be described shortly. Since the encoding of the sort lattice only needs to be performed once, while the execution of a LOGIN program, or the optimization of a CEDAR query, can involve a significant number of GLB operations, the initial cost of traversing the sort lattice is negligible. If $(\mathcal{S}, \preceq)$ is a lattice, then the result of $\varepsilon(s_0) \sqcap \varepsilon(s_1)$ is a code $b \in B$ such that $b = \varepsilon(s)$ for a unique $s \in \mathcal{S}$. Pairs of shape $(\varepsilon(s), s)$ can thus be stored in a hash table for the purpose of decoding [15].

We can now proceed with the description of the encoding algorithm. A lattice $(\mathcal{S}, \preceq)$ of sort symbols can be specified as the reflexive and transitive closure of subsumption declarations of shape $s \lhd s'$, forming a directed acyclic graph (DAG) $(\mathcal{S}, \lhd)$. In other words, $\preceq = \lhd^*$. We let $\mathit{Children}(s) \stackrel{\text{def}}{=} \{s' \in \mathcal{S} \mid s' \lhd s\}$ be the set of sorts s' that are immediately below s in $(\mathcal{S}, \lhd)$, while $\mathit{Parents}(s) \stackrel{\text{def}}{=} \{s' \in \mathcal{S} \mid s \lhd s'\}$ is the set of sorts s' that are immediately above s in $(\mathcal{S}, \lhd)$. For example, Fig. 1.2 in Section 1.1 is the graph representation of a lattice. We assume without loss of generality that the lattice $(\mathcal{S}, \preceq)$ is bounded, i.e., it has a greatest element $\top$ and a least element $\bot$ (if not, these two elements can simply be added to $\mathcal{S}$).

The classification algorithm of [13], depicted in Algorithm 1, proceeds bottom-up and assigns a bit vector to each sort as follows.

- The algorithms initializes the code of $\bot$ to `0`, a counter p to 0, and the first layer L to be encoded as $\mathit{Parents}(\bot)$ (Lines 2 to 4).
- Each sort $s \in L$ is then encoded as the binary OR (denoted $\sqcup$) of (the bit vector corresponding to the integer) 2^p and the codes of the sorts that are immediately below s (Line 7). The counter p is incremented each time a sort in L is encoded (Line 8).
- After each sort in L is encoded, a new layer is computed as the set of sorts that are



(a)

Sort	Code
$\bot$	0
a	1
b	10
c	100
d	1011
e	10101
f	100110
$\top$	1111111

(b)

Figure 3.11: A lattice and its encoding.

immediately above some sort in L. However, if a sort s has some uncoded child s' (i.e., for which $\varepsilon(s')$ is still undefined), then it is temporarily ignored. If $(\mathcal{S}, \lhd)$ is a DAG, s' will eventually be encoded, and thus also s.

Algorithm 2 Taxonomy classification by topological sort.

```
1: procedure ClassifyTaxonomyTopological
2:     T ← TopologicalSort(S, ◁)
3:     N ← |S|
4:     ε(T[0]) ← 0                                   ▷ T[0] = ⊥
5:     for all 1 ≤ i < N do                          ▷ Iterate over the rest of the sorts
6:         s ← T[i]
7:         ε(s) ← 2^i ⊔ ⨆_{s' ∈ Children(s)} ε(s')
```

In fact, if the is-a declarations form a DAG, Algorithm 1 simply proceeds by visiting sorts in topological order. A variation of Algorithm 1 based on a topological order on sorts is given in Algorithm 2. We assume that a procedure *TopologicalSort* is defined that takes a DAG $(\mathcal{S}, \lhd)$ as input and returns an array T containing the sorts of $\mathcal{S}$ in topological order. The algorithm then initializes the encoding of the bottom sort $\bot = T[0]$ and proceeds to encode all $N = |S|$ sorts in a fashion analogous to Algorithm 1.

Example 3.70 (Encoding a lattice). Fig. 3.11b shows the encoding of the sorts of the lattice represented in Fig. 3.11a, obtained by applying the classification algorithm of Algorithm 1 or Algorithm 2. The codes are computed as bit-vectors of variable length. Computing the GLB of d and f, for instance, amounts to performing the bitwise AND of their codes, i.e., $1011 \,\mathsf{AND}\, 100110 = 10$. The result is the code of the sort b, which is indeed the GLB of d and f in the lattice of Fig. 3.11a. ▷

Algorithm 2 is very simple to implement, as the Python snippet of Fig. 3.12 shows. The topological sort of the DAG representation of the sort lattice is obtained with a function

```
t = list(nx.topological_sort(sort_graph))
t[0].code = 0
code_table = {0: t[0]}
for i, sort in enumerate(t[1:]):
    code = (1 << i)
    for child in sort_graph.pred[sort]:
        code = code | child.code
    sort.code = code
    code_table[code] = sort
```

Figure 3.12: Python code for Algorithm 2.

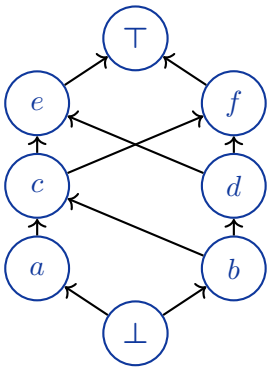
from the NetworkX package[9]. Sorts are represented as objects with a `code` attribute that stores their encoding, and are collected in a NetworkX DiGraph[10] called `sort_graph`. The `pred` attribute of this graph is a dictionary that maps each sort to the set of its children (immediate predecessors). Bit vectors are simply integers, which in Python can have an arbitrary size[11]. The integer 2^i of Line 7 of Algorithm 2 is computed using the left shift operator `<<`, while `|` is the binary OR operation. The implementation also takes care of collecting pairs $(\varepsilon(s), s)$ in a dictionary `code_table` for the purpose of decoding.

3.8.2 Encoding partial orders

Both Algorithms 1 and 2 can also be employed when $(\mathcal{S}, \preceq)$ is just a partial order rather than a lattice, but the strategy described earlier to decode $\varepsilon(s) \sqcap \varepsilon(s')$ does not longer apply if s and s' do not have a GLB. If $s \wedge s'$ exists in $(\mathcal{S}, \preceq)$, then, as stated above, $\varepsilon(s \wedge s')$ is exactly $\varepsilon(s) \sqcap \varepsilon(s')$, so that the decoding of the latter can be carried out simply by searching the sort in $\mathcal{S}$ whose encoding is equal to $\varepsilon(s) \sqcap \varepsilon(s')$ (for instance, using a hash table storing pairs of shape $(\varepsilon(s), s)$). If $s \wedge s'$ does not exist, however, this method cannot be employed.

The solution proposed by [13] and employed in the CEDAR reasoner [6] is to implicitly work in a completion of the partial order $(\mathcal{S}, \preceq)$, specifically on $(\mathit{Antichains}(X), \preceq)$ (Definition 1.15). In practice, this is achieved by computing the set of MLBs of two sorts (Definition 1.12) rather than by constructing the actual completion of $(\mathcal{S}, \preceq)$. It is clear by the inner workings of Algorithms 1 and 2 and from Example 3.70 that, for each sort s, the code $\varepsilon(s)$ corresponds exactly to the set of lower bounds of s, i.e., $\{s\}^l_\preceq = \{s' \in \mathcal{S} \mid s' \preceq s\}$ (note that each bit 1 in the code of s corresponds to a lower bound of s). Consequently, the code $\varepsilon(s) \sqcap \varepsilon(s')$ corresponds to the set $\{s, s'\}^l_\preceq = \{s'' \in \mathcal{S} \mid s'' \preceq s \text{ and } s'' \preceq s'\}$. In order to compute the set of MLBs of s and s' in $(\mathcal{S}, \preceq)$ (which corresponds to the GLB of

[9] `https://networkx.org/`.
[10] `https://networkx.org/documentation/stable/reference/classes/digraph.html`.
[11] `https://docs.python.org/3/whatsnew/3.0.html#integers`.



(a)

Sort	Code
$\bot$	0
a	1
b	10
c	111
d	1010
e	11111
f	101111
$\top$	1111111

(b)

Figure 3.13: A poset and its encoding.

$\{s\}$ and $\{s'\}$ in $(Antichains(X), \preceq))$ it is thus necessary to compute the maximal elements of this subset.

Algorithm 3 Return the MLBs of a subset of sorts.

```
1: procedure MaximalLowerBounds(s, s')
2:     L = {s, s'}^l_⪯
3:     M ← ∅
4:     for all s ∈ L do
5:         if ¬∃s' ∈ Parents(s) such that s' ∈ L then
6:             M ← M ∪ {s}
7:     return M                                  ▷ Return the MLBs of S
```

A procedure that computes the set of MLBs of two sorts s and s' is given in Algorithm 3. The algorithm first computes the set L of lower bounds of s and s' (this can be obtained, for instance, by taking each sort s'' such that the bit at the position corresponding to s'' in $\varepsilon(s) \sqcap \varepsilon(s')$ is equal to 1), and then collects in a set M the elements that are maximal in L. This algorithm is not particularly efficient, but it only needs to be applied when the two sorts s and s' do not have a GLBs, i.e., when $\varepsilon(s) \sqcap \varepsilon(s')$ is not stored in the hash table. Moreover, the decoding is only necessary for extracting the end result of a query, while all intermediate steps need not refer to the sorts and deal only with bit vectors [6]. After the computation is done, the pair $(\varepsilon(s) \sqcap \varepsilon(s'), \{s, s'\}^{mlb}_{\preceq})$ may be stored in a hash table for faster retrieval in future queries.

Example 3.71 (Encoding a partial order). Fig. 3.13b shows the encoding of the sorts of the poset represented in Fig. 3.13a, obtained by applying the classification algorithm of Algorithm 1 or Algorithm 2. The codes are computed as bit-vectors of variable length. Computing the GLB of c and d, for instance, amounts to performing the bitwise AND of their codes, i.e., $111 \,\mathsf{AND}\, 1010 = 10$. The result is the code of the sort b, which is indeed the GLB of c and d in the poset of Fig. 3.13a. The bitwise AND of the codes of the sorts e and

f, however, results in 11111 AND 101111 = 1111, which does not correspond to any of the sorts of the poset of Fig. 3.13a. The set of MLBs of e and f is then computed by applying the procedure MaximalLowerBounds of Algorithm 3. The procedure first computes the set $L = \{\bot, a, b, c, d, \}$ and then proceeds to find the set of maximal elements of L, i.e., $M = \{c, d\}$, which is the set of MLBs of e and f. ▷

Working with partial orders rather than lattices involves extending the syntax of OSF logic to support disjunctive terms [10], i.e., terms of shape $X : \{t_1, \ldots, t_n\}$, where each t_i is a (possibly disjunctive) OSF term. Given an interpretation $\mathcal{I}$ and an assignment α, the semantics of a disjunctive term is

$$[\![X : \{t_1, \ldots, t_n\}]\!]^{\mathcal{I},\alpha} = \{\alpha(X)\} \cap \bigcup_{i=1}^{n} [\![t_i]\!]^{\mathcal{I},\alpha}.$$

More details about OSF logic with disjunctive sorts can be found in [4, 10].

3.8.3 Dealing with cycles

Another issue arises when the considered sort hierarchy $(\mathcal{S}, \preceq)$ is neither a lattice nor a poset, but rather a preorder. This is the case whenever the DAG representation $(\mathcal{S}, \lhd)$ of $(\mathcal{S}, \preceq)$ contains cycles, indicating a violation of antisymmetry in $\preceq$.

While Algorithm 2 cannot be applied in this case, as it requires a topological sort of $(\mathcal{S}, \lhd)$, it is still possible to run Algorithm 1 on a DAG containing cycles [6]. The result is that, at the end of the procedure, the sorts in $\mathcal{S}$ that are involved in a cycle are not encoded. The approach adopted by the CEDAR reasoner is to identify the maximal cycles contained in $(\mathcal{S}, \lhd)$ and to report them as errors to the user in order to be corrected [5, 6]. Another approach, also mentioned in [5, 6], is to collapse the maximal cycles into a single sort, thus constructing a partial order according to Proposition 1.20.

3.8.4 Space considerations

While the encoding strategy of Sections 3.8.1 and 3.8.2 provides a very efficient way to compute GLBs and MLBs, a quadratic amount of space is required to store the encoding of a sort lattice (or partial order), which could be prohibitive for very large ontologies or knowledge bases. More precisely, if fixed-length sequences of bits are employed, each of the N sorts is assigned a bit vector of length N, requiring N^2 bits of space.

We have run a few experiments in Python to get an idea of the space required in practice to encode a partial order following the strategy of Sections 3.8.1 and 3.8.2. The implementation of the encoding algorithm is analogous to Fig. 3.12, which employs Python integers, comparable to bit vectors of variable length, to encode sorts. As a consequence, ignoring the

overhead of Python objects[12], for each $0 < i < |\mathcal{S}|$, the sort at position i in the topological order of the DAG of subsort declarations is encoded with a bit vector of length i, while the bit vector of $\bot$, which is at position 0 in the topological ordering, is equal to 0 and thus requires 1 bit to encode. The space necessary to encode the whole poset $(\mathcal{S}, \preceq)$ is thus equal to

$$1 + \sum_{0 \leq i < N} i = 1 + \frac{N \cdot (N-1)}{2} \tag{3.1}$$

bits, where $N = |\mathcal{S}|$, saving a significant amount of space with respect to the N^2 bits required by fixed-length sequences of bits. Table 3.1 shows the time (in seconds) needed to encode random DAGs of various sizes, and the space required to store the encoding (comparing the theoretical size that ignores the overhead of Python objects, and the space required in practice)[13]. As another example, the WordNet DAG is made up of 74374 nodes, so that by Eq. (3.1) its encoding size can be estimated at around 329 MB. In practice, the actual encoding takes up around 360 MB.

The space limitation of this strategy was already addressed by its authors, who proposed an alternative approach, called *modulation*, which relies on the observation that DAGs in many applications consist of several densely connected groups of nodes, with only a few links into other dense groups [13]. The worst-case space complexity of this approach improves to $O(|\mathcal{S}| \log(|\mathcal{S}|))$, but the complexity of the GLB operations increases to $O(\log(|\mathcal{S}|)$, rather than the virtually constant-time operation of performing a bitwise AND. Another encoding method, called *compact codes*, was proposed in the context of the CEDAR reasoner [6]. This method – which is again more efficient in terms of space, but more expensive with respect to time complexity – was only used to serialize a sort lattice to disk [6] (although it can also support lattice operations). Despite its space limitation, the encoding strategy of Section 3.8.1 was successfully employed by CEDAR to perform lattice operations [6, 15], encoding sort taxonomies consisting of up to 903617 sorts.

We conclude this section by considering another approach for efficient GLB computation (mentioned, for example, in [66]) that simply consists in storing GLBs of pairs of sorts in a hash table.

While storing the GLB of *every* pair of sorts in a table is clearly not space efficient, optimizations can be carried out by considering that the subsumption hierarchies used in knowledge representation are often very sparse, and resemble trees with a few back edges. For instance, the WordNet DAG has an average outdegree of only ~ 1.02. It may thus happen that there are very few non-bottom GLBs in these hierarchies (i.e., GLBs different

[12] Integers in Python are objects, and their size is at least 24 bytes.

[13] All experiments in this sections were run with implementations in Python 3.9.5 of the relevant algorithms on the WordNet dag (as provided by nltk) or on random DAGs (generated with NetworkX) with a least element and a greatest element and an average outdegree of 1.5. The experiments were run on a Dell XPS 13 running Ubuntu 20.04, with an Intel Core i7-8565U CPU @ 1.80GHz × 8, and 15.3 GiB RAM. The nltk library can be found at `https://www.nltk.org/`, and the NetworkX package can be found at `https://networkx.org/`.

	Time (s)	Encoding Size (MB)	
$\lvert\mathcal{S}\rvert$	Exp.	Theor.	Exp.
939	0.009	0.052	0.139
9504	0.119	5.383	6.512
47476	1.101	134.344	148.242
94943	2.768	537.281	582.983
142446	5.028	1209.421	1302.376
190016	7.830	2152.079	2315.324
237592	12.243	3364.666	3611.197
285198	17.425	4848.100	5195.975
500000	//	14901.131	//
750000	//	33527.568	//
1000000	//	59604.585	//

Table 3.1: Time and space required to encode a DAG of different sizes. The theoretical encoding size is computed according to Eq. (3.1).

from the bottom element $\bot$). For instance, it turns out that only around 0.022% of all possible pairs (not considering order) of nodes in the WordNet DAG have a non-bottom GLB or set of MLBs, i.e., 600992 out of $\frac{N\times(N-1)}{2} = 2765708751$ pairs, where $N = 74374$ is the number of nodes of WordNet. Using Python, for instance, it is possible to store these GLBs in around 38 MB, compared to the 360 MB required by the variable-length bit vectors discussed above. This is achieved as follows.

- Let $\mathcal{S}$ be the set of nodes of WordNet. We assume that, for each node $s_i \in \mathcal{S}$, i is the index of the sort s_i in a fixed topological ordering of $\mathcal{S}$ such that, if $i \leq j$, then s_i comes before s_j in this ordering.
- If $s_i \wedge s_j \in \{s_i, s_j\}$, with $i \leq j$, we store the hash of the pair[14] (i, j) in a set S.
- If $s_i \wedge s_j = s_k \in \mathcal{S} \setminus \{\bot, s_i, s_j\}$, with $i \leq j$, then we store the pair $(\texttt{hash}(i, j), k)$ in a Python dictionary (a hash table) D, where `hash` is the Python built-in function returning the hash of an object.
- If $s_i \wedge s_j$ does not exist, with $i \leq j$, then we store the pair $(\texttt{hash}(i, j), \{k_0, \ldots, k_n\})$ in D, where $\{s_i, s_j\}^{mlb}_{\preceq} = \{s_{k_0}, \ldots, s_{k_n}\}$.

The GLB or set of MLBs of two sorts $s_i, s_j \in \mathcal{S}$ $(i \leq j)$ can then be computed as follows: (i) if $i = j$, return s_i; (ii) else, if $s_j = \top$, return s_i; (iii) else, if $\texttt{hash}(i, j) \in S$, return s_i; (iv) else, if $\texttt{hash}(i, j)$ is a key of the dictionary D, return its value; (v) else, return $\bot$.

All of these steps can be performed in constant time, so that the whole computation is $O(1)$. In fact, GLB computation is faster than using bit vectors. This is due to the fact that,

[14] Recall that in Python tuples of hashable objects are hashable.

N	Time (μs)
100	0.65
1000	0.73
10000	0.37
100000	1.48
500000	6.65
1000000	13.20
5000000	72.60
10000000	146.00

Table 3.2: Time needed to compute the bitwise AND on two bit vectors of length N.

for large values of N, a binary operation on bit-vectors of length N is not a constant-time operation. Table 3.2 shows the average time for the computation of a binary AND on two bit-vectors of length N. Moreover, decoding the bit vector obtained as the bitwise AND of two bit vectors may involve computing the set of MLBs, which takes linear time in the number of nodes and edges of the DAG (Algorithm 3)

With respect to WordNet, computing the GLBs or MLBs of 100 randomly chosen pairs of sorts takes 123 μs using bit-vectors, and 49.7 μs using the method outlines above, including the time to recover the sorts from their integer representations.

While this method can achieve a smaller space complexity and slightly faster GLB computations, it may only be advantageous for very sparse DAGs such as WordNet, where only 0.02% of the GLBs need to be stored. Another drawback of this approach is that all GLBs need to be computed beforehand as an offline preprocessing step. While a naive approach would compute the GLBs or MLBs for each pair (s_i, s_j) of sorts (with $i \leq j$), an optimization can be implemented by considering that, if $s_0 \wedge s_1 = \bot$, then for all $s'_0 \preceq s_0$ and $s'_1 \preceq s_1$ it must be the case that $s'_0 \wedge s'_1 = \bot$, so that there is no need to compute the latter GLB. This optimization – which is mentioned, for instance, in [66], where no algorithm is however given – is implemented in Algorithm 4, which works as follows.

1. A hash table *GlbTable* that maps a pair of sorts to either a sort (their GLB) or a set of sorts (the set of their MLBs) is initialized on Line 1, while a topological sort of the subsumption DAG $(\mathcal{S}, \lhd)$ is computed on Line 2. In the rest of the algorithm, the subscript i of a sort s_i indicates its position in this topological ordering.

2. On Line 4 starts a loop that considers each sort s_i in topological order except for the bottom and top elements $T[0] = \bot$ and $T[N-1] = \top$, since computing the GLB of a sort s with either of these elements can be done in constant time without any encoding.

3. For each sort s_i, the algorithm performs a non-recursive depth-first search (DFS) starting from the top element $\top$. We keep track of already visited nodes in the set

Algorithm 4 Precompute all GLBs and MLBs.

```
1:  GlbTable ← new HashTable⟨S × S, S ∪ P(S)⟩();
2:  T ← TopologicalSort(S, ◁);
3:  N ← |T| ;
4:  for all 1 ≤ i < N − 1 do                    ▷ Iterate over the rest of the sorts
5:      s_i ← T[i];
6:      Seen ← ∅;
7:      NodesStack ← new Stack⟨S⟩();             ▷ Create empty stack of nodes;
8:      for all s ∈ Children(⊤) do
9:          NodesStack.push(s);
10:     while NodesStack is not empty do
11:         s_j ← NodesStack.pop();
12:         if j < i or s_j ∈ Seen then
13:             continue;
14:         Seen ← Seen ∪ {s_j};
15:         if GLB({s_i, s_j}) = ⊥ then
16:             continue;
17:         GlbTable.put((s_i, s_j), GLB({s_i, s_j}));
18:         for all s ∈ Children(s_j) do
19:             NodesStack.push(s)
```

Seen initialized on Line 6, and create a stack *NodesStack* of sorts for the DFS on Line 7, initialized with the children of $\top$ (Lines 8 and 9).

4. In depth-first fashion, a node s_j is popped from the stack until *NodesStack* is empty i.e., s_j is visited (Lines 10 and 11). The node s_j is skipped if it has already been seen, or if its topological rank is lower than that of s_i (Lines 12 and 13) since this means that the pair (s_j, s_i) has already been considered at a previous iteration of the loop starting at Line 4. Otherwise, on Line 14 the node s_j is added to the set of already visited nodes, so that it will not be considered again.

5. The GLB or MLBs of s_i and s_j is then computed. If this is the bottom element, nothing else is done, and the computation resumes on Line 10, visiting a new node, if available. Otherwise, $s_i \wedge s_j$ is stored in the hash table *GlbTable* (Line 17), and the children of s_j are pushed to the stack (Lines 18 and 19).

   Note that if $s_i \wedge s_j = \bot$, then the children of s_j are *not* pushed to the stack, since their GLB (and the GLB of their children, and so on) with respect to s_i would also be $\bot$, implementing the aforementioned optimization.

The procedure GLB at Lines 15 and 17 may be any algorithm for computing GLBs and MLBs, including the ones of Section 3.8.1 or Section 3.8.2. In this case, encoding the DAG would be necessary for this offline step, but the bit-vectors could be discarded once the hash

table has been obtained. Following this approach, the runtime of this algorithm is around 46 seconds for WordNet, compared to the several hours needed by a naive approach that computes the GLB/MLBs for all pairs of sorts (even taking commutativity into account). In general, the actual efficiency gains of using this algorithm compared with the naive one depend on how sparse the considered DAG is.

Chapter 4

Fuzzy Order-Sorted Feature Logic

In Chapter 3 we have explored Order-Sorted Feature (OSF) logic, a Knowledge Representation and Reasoning language based on functional features and partially ordered sorts. The terms of this language are record-like structures that denote partially ordered classes of objects, and that generalize first-order terms by allowing to represent partial information. The OSF term unification algorithm, together with implementation techniques that leverage the specificity of concept subsumption relations, enable an efficient calculus of type subsumption, which has been implemented in logic programming languages such as LOGIN [8] and automated reasoners such as CEDAR [6, 15].

This chapter introduces a fuzzy generalization of OSF logic – called *fuzzy OSF logic* – based on the Gödel t-norm (minimum) and t-conorm (maximum). This language incorporates the following key features.

1. The syntax of OSF logic is essentially unaltered, allowing, for instance, to benefit from the crisp results regarding the equivalence of normal OSF terms, rooted solved OSF clauses and OSF graphs.

2. The sort symbols of OSF logic are interpreted as *fuzzy subsets* of the domain of an interpretation rather than crisp subsets, enabling the support of real-valued degrees of membership of an object to a particular sort. This interpretation is extended to OSF terms, allowing to represent complex classes of object with a fuzzy denotation.

3. The sort subsumption relation is generalized to a *fuzzy subsumption relation between sorts*, which allows to model a weaker notion of inclusion. This allows scenarios where an object's membership degree to a sort s might be greater than its membership degree to a supersort of s. This notion is extended to OSF terms, so that every two terms are associated with a subsumption degree. OSF terms are proved to be ordered in a fuzzy lattice, generalizing the OSF term lattice of crisp OSF logic.

4. The greatest lower bound (GLB) of two OSF terms in the fuzzy subsumption lattice

can be computed through their *unification*, by applying constraint normalization rules that are essentially the same as those of crisp OSF logic. This constitutes a significant advantage with respect to the implementation of fuzzy OSF logic, as it allows to leverage the same efficient techniques of crisp OSF logic. Moreover, the computational complexity of finding the GLB of two terms in the fuzzy subsumption lattice is the same as in crisp OSF logic.

5. Differently from the crisp setting, the unifier of two OSF terms is associated with a *unification degree*, the computation of which can be optimized by exploiting the same encoding techniques of crisp OSF logic.

The idea of a fuzzy generalization of OSF logic was first introduced in [9], where a weaker notion of first-order term unification based on a similarity relation between term constructors is defined that also allows mismatches between functor arities and argument positions (see Section 2.2.1). While our generalization of the semantics of OSF logic to a fuzzy setting is novel, fuzzy OSF logic shares a few common aspects with fuzzy Description Logics (DLs) (e.g., [18, 30, 77, 113]), similarly to their crisp counterparts. For instance, fuzzy DL concepts are also interpreted as fuzzy subsets of a domain. On the other hand, fuzzy DLs interpret roles as fuzzy relations, while fuzzy OSF logic maintains a crisp interpretation for feature symbols, as will be seen in Section 4.3. Moreover, as will be clear shortly, the notion of fuzzy subsumption of sorts and OSF terms in fuzzy OSF logic differs significantly with that of fuzzy concept inclusions in fuzzy DLs, which relies on the fuzzy implication operator (see Section 2.1.2 and Example 2.5).

Before delving into the formal development of fuzzy OSF logic, Section 4.1 provides an informal presentation of our definition of fuzzy sort subsumption and how it is extended to the terms of OSF logic

The syntax of fuzzy OSF logic is essentially unchanged from that of its crisp version. The only difference, which is discussed in Section 4.2, concerns OSF signatures, which now feature a fuzzy subsumption ordering $\preceq$ on the set of sort symbols.

The fuzzy generalization of the semantics of OSF logic begins in Section 4.3, where we define the interpretation of the syntactic objects of this language – namely sorts, features, OSF terms and OSF clauses – in structures called fuzzy OSF interpretations. The fuzzy OSF graph algebra, a special interpretation whose elements are OSF graphs, is defined in Section 4.4.

In Section 4.5 we define structure-preserving mappings between fuzzy OSF interpretations called fuzzy OSF homomorphisms, which are valuable for proving several results regarding the satisfiability of OSF clauses in fuzzy interpretations.

Fuzzy OSF homomorphisms are then employed extensively in Section 4.6, where we extend the fuzzy sort subsumption ordering to OSF terms and prove that it constitutes a

fuzzy partial order. We provide similar orderings for OSF clauses and OSF graphs, and illustrate how the fuzzy OSF term subsumption ordering is related to its crisp counterpart.

Section 4.7 is devoted to unification. We show how to compute the GLB of two OSF terms in the fuzzy OSF term subsumption ordering through their unification, and how to find the degree to which two OSF terms are subsumed by each other. We also discuss the complexity of these computations. Implementation issues are then addressed in Section 4.8.

Finally, in Section 4.9 we discuss the potential application of fuzzy OSF logic as a fuzzy logic programming language based on OSF terms, which leverages a fuzzy subsumption relation between sort symbols in order to provide approximate answers to queries posed to a knowledge base.

The proofs of the main results of this chapter are reported in Appendix A.1.

4.1 Fuzzy OSF logic, informally

Let $\mathcal{S}$ and $\mathcal{F}$ be (finite) sets of *sort symbols* and *feature symbols*, respectively, and let us fix an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, i.e., a structure with a domain $\Delta^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that provides an interpretation to the elements of $\mathcal{S}$ and $\mathcal{F}$[1]. In fuzzy OSF logic, a sort $s \in \mathcal{S}$ is interpreted as a fuzzy subset $s^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0,1]$, while a feature symbol $f \in \mathcal{F}$ is interpreted as a function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}$. We rely on the Gödel t-norm $\wedge$ and t-conorm $\vee$.

As seen in Chapter 3, in crisp OSF logic a subsumption relation $\preceq \subseteq \mathcal{S} \times \mathcal{S}$ is a binary relation that denotes set inclusion. A natural fuzzy generalization of this notion is Zadeh's inclusion of fuzzy sets [44], according to which the subsumption $s \preceq s'$ has the following meaning:

$$\text{if } s \preceq s', \text{then } \forall d \in \Delta^{\mathcal{I}} : s^{\mathcal{I}}(d) \leq s'^{\mathcal{I}}(d). \tag{4.1}$$

That is, this definition of fuzzy inclusion requires that, whenever d is an instance of $s^{\mathcal{I}}$ with degree $s^{\mathcal{I}}(d) = \beta \in [0,1]$, then d must also be an instance of $s'^{\mathcal{I}}$ with a degree *greater than or equal to* β.

We consider a *fuzzy subsumption relation* as a way to model a weaker notion of inclusion, where the element d can be an instance of $s'^{\mathcal{I}}$ with a degree that *may possibly be smaller than* its degree of membership to $s^{\mathcal{I}}$. Thus, we define a fuzzy subsumption relation as a fuzzy partial order, i.e., a function $\preceq : \mathcal{S}^2 \to [0,1]$ that associates a subsumption degree β with each pair of sort symbols (s_0, s_1) (and satisfying every constraint of Definition 1.28) and having the following semantics:

$$\text{if } \preceq(s_0, s_1) = \beta, \text{ then } \forall d \in \Delta^{\mathcal{I}} : s_0^{\mathcal{I}}(d) \wedge \beta \leq s_1^{\mathcal{I}}(d). \tag{4.2}$$

[1] Fuzzy interpretations will be defined properly in Section 4.3.

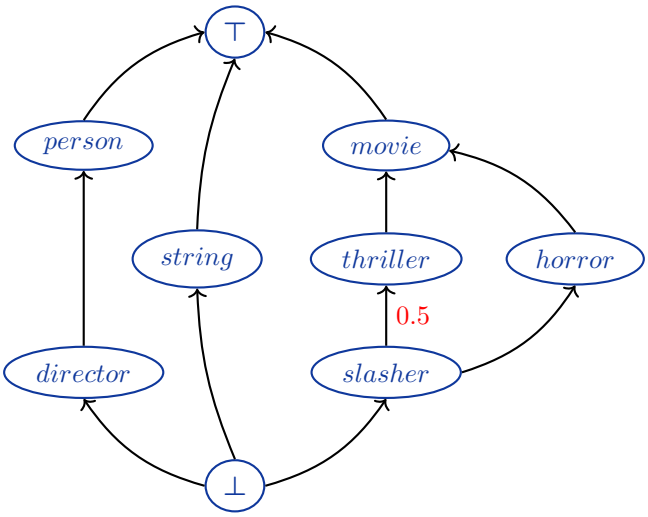


Figure 4.1: A fuzzy subsumption relation.

That is, any object d which is an instance of $s_0^{\mathcal{I}}$ with degree β_0 must also be an instance of $s_1^{\mathcal{I}}$ with a degree β_1 that is *greater than or equal to the minimum of* β_0 and β. Note that (4.1) is a special case of this equation with $\beta = 1$. An example of a fuzzy sort subsumption relation is given in Fig. 4.1[2].

Sort symbols are analogous to the atomic concepts of DLs. As illustrated in Chapter 3, complex concepts can be expressed with *OSF terms*, structures built from sort symbols, *feature symbols* (attributes) and variables. For example, the term

$$t_1 = X_1 : \mathit{movie} \left(\begin{array}{ll} \mathit{directed\_by} & \rightarrow Y_1 : \mathit{person}, \\ \mathit{genre} & \rightarrow Z_1 : \mathit{thriller} \end{array} \right)$$

denotes the class of movies directed by some person and whose genre is thriller.

In this chapter we show that a fuzzy subsumption relation between sort symbols can be extended to a fuzzy subsumption relation between OSF terms. For example, consider the fuzzy subsumption relation depicted in Fig. 4.1 and the term

$$t_2 = X_2 : \mathit{movie} \left(\begin{array}{ll} \mathit{title} & \rightarrow W_2 : \mathit{string}, \\ \mathit{genre} & \rightarrow Z_2 : \mathit{slasher}, \\ \mathit{directed\_by} & \rightarrow Y_2 : \mathit{director} \end{array} \right).$$

This term is more specific that t_1, as it provides additional information by introducing the feature title and by constraining its value to be of sort *string*, and because it defines more restrictive constraints, by requiring that the value of the feature *directed_by* be of sort *director*, which is subsumed by *person*, and that the value of the feature *genre* be of sort

[2]We represent a fuzzy subsumption relation $\preceq$ graphically as a weighted directed acyclic graph (DAG), of which $\preceq$ is the reflexive and transitive closure (see Definition 1.30 in Section 1.2).

slasher, which is subsumed by *thriller* with degree 0.5. In this case we say that t_1 *subsumes* t_2 with degree 0.5.

As seen in Chapter 3, one of the reasoning tasks supported by OSF logic is deciding whether a given term is subsumed by another, or in general finding the most general term which is subsumed by two given terms. This is also the case for fuzzy OSF logic, where, similarly to the crisp case, this problem can be solved by unifying two OSF terms, a process which aims to combine the constraints expressed by the two terms in a consistent way. For example, t_2 is the unifier of term t_1 and the following term:

$$t_3 = X_3 : \mathit{movie} \left(\begin{array}{lcl} \mathit{directed\_by} & \to & Y_3 : \mathit{director}, \\ \mathit{title} & \to & W_3 : \mathit{string}, \\ \mathit{genre} & \to & Z_3 : \mathit{horror} \end{array} \right).$$

In particular, the value for the feature *genre* in t_2 must be of sort *slasher*, as this sort is subsumed by both *thriller* and *horror* (the values of *genre* in t_1 and t_3), and it is the most general one with this property. Additionally, in fuzzy OSF logic the unifier t_2 is associated with a *unification degree*, which depends on the subsumption degrees of its sorts with respect to the corresponding sorts in t_1 and t_3. In this case the unification degree is 0.5, due to $\preceq(\mathit{slasher}, \mathit{thriller}) = 0.5$.

4.2 Syntax

Fuzzy OSF logic is based on the same syntactic structures of (crisp) OSF logic, so that the definitions of (normal) OSF terms and (rooted solved) OSF clauses are unchanged. As far as syntax is concerned, the only difference with crisp OSF logic is that the sort symbols are ordered in a fuzzy sort subsumption relation rather than a crisp one. The most significant differences concern the semantics, as will be seen starting from Section 4.3.

Definition 4.1 (Fuzzy OSF signature). A *fuzzy OSF signature* is a tuple $(\mathcal{S}, \mathcal{F}, \preceq)$ where $\mathcal{S}$ is a set of *sort symbols*, $\mathcal{F}$ is a set of *feature symbols*, and $(\mathcal{S}, \preceq)$ is a finite bounded fuzzy lattice with least element $\bot$ and greatest element $\top$[3]. The elements of $\mathcal{S}$ and $\mathcal{F}$ will also simply be called *sorts* and *features*, respectively. The greatest lower bound (GLB) $s \curlywedge s'$ of two sorts s and s' is also called their *greatest common subsort*.

Example 4.2 (Fuzzy OSF signature). As an example of a fuzzy OSF signature we may take the set of sorts and the fuzzy subsumption relation corresponding to the graph of Fig. 4.1, and $\mathcal{F} = \{\mathit{directed\_by}, \mathit{title}\}$ as the set of features. ▷

Since the sort symbols are now ordered in a fuzzy subsumption, the normalization of an OSF clause is performed over a fuzzy lattice. In fact, the constraint normalization rules of

[3] See Definition 1.40 in Section 1.2.

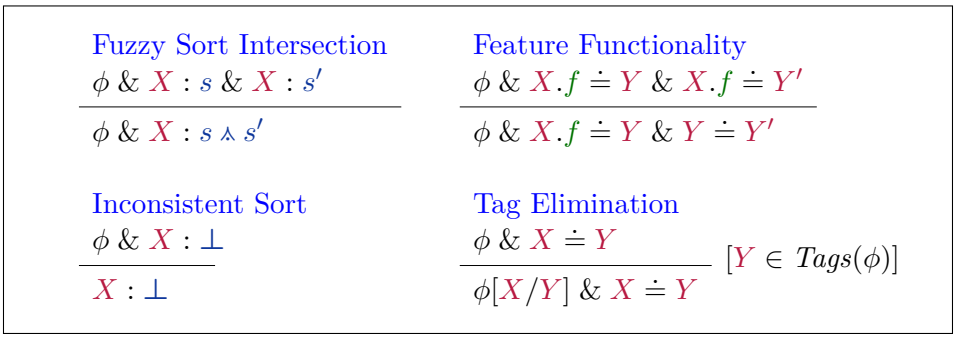


Figure 4.2: Fuzzy OSF constraint normalization rules.

fuzzy OSF logic (Fig. 4.2) are the same as the ones of crisp OSF logic (Fig. 3.2), except that the rule Fuzzy Sort Intersection involves the GLB operation $\curlywedge$ in a fuzzy lattice rather than a crisp one. Thanks to Proposition 1.42, this does not constitute a significant difference as far as the properties of the normalization procedure are concerned.

Proposition 4.3 (Fuzzy OSF clause normalization). *The rules of Fig. 4.2 are finite terminating and confluent (modulo variable renaming). Furthermore, they always result in a normal form that is either the inconsistent clause or an OSF clause in solved form together with a conjunction of equality constraints.*

Exactly like in the crisp setting, an OSF term ψ can be normalized by applying the constraint normalization rules to $\phi(\psi)$ and translating the result back into an OSF term.

4.3 Semantics

In this section we generalize the semantics of OSF logic [10] to a fuzzy setting. We begin by defining *fuzzy OSF interpretations*, the structures in which the syntactic objects of OSF logic are interpreted. Fuzzy OSF interpretation, or fuzzy OSF algebras, provide the meaning of a fuzzy sort subsumption relation and determine the denotations of sort symbols as fuzzy subsets and of feature symbols as functions.

Let us fix a fuzzy signature $(\mathcal{S}, \mathcal{F}, \preceq)$ for the rest of the section.

Definition 4.4 (Fuzzy OSF interpretation). A *(fuzzy) OSF interpretation* (or *algebra*) for a signature $(\mathcal{S}, \mathcal{F}, \preceq)$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that[4]

1. $\Delta^{\mathcal{I}}$ is a non-empty set, called the *domain* or *universe* of the interpretation;
2. for each $s \in \mathcal{S}$, $s^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0,1]$ is a fuzzy subset of $\Delta^{\mathcal{I}}$ (where in particular $\top^{\mathcal{I}} = \mathbf{1}_{\Delta^{\mathcal{I}}}$ and $\bot^{\mathcal{I}} = \mathbf{1}_{\emptyset}$);

[4]The symbol $\mathbf{1}_D$ denotes the characteristic function $\mathbf{1}_D : \Delta^{\mathcal{I}} \to \{0,1\}$ of the set $D \subseteq \Delta^{\mathcal{I}}$, which is defined by letting, for all $d \in \Delta^{\mathcal{I}}$, $\mathbf{1}_D(d) \stackrel{\text{def}}{=} 1$ if $d \in D$, and $\mathbf{1}_D(d) \stackrel{\text{def}}{=} 0$ otherwise. The symbol $\cap$ denotes the intersection of fuzzy subsets (Definition 1.23).

3. for each $s_0, s_1 \in \mathcal{S}$, $\forall d \in \Delta^{\mathcal{I}} : s_0^{\mathcal{I}}(d) \wedge \preceq(s_0, s_1) \leq s_1^{\mathcal{I}}(d)$;
4. for each $s_0, s_1 \in \mathcal{S}, \forall d \in \Delta^{\mathcal{I}}$: if $(s_0^{\mathcal{I}} \sqcap s_1^{\mathcal{I}})(d) > 0$, then $(s_0 \curlywedge s_1)^{\mathcal{I}}(d) > 0$; and
5. for each $f \in \mathcal{F}$: $f^{\mathcal{I}}$ is a function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}$.

In the following we write $d \in \mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$ interchangeably. Note that the same symbol $\mathcal{I}$ is used for crisp OSF interpretations and for fuzzy OSF interpretations, since its meaning is always clear from context.

As discussed in Section 4.1, the motivation behind condition (3) is to model a weaker notion of inclusion which allows scenarios where an object's membership degree to a sort s might be greater than its membership degree to a supersort of s.

In order to motivate condition (4), let us momentarily switch to the crisp setting of Chapter 3, where $s^{\mathcal{I}}$ denotes a (crisp) subset of the domain $\Delta^{\mathcal{I}}$, $\preceq \subseteq \mathcal{S}^2$ is a binary subsumption relation and $\wedge$ is the GLB operation on $(\mathcal{S}, \preceq)$. The definition of an OSF algebra (Definition 3.17) requires that the following holds for every $s_0, s_1 \in \mathcal{S}$:

$$s_0^{\mathcal{I}} \cap s_1^{\mathcal{I}} \subseteq (s_0 \wedge s_1)^{\mathcal{I}}. \tag{4.3}$$

In other words, whenever an element belongs to the intersection of the denotations of two sorts, then it must also belong to the denotation of their GLB. Our fuzzy generalization of this requirement is very flexible as it does not impose restrictive constraints on the membership degree of d to $(s_0 \curlywedge s_1)^{\mathcal{I}}$, but it simply requires that, whenever an object is an instance of two sorts s_0 and s_1 with a degree greater than 0, then it must also be an instance of their GLB with a degree greater than 0. A direct generalization of (4.3), stating that $s_0^{\mathcal{I}} \sqcap s_1^{\mathcal{I}}(d) \leq (s_0 \curlywedge s_1)^{\mathcal{I}}(d)$ for all $d \in \Delta^{\mathcal{I}}$ and $s_0, s_1 \in \mathcal{S}$, would be too restrictive. Indeed, this formulation would require that the degree of membership of an object d to the sort $s_0 \curlywedge s_1$ is always greater than or equal to its degree of membership to either s_0 or s_1. There may however be scenarios where it is convenient for $(s_0 \curlywedge s_1)^{\mathcal{I}}(d)$ to be smaller than both ${s_0}^{\mathcal{I}}(d)$ and ${s_1}^{\mathcal{I}}(d)$. For instance, the following example models a situation where $\mathit{thriller}^{\mathcal{I}}(\mathit{psycho}) = \mathit{horror}^{\mathcal{I}}(\mathit{psycho}) = 1$, but $(\mathit{thriller} \curlywedge \mathit{horror})^{\mathcal{I}}(\mathit{psycho}) = \mathit{slasher}^{\mathcal{I}}(\mathit{psycho}) = 0.7$, which is allowed by the more flexible formulation of Definition 4.4.

Example 4.5 (Fuzzy OSF interpretation). Consider the fuzzy OSF signature of Example 4.2 and let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be defined as follows.

- The domain is

$$\Delta^{\mathcal{I}} = \{\mathit{psycho}, \mathit{halloween}, \mathit{hitchcock}, \mathit{carpenter}, \text{“}\mathit{Psycho}\text{”}, \text{“}\mathit{Halloween}\text{”}, \mathit{null}\}.$$

- The interpretation of the sort symbols is defined by letting

- $\mathit{thriller}^{\mathcal{I}}(\mathit{halloween}) = 0.5$;
- $\mathit{horror}^{\mathcal{I}}(\mathit{halloween}) = \mathit{slasher}^{\mathcal{I}}(\mathit{halloween}) = 1$;
- $\mathit{thriller}^{\mathcal{I}}(\mathit{psycho}) = \mathit{horror}^{\mathcal{I}}(\mathit{psycho}) = 1$ and $\mathit{slasher}^{\mathcal{I}}(\mathit{psycho}) = 0.7$;
- $\mathit{movie}^{\mathcal{I}}(\mathit{psycho}) = \mathit{movie}^{\mathcal{I}}(\mathit{halloween}) = 1$;
- $\mathit{string}^{\mathcal{I}}(\text{“}\mathit{Psycho}\text{”}) = \mathit{string}^{\mathcal{I}}(\text{“}\mathit{Halloween}\text{”}) = 1$;
- $\mathit{person}^{\mathcal{I}}(\mathit{hitchcock}) = \mathit{person}^{\mathcal{I}}(\mathit{carpenter}) = 1$;
- $\mathit{director}^{\mathcal{I}}(\mathit{hitchcock}) = \mathit{director}^{\mathcal{I}}(\mathit{carpenter}) = 1$;
- $\top^{\mathcal{I}}(x) = 1$ for every $x \in \Delta^{\mathcal{I}}$; and
- all the remaining membership degrees are equal to 0.

- The interpretation of the feature symbols is defined by letting
  - $\mathit{directed\_by}^{\mathcal{I}}(\mathit{psycho}) = \mathit{hitchcock}$;
  - $\mathit{directed\_by}^{\mathcal{I}}(\mathit{halloween}) = \mathit{carpenter}$;
  - $\mathit{title}^{\mathcal{I}}(\mathit{psycho}) = \text{“}\mathit{Psycho}\text{”}$ and $\mathit{title}^{\mathcal{I}}(\mathit{halloween}) = \text{“}\mathit{Halloween}\text{”}$; and
  - all the remaining feature applications are equal to *null*.

This is easily verified to satisfy all constraints of Definition 4.4. In particular

$$\mathit{thriller}^{\mathcal{I}}(\mathit{halloween}) = 0.5 \geq 1 \wedge 0.5 = \mathit{slasher}^{\mathcal{I}}(\mathit{halloween}) \wedge \preceq(\mathit{slasher}, \mathit{thriller}). \qquad \triangleright$$

Similarly to crisp OSF logic, since features are interpreted as *total* functions, in the last example the feature *title* had to be defined also for elements of sort *person* such as *hitchcock*. An extension of fuzzy OSF logic that interprets features as partial functions is left for future work.

We now define the meaning of an OSF term in a fuzzy OSF interpretation. Let $\mathit{Val}(\mathcal{I})$ be the set of all variable assignments $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$.

Definition 4.6 (Fuzzy denotation of an OSF term). Let $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$ be an OSF term, let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a fuzzy OSF interpretation, and let $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ be a variable assignment. *The denotation of t in the interpretation $\mathcal{I}$ under the assignment α* is the fuzzy subset of $\Delta^{\mathcal{I}}$ defined by letting, for all $d \in \Delta^{\mathcal{I}}$:

$$[\![t]\!]^{\mathcal{I},\alpha}(d) \stackrel{\text{def}}{=} \mathbf{1}_{\{\alpha(X)\}}(d) \wedge s^{\mathcal{I}}(d) \wedge \bigwedge_{1 \leq i \leq n} [\![t_i]\!]^{\mathcal{I},\alpha}(f_i^{\mathcal{I}}(d)).$$

The denotation of t in the interpretation $\mathcal{I}$ is defined as[5]

$$[\![t]\!]^{\mathcal{I}} \stackrel{\text{def}}{=} \biguplus_{\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}} [\![t]\!]^{\mathcal{I},\alpha}.$$

[5]The symbol $\uplus$ denotes the union of fuzzy subsets (Definition 1.23).

Definition 4.6 is a direct generalization of the crisp denotation of an OSF term in an OSF interpretation under an assignment α (Definition 3.19). Note that the crisp denotation of a term is always a singleton or the empty set, while in the fuzzy setting $[\![t]\!]^{\mathcal{I},\alpha}$ has a value greater than 0 for at most one element $d \in \Delta^{\mathcal{I}}$.

Example 4.7 (Fuzzy denotation of an OSF term). Continuing from Example 4.5, let t be the term $X : \mathit{thriller}\,(\mathit{directed\_by} \to Y : \mathit{director})$ and α be an assignment such that $\alpha(X) = \mathit{halloween}$ and $\alpha(Y) = \mathit{carpenter}$. Then (with $h \stackrel{\text{def}}{=} \mathit{halloween}$ and $c \stackrel{\text{def}}{=} \mathit{carpenter}$):

$$\begin{aligned}
[\![t]\!]^{\mathcal{I},\alpha}(h) &= \mathbf{1}_{\{\alpha(X)\}}(h) \wedge \mathit{thriller}^{\mathcal{I}}(h) \wedge [\![Y : \mathit{director}]\!]^{\mathcal{I},\alpha}(\mathit{directed_by}^{\mathcal{I}}(h)) \\
&= 1 \wedge 0.5 \wedge [\![Y : \mathit{director}]\!]^{\mathcal{I},\alpha}(c) \\
&= 1 \wedge 0.5 \wedge \mathbf{1}_{\{\alpha(Y)\}}(c) \wedge \mathit{director}^{\mathcal{I}}(c) \\
&= 1 \wedge 0.5 \wedge 1 \wedge 1 = 0.5.
\end{aligned}$$

Now consider the term

$$t' = X : \mathit{movie}\,(\mathit{directed\_by} \to Y : \mathit{director}, \mathit{directed\_by} \to Y : \mathit{string})$$

and note that $\mathit{director} \curlywedge \mathit{string} = \bot$. It follows that the denotation of t' in any fuzzy OSF interpretation $\mathcal{I}$ is always equal to 0, i.e., the term t' is contradictory. Indeed, suppose towards a contradiction that $[\![t']\!]^{\mathcal{I},\alpha}(d) > 0$ for some OSF interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, $d \in \Delta^{\mathcal{I}}$ and valuation α. Then it must be the case that $\mathit{director}^{\mathcal{I}}(d') \wedge \mathit{string}^{\mathcal{I}}(d') > 0$ or, equivalently, $(\mathit{director}^{\mathcal{I}} \curlywedge \mathit{string}^{\mathcal{I}})(d') > 0$, where $d' = \alpha(Y)$. From Definition 4.4 it follows that $(\mathit{director} \curlywedge \mathit{string})^{\mathcal{I}}(d') = \bot^{\mathcal{I}}(d') > 0$, which is impossible since $\bot^{\mathcal{I}} = \mathbf{1}_{\emptyset}$. ▷

Remark 5 (Fuzzy denotation of an OSF term). Let $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$ be an OSF term and $X_i = \mathit{RootTag}(t_i)$ for each i. Let $\mathcal{I}$ be an OSF interpretation and $d \in \Delta^{\mathcal{I}}$. It is easy to see that, if α_0 and α_1 are assignments such that

$$[\![t]\!]^{\mathcal{I},\alpha_0}(d) = \beta_0 > 0 \quad \text{and} \quad [\![t]\!]^{\mathcal{I},\alpha_1}(d) = \beta_1 > 0,$$

then it must be the case that $\alpha_0(Y) = \alpha_1(Y)$ for all $Y \in \mathit{Tags}(t)$, and thus $\beta_0 = \beta_1$. Indeed, it must be the case that $\alpha_0(X) = \alpha_1(X) = d$, and so for each f_i is must hold that $\alpha_0(X_i) = f_i^{\mathcal{I}}(d) = \alpha_1(X_i)$, and so on for all subterms. Hence the set $\{\beta \mid [\![t]\!]^{\mathcal{I},\alpha}(d) = \beta > 0 \text{ for some } \alpha : \mathcal{V} \to \Delta^{\mathcal{I}}\}$ is a singleton and if $[\![t]\!]^{\mathcal{I}}(d) = \bigcup_{\alpha:\mathcal{V}\to\Delta^{\mathcal{I}}} [\![t]\!]^{\mathcal{I},\alpha}(d) = \beta > 0$, then there exists some α such that $[\![t]\!]^{\mathcal{I},\alpha}(d) = \beta$.

We now define the graded satisfaction of an OSF clause in a fuzzy OSF interpretation, generalizing Definition 3.21.

Definition 4.8 (Graded satisfaction of an OSF clause). If $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a fuzzy OSF interpretation and $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ is an assignment, then *the satisfaction of an OSF clause* ϕ

to a degree $\beta \in [0,1]$ *in the interpretation* $\mathcal{I}$ *under the assignment* α (notation: $\mathcal{I}, \alpha \models_\beta \phi$) is defined recursively as follows:

$$\begin{aligned}
\mathcal{I}, \alpha \models_\beta X : s \quad &\Leftrightarrow \quad s^\mathcal{I}(\alpha(X)) \geq \beta, \\
\mathcal{I}, \alpha \models_\beta X \doteq Y \quad &\Leftrightarrow \quad \mathbf{1}_{\{\alpha(X)\}}(\alpha(Y)) \geq \beta, \\
\mathcal{I}, \alpha \models_\beta X.f \doteq Y \quad &\Leftrightarrow \quad \mathbf{1}_{\{f^\mathcal{I}(\alpha(X))\}}(\alpha(Y)) \geq \beta, \\
\mathcal{I}, \alpha \models_\beta \phi \,\&\, \phi' \quad &\Leftrightarrow \quad \mathcal{I}, \alpha \models_\beta \phi \text{ and } \mathcal{I}, \alpha \models_\beta \phi'.
\end{aligned}$$

If $\mathcal{I}, \alpha \models_\beta \phi$, then α is called a *solution* in $\mathcal{I}$ for the clause ϕ with degree β, and ϕ is said to be *satisfiable* in $\mathcal{I}$ with degree β. The clause ϕ is said to be satisfiable in $\mathcal{I}$ if there is some $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$ and $\beta > 0$ such that $\mathcal{I}, \alpha \models_\beta \phi$, and ϕ is said to be satisfiable if there is some $\mathcal{I}$ such that ϕ is satisfiable in $\mathcal{I}$.

Note that $\mathcal{I}, \alpha \models_0 \phi$ always holds for any fuzzy OSF interpretation $\mathcal{I}$, assignment α and OSF clause ϕ (the satisfaction degree of a clause for a given assignment and interpretation is always greater than or equal to 0). Moreover, for any $\beta \in (0,1]$,

$$\begin{aligned}
&\mathcal{I}, \alpha \models_\beta X \doteq Y \quad \Leftrightarrow \mathbf{1}_{\{\alpha(X)\}}(\alpha(Y)) \geq \beta \quad \Leftrightarrow \alpha(X) = \alpha(Y) \quad \Leftrightarrow \mathcal{I}, \alpha \models_1 X \doteq Y, \\
&\mathcal{I}, \alpha \models_\beta X.f \doteq Y \Leftrightarrow \mathbf{1}_{\{f^\mathcal{I}(\alpha(X))\}}(\alpha(Y)) \geq \beta \Leftrightarrow f^\mathcal{I}(\alpha(X)) = \alpha(Y) \Leftrightarrow \mathcal{I}, \alpha \models_1 X.f \doteq Y,
\end{aligned}$$

and

$$\begin{aligned}
\alpha(X) \neq \alpha(Y) \quad &\Rightarrow \quad \mathbf{1}_{\{\alpha(X)\}}(\alpha(Y)) = 0 \quad \Rightarrow \quad \mathcal{I}, \alpha \models_0 X \doteq Y, \\
f^\mathcal{I}(\alpha(X)) \neq \alpha(Y) \quad &\Rightarrow \quad \mathbf{1}_{\{f^\mathcal{I}(\alpha(X))\}}(\alpha(Y)) = 0 \quad \Rightarrow \quad \mathcal{I}, \alpha \models_0 X.f \doteq Y.
\end{aligned}$$

Example 4.9 (Graded satisfaction of an OSF clause). Continuing from Example 4.5, let $\phi = X : \mathit{slasher} \,\&\, X.\mathit{directed\_by} \doteq Y \,\&\, Y : \mathit{director}$ and let α be an assignment such that $\alpha(X) = \mathit{psycho}$ and $\alpha(Y) = \mathit{hitchcock}$. Then $\mathcal{I}, \alpha \models_{0.7} \phi$:

- $\mathcal{I}, \alpha \models_{0.7} X : \mathit{slasher}$, since $\mathit{slasher}^\mathcal{I}(\alpha(X)) = \mathit{slasher}^\mathcal{I}(\mathit{psycho}) = 0.7$;
- $\mathcal{I}, \alpha \models_{0.7} X.\mathit{directed\_by} \doteq Y$, since $\mathit{directed\_by}^\mathcal{I}(\alpha(X)) = \mathit{directed\_by}^\mathcal{I}(\mathit{psycho}) = \mathit{hitchcock} = \alpha(Y)$; and
- $\mathcal{I}, \alpha \models_{0.7} Y : \mathit{director}$, since $\mathit{director}^\mathcal{I}(\alpha(Y)) = \mathit{director}^\mathcal{I}(\mathit{hitchcock}) = 1$. ▷

As seen in Chapter 3, as far as crisp OSF logic is concerned, the denotation of an OSF term and the satisfaction of its corresponding OSF clause are equivalent (Proposition 3.23). A similar relationship can be proved for the denotation of an OSF term t in a fuzzy interpretation and the degree of satisfaction of the corresponding OSF clause $\phi(t)$.

Proposition 4.10 (Equivalence of fuzzy term denotation and graded constraint satisfaction). *For every OSF term* t *(with root variable* X*), every interpretation* $\mathcal{I}$*, every*

assignment α*, and every* $\beta \in [0,1]$*:*

$$[\![t]\!]^{\mathcal{I},\alpha}(\alpha(X)) \geq \beta \;\Leftrightarrow\; \mathcal{I}, \alpha \models_\beta \phi(t).$$

Therefore $[\![t]\!]^{\mathcal{I},\alpha}(\alpha(X)) = \sup(\{\beta \mid \mathcal{I}, \alpha \models_\beta \phi(t)\})$ *and thus, for all* $d \in \Delta^\mathcal{I}$*:*

$$[\![t]\!]^{\mathcal{I}}(d) = \sup(\{\beta \mid \alpha \in \mathit{Val}(\mathcal{I}) \text{ and } \alpha(X) = d \text{ and } \mathcal{I}, \alpha \models_\beta \phi(t)\}).$$

In crisp OSF logic, the OSF clause ϕ' resulting from the application of a constraint normalization rule of Fig. 3.2 is semantically equivalent to the original clause ϕ. In a fuzzy context, the constraint normalization rules of Fig. 4.2 still preserve the satisfiability of an OSF clause, but not necessarily the satisfaction degree.

Proposition 4.11 (Solution-preservation of fuzzy OSF clause normalization). *For any rule of Fig. 4.2 with premise* ϕ *and conclusion* ϕ'*, for every fuzzy OSF algebra* $\mathcal{I}$ *and assignment* α*:* $\mathcal{I}, \alpha \models_\beta \phi$ *for some* $\beta > 0$ *if and only if* $\mathcal{I}, \alpha \models_{\beta'} \phi'$ *for some* $\beta' > 0$*.*

Note that the degrees β and β' in the last proposition may be not be the same, so that the degree of satisfaction of an OSF constraint in an algebra $\mathcal{I}$ according to some assignment α may be different from the degree of satisfaction of its solved form in the same algebra and according to the same assignment. Together with Proposition 4.10, this implies that the denotation of an OSF term t in a fuzzy OSF interpretation may not coincide with the denotation of its normal form ψ obtained by applying the constraint normalization procedure to $\phi(t)$. This constitutes a significant departure from crisp OSF logic, where in any OSF algebra every OSF clause is equivalent to its solved form, and every OSF term has the same denotation as its normal form.

Example 4.12 (Solution-preservation of fuzzy OSF clause normalization). Consider the fuzzy OSF signature of Example 4.2 and the OSF clause $\phi \stackrel{\text{def}}{=} X : \mathit{thriller} \;\&\; X : \mathit{horror}$. An application of the rule Sort Intersection to ϕ results in the OSF clause $\phi' \stackrel{\text{def}}{=} X : \mathit{slasher}$. Let $\mathcal{I}$ be the fuzzy OSF interpretation from Example 4.5, and α be an assignment such that $\alpha(X) = \mathit{psycho}$. Then $\mathcal{I}, \alpha \models_1 \phi$, but $\mathcal{I}, \alpha \models_{0.7} \phi'$. As another example, if α' is an assignment such that $\alpha'(X) = \mathit{halloween}$, then $\mathcal{I}, \alpha' \models_{0.5} \phi$, and $\mathcal{I}, \alpha' \models_1 \phi'$. ▷

We conclude the section by exploring subalgebras of fuzzy OSF algebras.

Definition 4.13 (Subalgebra of a fuzzy OSF algebra). Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ and $\mathcal{J} = (\Delta^\mathcal{J}, \cdot^\mathcal{J})$ be fuzzy OSF interpretations. The algebra $\mathcal{I}$ *is a subalgebra of* $\mathcal{J}$ if $\Delta^\mathcal{I} \subseteq \Delta^\mathcal{J}$ and for all $d \in \Delta^\mathcal{I}$, all $s \in \mathcal{S}$ and all $f \in \mathcal{F}$: $s^\mathcal{I}(d) = s^\mathcal{J}(d)$ and $f^\mathcal{I}(d) = f^\mathcal{J}(d)$.

Similarly to the crisp case, the denotation of an OSF term t under a valuation $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$ is the same in an algebra $\mathcal{J}$ and in a subalgebra $\mathcal{I}$ of $\mathcal{J}$. Moreover, an assignment $\alpha : \mathcal{V} \to \Delta^\mathcal{I}$

is a solution for a clause ϕ in $\mathcal{I}$ if and only if it is a solution for ϕ in $\mathcal{J}$, with the same degree of satisfaction.

Proposition 4.14 (Fuzzy denotation and graded satisfaction in subalgebras). *Let $\mathcal{I}$ be a subalgebra of a fuzzy OSF algebra $\mathcal{J}$. For every OSF term t, every OSF clause ϕ, every assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ and every $\beta \in [0,1]$: (i) $[\![t]\!]^{\mathcal{I},\alpha}(d) = [\![t]\!]^{\mathcal{J},\alpha}(d)$ for all $d \in \Delta^{\mathcal{I}}$, and (ii) $\mathcal{I}, \alpha \models_\beta \phi$ if and only if $\mathcal{J}, \alpha \models_\beta \phi$.*

Finally, the definition of a subalgebra generated by a subset of a fuzzy algebra is a straightforward generalization of the same notion of crisp OSF logic (Definition 3.28).

Definition 4.15 (Fuzzy OSF subalgebra generated by a set). Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a fuzzy OSF interpretation and $D \subseteq \Delta^{\mathcal{I}}$ be nonempty. *The fuzzy OSF subalgebra generated by D* is the structure $\mathcal{I}[D] = (\mathcal{F}^*(D), \cdot^{\mathcal{I}[D]})$ such that, for each $s \in \mathcal{S}$, $s^{\mathcal{I}[D]} \stackrel{\text{def}}{=} s^{\mathcal{I}} \cap \mathbf{1}_{\mathcal{F}^*(D)}$, and for each $f \in \mathcal{F}$, $f^{\mathcal{I}[D]}$ is defined as the restriction of $f^{\mathcal{I}}$ to $\mathcal{F}^*(D)$.

When $D = \{d\}$ is a singleton we write $\mathcal{I}[d]$ instead of $\mathcal{I}[\{d\}]$.

Proposition 4.16 (Least fuzzy OSF subalgebra generated by a set). *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an OSF interpretation. For any non-empty subset $D \subseteq \Delta^{\mathcal{I}}$ the structure $\mathcal{I}[D]$ is the least fuzzy subalgebra of $\mathcal{I}$ containing D.*

4.4 The fuzzy OSF graph algebra

As seen in Chapter 3, OSF graphs play a fundamental role in the semantics of crisp OSF logic [10]. The same is true for the fuzzy generalization of this language. In particular, OSF graphs are the elements of the domain of the fuzzy OSF graph algebra, a fuzzy OSF interpretation which will be essential for proving many results about fuzzy OSF logic. This fuzzy interpretation is defined next.[6]

Definition 4.17 (Fuzzy OSF graph algebra). The *fuzzy OSF graph algebra* is the pair $\mathcal{G} = (\Delta^{\mathcal{G}}, \cdot^{\mathcal{G}})$ defined as follows.

1. The domain $\Delta^{\mathcal{G}}$ is the set of all OSF graphs.
2. For each $s \in \mathcal{S}$ and for each graph $g = (N, E, \lambda_N, \lambda_E, X) \in \Delta^{\mathcal{G}}$: $s^{\mathcal{G}}(g) \stackrel{\text{def}}{=} \preceq(\lambda_N(X), s)$.
3. For each $f \in \mathcal{F}$, the function $f^{\mathcal{G}} : \Delta^{\mathcal{G}} \to \Delta^{\mathcal{G}}$ is defined by letting, for any $g = (N, E, \lambda_N, \lambda_E, X)$:

$$f^{\mathcal{G}}(g) \stackrel{\text{def}}{=} \begin{cases} g|_Y & \text{if } \exists Y \in N \text{ such that } \lambda_E(X, Y) = f, \\ G(Z_{f,g} : \top) & \text{otherwise,} \end{cases}$$

[6] Note that the same symbol $\mathcal{G}$ is used for the crisp OSF graph algebra and for the fuzzy OSF graph algebra, since its meaning is always clear from context.

where

- $g|_Y$ is the maximally connected subgraph of g rooted at Y; and
- $G(Z_{f,g} : \top)$ denotes the trivial OSF graph $(\{Z_{f,g}\}, \emptyset, \{(Z_{f,g}, \top)\}, \emptyset, Z_{f,g})$ whose only node is the new variable $Z_{f,g} \in \mathcal{V} \setminus N$ – labeled by $\top$ – which is uniquely determined by the feature f and the graph g, i.e., if $f \neq f'$ or $g \neq g'$, then $Z_{f,g} \neq Z_{f',g'}$.

This definition generalizes the OSF graph algebra of Definition 3.36. The interpretation of feature symbols is the same (e.g., Example 3.38), but the denotation of a sort symbol s is now a fuzzy set whose value for a graph g with root X is the subsumption degree of the root sort $\lambda_N(X)$ with respect to s.

Example 4.18 (Interpretation of sorts in the fuzzy OSF graph algebra). Consider the fuzzy subsumption relation corresponding to the graph of Fig. 4.1. Then, for example, $\mathit{thriller}^{\mathcal{G}}$ is the function such that:

- $\mathit{thriller}^{\mathcal{G}}(g) = 1$ for any OSF graph g whose root is labeled by *thriller* or by $\bot$;
- $\mathit{thriller}^{\mathcal{G}}(g) = 0.5$ for any OSF graph g whose root is labeled by *slasher*; and
- $\mathit{thriller}^{\mathcal{G}}(g) = 0$ for any other OSF graph g. ▷

Proposition 4.19 (Fuzzy OSF graph algebra). *The fuzzy OSF graph algebra of Definition 4.17 is a fuzzy OSF interpretation in the sense of Definition 4.4.*

Fuzzy OSF logic preserves the result from crisp OSF logic (Theorem 3.41) stating that every solved OSF clause ϕ is satisfiable in the subalgebra of the OSF graph algebra generated by the graphs corresponding to the maximal rooted subclauses of ϕ. In fuzzy OSF logic, it can be proved that in these subalgebras solved OSF clauses can always be satisfied with the maximum degree, i.e., 1.

Definition 4.20 (Fuzzy canonical graph algebra). Let ϕ be a solved OSF clause. The subalgebra $\mathcal{G}[\Delta^{\mathcal{G},\phi}]$ of the fuzzy OSF graph algebra $\mathcal{G}$ generated by $\Delta^{\mathcal{G},\phi} \stackrel{\text{def}}{=} \{G(\phi(X)) \mid X \in \mathit{Tags}(\phi)\}$ is called the *(fuzzy) canonical graph algebra* induced by ϕ.

Proposition 4.21 (Satisfiability in the fuzzy canonical graph algebra). *Any solved clause $\phi \in \Phi$ is satisfiable in $\mathcal{G}[\Delta^{\mathcal{G},\phi}]$ with degree 1 under any assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]}$ such that $\alpha(Y) = G(\phi(Y))$ for all $Y \in \mathit{Tags}(\phi)$.*

Since $\mathcal{G}[\Delta^{\mathcal{G},\phi}]$ is a subalgebra of $\mathcal{G}$, as a corollary of Propositions 4.14 and 4.21 it is possible to prove that every solved OSF clause is satisfiable in $\mathcal{G}$ (with degree 1) under any assignment mapping each variable Y of ϕ to the graph $G(\phi(Y))$, thus carrying Corollary 3.42 to fuzzy OSF logic. Analogously to crisp OSF logic, such a mapping is called a *canonical solution* for the clause ϕ in $\mathcal{G}$.

Corollary 4.22 (Canonical solutions in the fuzzy OSF graph algebra). *Any solved OSF clause ϕ is satisfiable in the fuzzy OSF graph algebra $\mathcal{G}$ with degree 1 under any assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{G}}$ such that, for each $Y \in Tags(\phi)$, $\alpha(Y) = G(\phi(Y))$.*

4.5 Fuzzy OSF algebra homomorphisms

In this section we introduce fuzzy β-homomorphisms, mappings between fuzzy OSF interpretations that preserve feature applications and, to some degree, the sorts of the elements of the domain. Fuzzy β-morphisms allow us to prove several results regarding the satisfaction of OSF clauses in fuzzy OSF interpretations. In particular, we prove that the OSF graph algebra $\mathcal{G}$ is canonical in the sense that any OSF clause is satisfiable if and only if it is satisfiable in $\mathcal{G}$. We also prove that the denotation of a normal OSF term ψ in a fuzzy OSF interpretation can be characterized through the existence of fuzzy homomorphisms from the subalgebra of $\mathcal{G}$ generated by $G(\psi)$. In the next section, β-morphisms will be used to define a fuzzy ordering on the domain of any fuzzy OSF interpretation, which will eventually lead to the fuzzy subsumption ordering between OSF terms.

Definition 4.23 (Fuzzy OSF algebra β-homomorphism). A *β-homomorphism* (or *β-morphism*) $\gamma : \mathcal{I} \to \mathcal{J}$ between two fuzzy OSF interpretations $\mathcal{I}$ and $\mathcal{J}$ is a function $\gamma : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ such that, for all $f \in \mathcal{F}$, all $s \in \mathcal{S}$, and all $d \in \Delta^{\mathcal{I}}$:

- $\gamma(f^{\mathcal{I}}(d)) = f^{\mathcal{J}}(\gamma(d))$, and
- $s^{\mathcal{I}}(d) \wedge \beta \leq s^{\mathcal{J}}(\gamma(d))$.

Our definition of fuzzy OSF algebra β-morphism generalizes the corresponding crisp one of Definition 3.43. The condition on features is unchanged: a β-morphism must preserve the structure of the input algebra $\mathcal{I}$, i.e., it must commute with feature applications. The original crisp condition on sorts states that, whenever d is an element of the interpretation of s in $\mathcal{I}$, then its image $\gamma(d)$ must be an element of the interpretation of s in $\mathcal{J}$. A direct fuzzy generalization of this statement would specify that, for all $s \in \mathcal{S}$ and $d \in \Delta^{\mathcal{I}}$,

$$s^{\mathcal{I}}(d) \leq s^{\mathcal{J}}(\gamma(d)). \tag{4.4}$$

Similarly to our definition of a fuzzy subsumption relation, we further generalize this constraint by requiring that, in order for γ to be a β-morphism, whenever d is a member of $s^{\mathcal{I}}$ with degree β', then $\gamma(d)$ must be a member of $s^{\mathcal{J}}$ with degree greater than or equal to the minimum of β and β'. Clearly (4.4) is recovered simply by setting $\beta = 1$.

Example 4.24 (Fuzzy OSF algebra β-morphism). Consider the fuzzy interpretation $\mathcal{I}$, the assignment α and the term $t = X : thriller\,(directed\_by \to Y : director)$ from

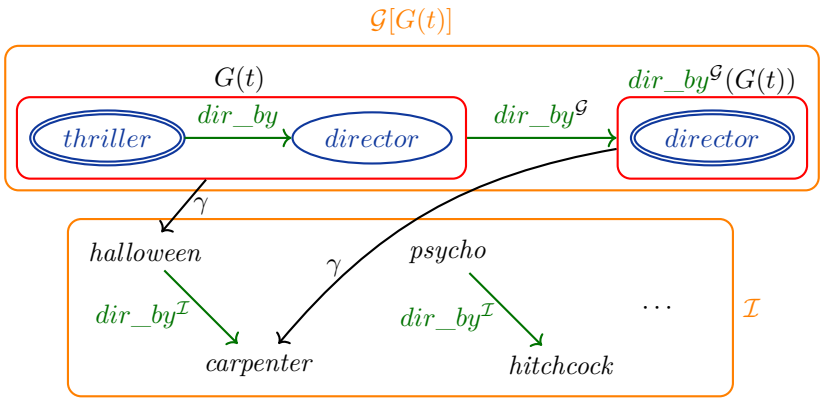


Figure 4.3: The fuzzy OSF algebra morphism of Example 4.24.

Example 4.7. Consider the subalgebra of $\mathcal{G}$ generated from the element $G(t)$ and define the function $\gamma : \Delta^{\mathcal{G}[G(t)]} \to \Delta^{\mathcal{I}}$ by setting, for $g = w^{\mathcal{G}}(G(t)) \in \Delta^{\mathcal{G}[G(t)]}$, $\gamma(g) \stackrel{\text{def}}{=} w^{\mathcal{I}}(halloween)$, where $w \in \mathcal{F}^*$. In particular we have $\gamma(G(t)) = halloween$ and $\gamma(G(Y : director)) = carpenter$. This is easily verified to be a 0.5-morphism. This is depicted in Fig. 4.3 (where trivial graphs are not shown, and some names have been shortened). ▷

The following proposition provides a few properties of fuzzy OSF algebra homomorphisms. In particular, if a function $\gamma : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ is a β-morphisms, then there is always a maximum β' such that γ is also a β'-morphism.

Proposition 4.25 (Fuzzy homomorphisms). *Let $\gamma : \mathcal{I} \to \mathcal{J}$ be a β-morphism.*

1. *If $\gamma' : \mathcal{J} \to \mathcal{K}$ is a β'-morphism, then $\gamma' \circ \gamma : \mathcal{I} \to \mathcal{K}$ is a $\beta \wedge \beta'$-morphism.*
2. *For all $\beta' \leq \beta$: γ is a β' morphism.*
3. *There is a maximum β' such that γ is a β'-morphism.*

On the other hand, as Example 4.26 shows, it is not true in general that, for any fuzzy interpretations $\mathcal{I}$ and $\mathcal{J}$, there is a maximum β such that there exists a β-morphism $\gamma : \mathcal{I} \to \mathcal{J}$. This property, which will be valuable later, holds however for specific homomorphisms relating subalgebras generated by singletons, as stated in Proposition 4.27.

Example 4.26 (Homomorphism degrees). Consider a simple signature where $\mathcal{S} = \{\bot, s, \top\}$, the subsumption ordering is $\bot \preceq_1 s \preceq_1 \top$, and $\mathcal{F} = \{f\}$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be such that $\Delta^{\mathcal{I}} = \{a\}$, $s^{\mathcal{I}}(a) = 1$, and $f^{\mathcal{I}}(a) = a$. Let $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be such that

- $\Delta^{\mathcal{J}} = \{a_i \mid i \in \mathbb{N} \setminus \{0\}\}$;
- $\forall i > 0$: $s^{\mathcal{J}}(a_i) = 1 - 1/i$;
- $\forall i > 0$: $f^{\mathcal{J}}(a_i) = a_i$.

For each $i > 0$, let $\beta_i \stackrel{\text{def}}{=} 1 - 1/i$. Any morphism $\gamma : \mathcal{I} \to \mathcal{J}$ has shape $\gamma_i : a \mapsto a_i$ for some $i > 0$. Note that for each $i > 0$, γ_i is a β_i-morphism, since $s^{\mathcal{I}}(a) \wedge \beta_i = 1 \wedge (1-1/i) = 1-1/i = s^{\mathcal{J}}(a_i) = s^{\mathcal{J}}(\gamma_i(a))$. Let $B \stackrel{\text{def}}{=} \{\beta \mid \exists\ \beta\text{-morphism } \gamma' : \mathcal{I} \to \mathcal{J}\}$. Clearly $B = \{\beta_i \mid i \in \mathbb{N} \setminus \{0\}\}$ and $\sup(B) = 1$, but there is no 1-morphism from $\mathcal{I}$ to $\mathcal{J}$. ▷

Proposition 4.27 (Homomorphisms originating from singletons). *Let $\mathcal{I}$ and $\mathcal{J}$ be fuzzy OSF interpretations and fix $d \in \Delta^{\mathcal{I}}$. Let $\gamma : \mathcal{I}[d] \to \mathcal{J}$ be a β-morphism.*

1. *For every $d' \in \mathcal{I}[d]$, $\gamma(d')$ is an element of the domain of $\mathcal{J}[\gamma(d)]$, i.e., $\gamma : \mathcal{I}[d] \to \mathcal{J}[\gamma(d)]$.*
2. *For any β'-homomorphism $\gamma' : \mathcal{I}[d] \to \mathcal{J}$: if $\gamma'(d) = \gamma(d)$, then $\gamma = \gamma'$, i.e., for all $d' \in \Delta^{\mathcal{I}[d]}$, $\gamma(d') = \gamma'(d')$.*
3. *There is a maximum β' such that there exists a β'-morphism $\gamma' : \mathcal{I}[d] \to \mathcal{J}$ that satisfies $\gamma'(d) = \gamma(d)$.*

As should be expected, the satisfiability of an OSF clause ϕ is preserved under an OSF algebra β-morphism $\gamma : \mathcal{I} \to \mathcal{J}$, modulo the degree β: if ϕ is satisfiable in $\mathcal{I}$ with degree β', then ϕ must also be satisfiable in $\mathcal{J}$ with degree $\beta \wedge \beta'$.

Proposition 4.28 (Extending solutions through fuzzy homomorphisms). *Let $\mathcal{I}$ and $\mathcal{J}$ be two fuzzy OSF interpretations and $\gamma : \mathcal{I} \to \mathcal{J}$ be a β-morphism. For every OSF clause ϕ and assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$, if $\mathcal{I}, \alpha \models_{\beta_{\mathcal{I}}} \phi$, then $\mathcal{J}, \gamma \circ \alpha \models_{\beta_{\mathcal{I}} \wedge \beta} \phi$.*

An interesting property that carries over from crisp OSF logic (Theorem 3.47) is that it is always possible to define a β-morphism from any fuzzy interpretation $\mathcal{I}$ into the fuzzy OSF graph algebra for some positive degree β. Consequently, it holds that any OSF clause is satisfiable if and only if it is satisfiable in $\mathcal{G}$ (Corollary 4.30), generalizing the analogous crisp result (Corollary 3.48).

Theorem 4.29 (Fuzzy morphisms into $\mathcal{G}$). *For any fuzzy OSF interpretation $\mathcal{I}$ there exists a β-homomorphism into the fuzzy OSF graph algebra $\mathcal{G}$ for some $\beta \in (0, 1]$.*

Corollary 4.30 (Canonicity of the fuzzy OSF graph algebra). *An OSF clause is satisfiable if and only if it is satisfiable in the fuzzy OSF graph algebra.*

Another core property that is preserved from crisp OSF logic (Theorem 3.46) is that any solution α for a clause ϕ in any fuzzy interpretation $\mathcal{I}$ can be obtained through a homomorphism from the canonical graph algebra induced by ϕ. Specifically, if α is a solution to ϕ in $\mathcal{I}$ with degree β, then it is possible to define a β-homomorphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ such that α is the homomorphic image of the canonical solution for ϕ in $\mathcal{G}$, as stated next. An example of the application of this theorem was given in Example 4.24.

Theorem 4.31 (Extracting solutions through fuzzy homomorphisms). *For any solved OSF clause ϕ, fuzzy interpretation $\mathcal{I}$, assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ and $\beta \in (0,1]$ such that $\mathcal{I}, \alpha \models_\beta \phi$ there exists an OSF algebra β-homomorphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ such that $\alpha(X) = \gamma(G(\phi(X)))$ for each $X \in Tags(\phi)$.*

Thanks to Proposition 4.10 and Theorem 4.31 we can show that the denotation of a normal OSF term ψ in a fuzzy OSF interpretation can be characterized through the existence of fuzzy homomorphisms from the canonical graph algebra induced by $G(\psi)$, generalizing the analogous result of crisp OSF logic (Theorem 3.49).

Theorem 4.32 (Denotation of ψ-terms via fuzzy morphisms). *Let ψ be a normal OSF term, let $\phi = \phi(\psi)$, and let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a fuzzy OSF interpretation. For all $d \in \Delta^{\mathcal{I}}$ and $\beta \in (0,1]$:*

$$[\![\psi]\!]^{\mathcal{I}}(d) \geq \beta \;\Leftrightarrow\; \text{there is a } \beta\text{-morphism } \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \text{ such that } d = \gamma(G(\psi))$$

and thus $[\![\psi]\!]^{\mathcal{I}}(d) = \sup(\{\beta \in [0,1] \mid \exists \beta\text{-morphism } \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \text{ such that } d = \gamma(G(\psi))\})$.

4.6 Fuzzy OSF orderings and subsumption

One of the main features of (crisp) OSF logic [10] is its definition of a subsumption ordering between OSF terms, which extends the subsumption ordering between sort symbols. This notion allows to understand whether a concept represented as an OSF term is more general or more specific than another concept, thereby creating a concept hierarchy. As discussed in Section 3.5, equivalent partial orders can also be defined on rooted solved OSF clauses and OSF graphs.

In this section we generalize this result to a fuzzy setting by defining a fuzzy subsumption ordering between OSF terms, a graded implication ordering between OSF clauses, and a fuzzy approximation ordering between OSF graphs. We show that these fuzzy relations are equivalent and that they constitute fuzzy partial orders[7] on, respectively, OSF graphs, normal OSF terms and rooted solved OSF clauses. We give both a *semantic* definition and a *syntactic* definition of fuzzy subsumption of OSF terms, and prove their equivalence. A connection between the crisp definition of subsumption of OSF terms (Definition 3.57) and our fuzzy generalization is also provided.

We begin by defining a fuzzy preorder on the domain of any fuzzy OSF interpretation, generalizing Definition 3.50.

Definition 4.33 (Fuzzy endomorphic approximation). On every fuzzy OSF interpretation $\mathcal{I}$ a fuzzy binary relation $\sqsubseteq^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0,1]$ is defined by letting, for all

[7] To be more precise, these fuzzy orders are antisymmetric on *equivalence classes* of OSF graphs, normal OSF terms and rooted solved OSF clauses, as detailed later.

$d, d' \in \Delta^{\mathcal{I}}$:

$$\sqsubseteq^{\mathcal{I}}(d, d') \stackrel{\text{def}}{=} \sup(\{\beta \mid \gamma(d) = d' \text{ for some } \beta\text{-homomorphism } \gamma : \mathcal{I}[d] \to \mathcal{I}[d']\}).$$

If $\sqsubseteq^{\mathcal{I}}(d, d') = \beta$ (abbreviated as $d \sqsubseteq^{\mathcal{I}}_{\beta} d'$) we say that d *approximates* d' with degree β.

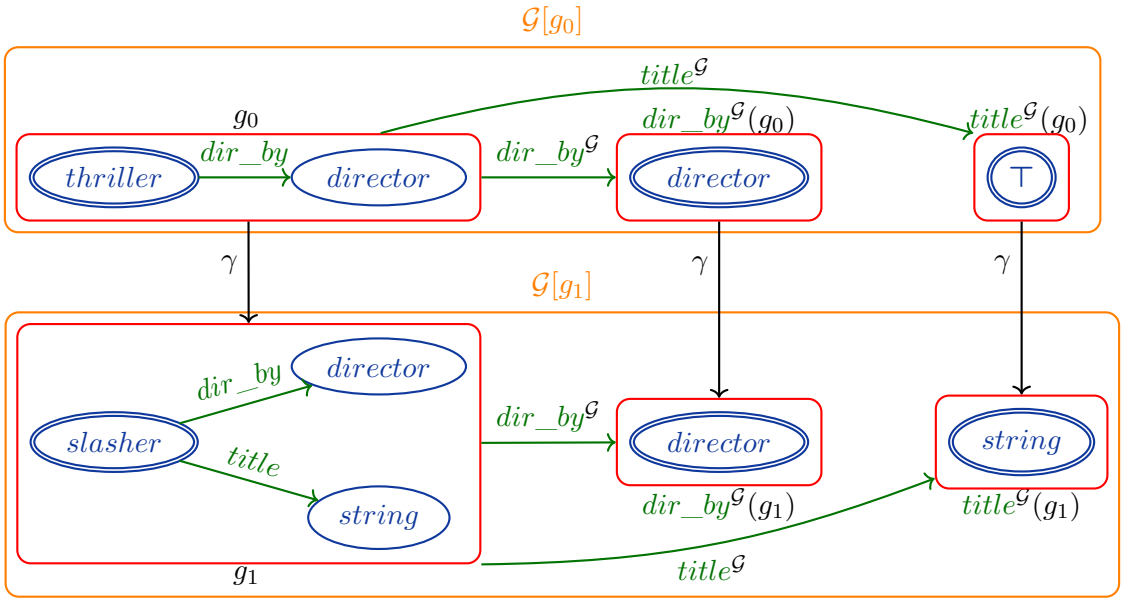


Figure 4.4: The fuzzy OSF algebra morphism of Example 4.34.

Example 4.34 (OSF graphs fuzzy approximation ordering). Let

$$g_0 = G(X_0 : thriller\,(directed\_by \to Y_0 : director)) \text{ and}$$
$$g_1 = G(X_1 : slasher\,(directed\_by \to Y_1 : director, title \to Z_1 : string)).$$

Define $\gamma : \Delta^{\mathcal{G}[g_0]} \to \Delta^{\mathcal{G}[g_1]}$ by letting, for each $g = w^{\mathcal{G}}(g_0)$ in $\Delta^{\mathcal{G}[g_0]}$ (with $w \in \mathcal{F}^*$): $\gamma(g) \stackrel{\text{def}}{=} w^{\mathcal{G}}(g_1)$. The graphs g_0 and g_1 and the function γ are depicted in Fig. 4.4 (where not all trivial graphs are shown, and some names are shortened). In particular $\gamma(g_0) = g_1$. The function γ is a 0.5-morphism witnessing $\sqsubseteq^{\mathcal{G}}(g_0, g_1) = 0.5$. ▷

Remark 6. Let $\mathcal{I}$ be a fuzzy OSF algebra and $d, d' \in \Delta^{\mathcal{I}}$. Suppose that $\sqsubseteq^{\mathcal{I}}(d, d') = \beta' > 0$. Then Proposition 4.27 guarantees that there exists a β'-morphism $\gamma' : \mathcal{I}[d] \to \mathcal{I}[d']$ such that $\gamma'(d) = d'$, i.e., that $\beta' = \sup(\{\beta \mid \gamma(d) = d' \text{ for some } \beta\text{-morphism } \gamma : \mathcal{I}[d] \to \mathcal{I}[d']\})$ is a maximum.

Proposition 4.35 (Endomorphic approximation fuzzy preorder). *For all fuzzy OSF interpretations $\mathcal{I}$, the fuzzy binary relation $\sqsubseteq^{\mathcal{I}}$ is a fuzzy preorder.*

Similarly to the crisp setting (Proposition 3.51), the approximation ordering on OSF graphs does not satisfy (fuzzy) antisymmetry, that is, it may be the case that $\sqsubseteq^{\mathcal{G}}(g_0, g_1) = \beta_0 > 0$ and $\sqsubseteq^{\mathcal{G}}(g_1, g_0) = \beta_1 > 0$, but $g_0 \neq g_1$. However, while g_0 and g_1 may not be identical, it is possible to show that they are essentially the same graph except for different variable names or possibly the presence of trivial subgraphs (subgraphs consisting of a single node labeled $\top$), like in the crisp case (Proposition 3.53).

Proposition 4.36 (Fuzzy morphisms between two OSF graphs). *Let g_0 and g_1 be two OSF graphs. If $\gamma_0 : \mathcal{G}[g_0] \to \mathcal{G}[g_1]$ is a β_0-morphism ($\beta_0 > 0$) and $\gamma_1 : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$ is a β_1-morphism ($\beta_1 > 0$) such that $\gamma_0(g_0) = g_1$ and $\gamma_1(g_1) = g_0$, then:*

1. *for all $g \in \mathcal{G}[g_0]$, g and $\gamma_0(g)$ are labeled by the same sort;*
2. *for all $g \in \mathcal{G}[g_1]$, g and $\gamma_1(g)$ are labeled by the same sort;*
3. *$\gamma_0 \circ \gamma_1 = id_{\Delta^{\mathcal{G}[g_1]}}$ and $\gamma_1 \circ \gamma_0 = id_{\Delta^{\mathcal{G}[g_0]}}$, and thus γ_0 and γ_1 are bijections;*
4. *γ_0 and γ_1 are 1-morphisms.*

In line with this result, it is convenient to define a notion of OSF graph equivalence.

Definition 4.37 (OSF graph equivalence). Two OSF graphs g_0 and g_1 are *equivalent* (notation: $g_0 \equiv g_1$) if there are 1-morphisms $\gamma_0 : \mathcal{G}[g_0] \to \mathcal{G}[g_1]$ and $\gamma_1 : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$ such that $\gamma_0(g_0) = g_1$ and $\gamma_1(g_1) = g_0$.

Example 4.38 (OSF graph equivalence). Consider the OSF graphs g_0 and g_1 and the functions γ_0 and γ_1 from Example 3.54[8]. Similarly to how in Example 3.54 the two functions are crisp homomorphisms witnessing the equivalence of the two graphs, int the fuzzy setting γ_0 and γ_1 are easily verified to be 1-morphisms, and thus $g_0 \equiv g_1$. ▷

By Proposition 4.36 and Proposition 4.35, we can thus prove that $\sqsubseteq^{\mathcal{G}}$ is a fuzzy partial ordering on OSF graphs modulo OSF graph equivalence.

Theorem 4.39 (Endomorphic graph approximation fuzzy partial order). *The fuzzy binary relation $\sqsubseteq^{\mathcal{G}}$ is a fuzzy partial order on (equivalence classes of) OSF graphs, i.e., if $\sqsubseteq^{\mathcal{G}}(g_0, g_1) > 0$ and $\sqsubseteq^{\mathcal{G}}(g_1, g_0) > 0$, then $g_0 \equiv g_1$.*

The equivalence relation on OSF graphs induces analogous equivalence relations on normal OSF terms and rooted solved OSF clauses by letting $\psi_0 \equiv \psi_1 \Leftrightarrow G(\psi_0) \equiv G(\psi_1)$, and $\phi_0 \equiv \phi_1 \Leftrightarrow G(\phi_0) \equiv G(\phi_1)$. As expected, in every fuzzy OSF interpretation equivalent normal OSF terms have the same denotation, and equivalent rooted solved OSF clauses are satisfied with the same degree.

[8]Note that in Example 3.54 the functions γ_0 and γ_1 were defined on the domain of the crisp OSF graph algebra, which is the same as the domain of the fuzzy OSF graph algebra.

Proposition 4.40 (Fuzzy denotation of equivalent OSF terms). *If the normal OSF terms ψ_0 and ψ_1 are equivalent, then $[\![\psi_0]\!]^{\mathcal{I}} = [\![\psi_1]\!]^{\mathcal{I}}$ for every fuzzy interpretation $\mathcal{I}$.*

Proposition 4.41 (Graded satisfaction of equivalent OSF clauses). *If the rooted solved OSF clauses ϕ_0 and ϕ_1 are equivalent, then for every fuzzy interpretation $\mathcal{I}$ and every $\beta \in [0,1]$: $\mathcal{I}, \alpha_0 \models_\beta \phi_0$ for some assignment α_0 if and only if $\mathcal{I}, \alpha_1 \models_\beta \phi_1$ for some assignment α_1.*

We now give a semantic definition of fuzzy OSF term subsumption. Recall that the fuzzy sort subsumption $\preceq(\mathit{slasher}, \mathit{thriller}) = \beta$ means that, on any interpretation $\mathcal{I}$, every object that is an instance of $\mathit{slasher}^{\mathcal{I}}$ with degree β' must also be an instance of $\mathit{thriller}^{\mathcal{I}}$ with a degree greater than or equal to $\beta \wedge \beta'$. Along these lines, we may say that an OSF term t_1 is subsumed by a term t_2 with degree β if, for any interpretation $\mathcal{I}$, every object in the denotation of t_1 with degree β' also belongs to the denotation of t_2 with a degree greater than or equal to $\beta' \wedge \beta$. We thus define fuzzy OSF term subsumption as the fuzzy relation that assigns to each pair of OSF terms the supremum of all degrees $\beta \in [0,1]$ that satisfy this property.

Definition 4.42 (Semantic OSF term fuzzy subsumption). The sort subsumption relation $\preceq : \mathcal{S}^2 \to [0,1]$ is extended to a fuzzy binary relation on OSF terms by letting, for all OSF terms t_1 and t_2: $\preceq(t_1, t_2) \stackrel{\text{def}}{=} \sup(\{\beta \mid \forall \mathcal{I}, \forall d \in \Delta^{\mathcal{I}} : [\![t_1]\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![t_2]\!]^{\mathcal{I}}(d)\})$. We abbreviate $\preceq(\psi_1, \psi_2) = \beta$ by writing $\psi_1 \preceq_\beta \psi_2$.

We provide an analogous definition for the graded implication of OSF clauses, which generalizes the crisp one of Definition 3.58.

Definition 4.43 (Graded implication of OSF clauses). The *OSF clause ϕ_1 implies the OSF clause ϕ_2 at degree β* if, for all fuzzy OSF interpretations $\mathcal{I}$ and assignments α such that $\mathcal{I}, \alpha \models_{\beta_{\mathcal{I}}} \phi$, there exists an assignment α' such that: (i) $\forall X \in \mathit{Tags}(\phi_1) \cap \mathit{Tags}(\phi_2)$: $\alpha'(X) = \alpha(X)$, and (ii) $\mathcal{I}, \alpha' \models_{\beta_{\mathcal{I}} \wedge \beta} \phi_2$. The fuzzy binary relation $\models$ on OSF clauses is defined by letting $\models(\phi_1, \phi_2) \stackrel{\text{def}}{=} \sup(\{\beta \mid \phi_1 \text{ implies } \phi_2 \text{ at degree } \beta\})$. We abbreviate $\models(\phi_1, \phi_2) = \beta$ by writing $\phi_1 \models_\beta \phi_2$.

Definition 4.44 (Graded implication of rooted OSF clauses). Let ϕ_X and ϕ'_Y be two rooted OSF clauses with no common variables. *The OSF clause ϕ_X is said to imply the clause ϕ'_Y at degree β* (denoted $\phi_X \models_\beta \phi'_Y$) if $\phi \models_\beta \phi'[X/Y]$.

We now prove that the fuzzy relations $\sqsubseteq^{\mathcal{G}}$ on OSF graphs, $\preceq$ on normal OSF term and $\models$ on rooted solved OSF constraints are equivalent, generalizing the analogous crisp result (Theorem 3.60).

Theorem 4.45 (Equivalence of fuzzy OSF orderings). *If the normal OSF terms ψ and ψ' (with roots Y and X, respectively, and no common variables), the OSF graphs g and g', and the rooted solved OSF clauses ϕ_Y and ϕ'_X respectively correspond to one another though the syntactic mappings, then the following are equivalent: (1) $g \sqsubseteq^{\mathcal{G}}_{\beta} g'$, (2) $\psi' \preceq_\beta \psi$, and (3) $\phi'_X \models_\beta \phi_Y$.*

The fact that $\preceq$ and $\models$ are fuzzy partial orders on normal OSF terms and rooted solved OSF clauses (modulo OSF term and OSF clause equivalence), respectively, is obtained as a corollary of Theorems 4.39 and 4.45.

Corollary 4.46 (Fuzzy partial orders on normal OSF terms and rooted solved OSF clauses). *The fuzzy relation $\preceq$ is a fuzzy partial order on normal OSF terms modulo OSF term equivalence. The fuzzy relation $\models$ is a fuzzy partial order on rooted solved OSF clauses modulo OSF clause equivalence.*

Next, we provide a *syntactic* definition of OSF term subsumption and prove that it is equivalent to the semantic one of Definition 4.42. The syntactic definition will be useful in Section 4.7 for the computation of the subsumption degree between two OSF terms.

Definition 4.47 (Syntactic OSF term fuzzy subsumption). *The normal OSF term ψ_0 is (syntactically) subsumed by the normal OSF term ψ_1 with degree β* (denoted $\psi_0 \trianglelefteq_\beta \psi_1$) if there is a mapping $h : \mathit{Tags}(\psi_1) \to \mathit{Tags}(\psi_0)$ such that

1. $h(\mathit{RootTag}(\psi_1)) = \mathit{RootTag}(\psi_0)$;
2. if $X \xrightarrow{f}_{\psi_1} Y$, then $h(X) \xrightarrow{f}_{\psi_0} h(Y)$; and
3. $\beta = \min\{\trianglelefteq(\mathit{Sort}_{\psi_0}(h(X)), \mathit{Sort}_{\psi_1}(X)) \mid X \in \mathit{Tags}(\psi_1)\}$.

We write $\trianglelefteq(\psi_0, \psi_1) \geq \beta$ to express that $\psi_0 \trianglelefteq_{\beta'} \psi_1$ and $\beta' \geq \beta$.

Remark 7. Syntactic OSF term subsumption is well-defined. Indeed, if $h : \mathit{Tags}(\psi_1) \to \mathit{Tags}(\psi_0)$ and $h' : \mathit{Tags}(\psi_1) \to \mathit{Tags}(\psi_0)$ are two mappings that satisfy

1. $h(\mathit{RootTag}(\psi_1)) = \mathit{RootTag}(\psi_0) = h'(\mathit{RootTag}(\psi_1))$, and
2. if $X \xrightarrow{f}_{\psi_1} Y$, then $h(X) \xrightarrow{f}_{\psi_0} h(Y)$ and $h'(X) \xrightarrow{f}_{\psi_0} h'(Y)$,

then necessarily $h = h'$, which means that the value β in Definition 4.47 is unique.

Theorem 4.48 (Semantic and syntactic fuzzy subsumption). *Let ψ_0 and ψ_1 be two normal OSF terms. Then, for all $\beta \in (0, 1]$: $\psi_0 \preceq_\beta \psi_1$ if and only if there are two (normal) OSF terms ψ'_0 and ψ'_1 such that $\psi_0 \equiv \psi'_0$, $\psi_1 \equiv \psi'_1$, and $\psi'_0 \trianglelefteq_\beta \psi'_1$.*

Let $\preceq \stackrel{\text{def}}{=} |\preceq|$ and recall that $\preceq$ is a (crisp) subsumption relation on $\mathcal{S}$ such that GLBs in $(\mathcal{S}, \preceq)$ correspond to GLBs in $(\mathcal{S}, \preceq)$ (Proposition 1.42). The next theorem provides the connection between the crisp subsumption $\preceq$ and the fuzzy subsumption $\preceq$ on OSF terms.

Theorem 4.49 (Crisp and fuzzy subsumption). *For all normal OSF terms ψ_1 and ψ_2: $\psi_1 \preceq \psi_2$ if and only if $\preceq(\psi_1, \psi_2) > 0$.*

The next corollary follows from Theorem 4.49 and the analogous result for crisp OSF logic.

Corollary 4.50 (Fuzzy OSF term lattice). *The fuzzy binary relation $\preceq$ is a fuzzy lattice on (equivalence classes) of normal OSF terms.*

4.7 Fuzzy OSF term unification

In OSF logic, reasoning hinges on the unification of OSF terms, an operation that aims to combine the constraints expressed by two OSF terms ψ_1 and ψ_2 in a consistent way in a single term ψ. The resulting term – the unifier of ψ_1 and ψ_2 – is the GLB of ψ_1 and ψ_2 within the OSF term subsumption lattice. Thanks to efficient implementation techniques, OSF term unification has been integrated in logic programming languages such as LOGIN [8] and LIFE [10], and in the CEDAR Semantic Web reasoner [6, 15].

In this section we prove that computing the GLB of two OSF terms in the fuzzy subsumption lattice is no more difficult than computing it in the crisp setting, as the same unification procedure can be employed. We provide an algorithm for computing the unifier of two OSF terms and the associated unification degree, prove its correctness, and study its computational complexity.

Definition 4.51 (Fuzzy OSF term unification). The *unifier* of two normal OSF terms ψ_1 and ψ_2 is their GLB in the OSF term fuzzy subsumption lattice (modulo OSF term equivalence) and is denoted $\psi_1 \curlywedge \psi_2$. The *unification degree* of ψ_1 and ψ_2 is defined as $\min(\preceq(\psi_1 \curlywedge \psi_2, \psi_1), \preceq(\psi_1 \curlywedge \psi_2, \psi_2))$. We write $\psi = \psi_1 \curlywedge_\beta \psi_2$ if ψ is the unifier of ψ_1 and ψ_2 with unification degree β.

Recall that applying the OSF constraint normalization rules to an OSF clause results in a normal form ϕ that is either the inconsistent clause $X : \bot$, or an OSF clause in solved form together with a conjunction of equality constraints (Proposition 4.3). The subclause of ϕ in solved form is denoted $\mathit{Solved}(\phi)$. The following theorem is an immediate consequence of Theorems 4.49 and 3.65 and Proposition 1.42.

Theorem 4.52 (Fuzzy OSF term unification). *Let ψ_1 and ψ_2 be OSF terms with no common variables, and let ϕ be the OSF clause obtained by non-deterministically applying*

Algorithm 5 Compute the unifier of ψ_1 and ψ_2 and their unification degree.

Input $\psi_1, \psi_2 \in \Psi$
Output (ψ, β) such that $\psi = \psi_1 \curlywedge_\beta \psi_2$

1: **procedure** UNIFY(ψ_1, ψ_2)
2: $\quad\phi \leftarrow \phi(\psi_1)\ \&\ \phi(\psi_2)\ \&\ \mathit{RootTag}(\psi_1) \doteq \mathit{RootTag}(\psi_2)$
3: $\quad$**while** any rule r of Fig. 4.2 is applicable to ϕ **do**
4: $\quad\quad\phi \leftarrow$ result of applying r to ϕ
5: $\quad$**if** ϕ is the inconsistent clause **then**
6: $\quad\quad$**return** $(X : \bot, 1)$
7: $\quad$**else**
8: $\quad\quad\phi' \leftarrow \mathit{Solved}(\phi)$
9: $\quad\quad\mathit{Eq} = \{(X, Y) \mid X \doteq Y \text{ or } Y \doteq X \text{ is a conjunct of } \phi\}^*$
10: $\quad\quad\mathit{Tags}(\phi)_{/\mathit{Eq}} = \{[X] \mid X \in \mathit{Tags}(\phi)\}$
11: $\quad\quad$**for** $X \in \mathit{Tags}(\phi')$ **do**
12: $\quad\quad\quad\phi' \leftarrow \phi'[Z_{[X]}/X]$ ▷ $Z_{[X]}$ is a new variable in $\mathcal{V} \setminus \mathit{Tags}(\phi)$
13: $\quad\quad\psi \leftarrow \psi(\phi')$ ▷ Unifier $\psi = \psi_1 \curlywedge \psi_2$
14: $\quad\quad\beta_1 \leftarrow \min\{\preceq(\mathit{Sort}_\psi(Z_{[X]}), \mathit{Sort}_{\psi_1}(X)) \mid X \in \mathit{Tags}(\psi_1)\}$
15: $\quad\quad\beta_2 \leftarrow \min\{\preceq(\mathit{Sort}_\psi(Z_{[X]}), \mathit{Sort}_{\psi_2}(X)) \mid X \in \mathit{Tags}(\psi_2)\}$
16: $\quad\quad\beta \leftarrow \min(\beta_1, \beta_2)$ ▷ Unification degree $\beta \in [0, 1]$
17: $\quad\quad$**return** $(\psi,\ \beta)$

any applicable constraint normalization rule (Fig. 4.2) to the clause

$$\phi(\psi_1)\ \&\ \phi(\psi_2)\ \&\ \mathit{RootTag}(\psi_1) \doteq \mathit{RootTag}(\psi_2)$$

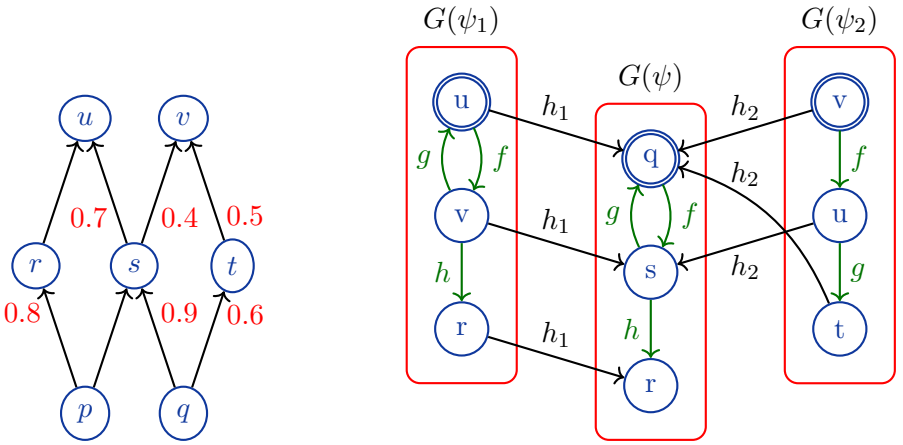
until none applies. Then, ϕ is the inconsistent clause iff the GLB of ψ_1 and ψ_2 is $X : \bot$. If ϕ is not the inconsistent clause, then $\psi_1 \curlywedge \psi_2 = \psi(\mathit{Solved}(\phi))$.

Algorithm 5 shows a procedure to unify two normal OSF terms and to compute their unification degree[9]. Given two normal OSF terms ψ_1 and ψ_2, the algorithm proceeds as follows.

- Lines 2 to 4 involve the application of the constraint normalization rules of Fig. 4.2 to the clause $\phi(\psi_1)\ \&\ \phi(\psi_2)\ \&\ \mathit{RootTag}(\psi_1) \doteq \mathit{RootTag}(\psi_2)$, resulting in an OSF clause ϕ in normal form.
- If this normal form is (equivalent to) the inconsistent clause, then $X : \bot$ is returned with unification degree 1[10] on Line 6. Otherwise, the algorithm proceeds with the computation of the unification degree.

[9] The following notation is adopted in Algorithm 5. The reflexive and transitive closure of a binary relation R is denoted R^*. The expression $\mathit{Tags}(\phi)_{/\mathit{Eq}}$ denotes the quotient of the set $\mathit{Tags}(\phi)$ modulo the equivalence relation $\mathit{Eq} \subseteq \mathit{Tags}(\phi)^2$. For a variable $X \in \mathit{Tags}(\phi)$, its equivalence class with respect to the relation Eq is $[X] \stackrel{\text{def}}{=} \{Y \mid (X, Y) \in \mathit{Eq}\}$.

[10] Recall that by Definition 1.40 $\bot$ is subsumed by every sort with degree 1.



(a) Fuzzy lattice (the elements $\bot$ and $\top$ are left implicit)

(b) The fuzzy unification of Example 4.54

Figure 4.5: The fuzzy subsumption relation and unification of Example 4.54.

- In Lines 8 to 12 the set $\mathit{Tags}(\phi)$ is partitioned into equivalence classes according to the equality constraints contained in ϕ, and each variable X in $\phi' = \mathit{Solved}(\phi)$ is uniformly renamed with a new variable $Z_{[X]} \in \mathcal{V} \setminus \mathit{Tags}(\phi)$ corresponding to its equivalence class. This step allows each term to maintain its own variable scope, and facilitates the computation of the unification degree in the next lines.

- Finally, the GLB $\psi = \psi_1 \curlywedge \psi_2$ is obtained on Line 13 as the OSF term corresponding to ϕ', while Lines 14 to 16 deal with the computation of the unification degree β of the two OSF terms.

The output of the algorithm is a pair (ψ, β) such that $\psi = \psi_1 \curlywedge_\beta \psi_2$. Note that the unifier of the two terms can already be obtained on Line 8 as $\psi(\mathit{Solved}(\phi))$, and the subsequent steps are only concerned with the computation of the unification degree. The same algorithm can also be employed to decide whether the two terms subsume each other, and to what degree: if ψ is equivalent to ψ_1, then $\preceq(\psi_1, \psi_2) = \beta$; alternatively, if ψ is equivalent to ψ_2, then $\preceq(\psi_2, \psi_1) = \beta$.

Theorem 4.53 (Correctness of Algorithm 5). *Let ψ_1 and ψ_2 be normal OSF terms. If (ψ, β) is the output of the procedure* UNIFY(ψ_1, ψ_2) *of Algorithm 5, then $\psi = \psi_1 \curlywedge_\beta \psi_2$.*

The following example clarifies each step of Algorithm 5.

Example 4.54 (Fuzzy OSF term unification). Consider the fuzzy lattice of Fig. 4.5a

and the OSF terms

$$\psi_1 = Y_0 : u\,(f \to Y_1 : v\,(g \to Y_0, h \to Y_2 : r))\,, \text{ and}$$
$$\psi_2 = X_0 : v\,(f \to X_1 : u\,(g \to X_2 : t))\,.$$

After Lines 2 to 4, an application of the rules of Fig. 4.2 to $\phi(\psi_1)$ & $\phi(\psi_2)$ & $X_0 \doteq Y_0$ yields the OSF clause in normal form

$$\begin{array}{rcllllll} \phi & = & X_0 : q & \& & X_1 : s & \& & Y_2 : r & \& \\ & & X_0.f \doteq X_1 & \& & X_1.g \doteq X_0 & \& & X_1.h = Y_2 & \& \\ & & X_0 \doteq Y_0 & \& & X_0 \doteq X_2 & \& & X_1 \doteq Y_1 & \end{array}$$

or an equivalent clause. On Lines 9 and 10 the set $\mathit{Tags}(\phi)$ is thus partitioned into the equivalence classes $[X_0] = \{X_0, X_2, Y_0\}$, $[X_1] = \{X_1, Y_1\}$ and $[Y_2] = \{Y_2\}$. The solved part of ϕ is renamed accordingly (Lines 11 and 12) and translated on Line 13 into the OSF term

$$\psi = Z_{[X_0]} : q(f \to Z_{[X_1]} : s(g \to Z_{[X_0]}, h \to Z_{[Y_2]} : r)).$$

The subsumption degree $\preceq(\psi, \psi_1)$ is then computed on Line 14 as $\beta_1 = \min\{\preceq(q,u), \preceq(s,v), \preceq(r,r)\} = 0.4$. In particular, $\preceq(\mathit{Sort}_\psi(Z_{[X_1]}), \mathit{Sort}_{\psi_1}(Y_1)) = \preceq(s,v) = 0.4$. Similarly, $\preceq(\psi, \psi_2)$ is computed on Line 15 as $\beta_2 = \min\{\preceq(q,v), \preceq(s,u), \preceq(q,t)\} = 0.5$. Overall, the unification degree of ψ_1 and ψ_2 is thus 0.4 (Line 16). For $i \in \{1,2\}$, the mapping $h_i : \mathit{Tags}(\psi_i) \to \mathit{Tags}(\psi)$ witnessing the (syntactic) subsumption $\psi \trianglelefteq_{\beta_i} \psi_i$ is defined by letting $h_i(X) = Z_{[X]}$ for all $X \in \mathit{Tags}(\psi_i)$. The unification is depicted in Fig. 4.5b, where the mappings h_1 and h_2 are depicted as arrows relating the nodes of $G(\psi_1)$, $G(\psi_2)$ and $G(\psi)$ corresponding to the variables of their respective OSF terms. ▷

Finally, let us analyze Algorithm 5 in terms of complexity. Finding the unifier of two normal OSF terms ψ_1 and ψ_2 with respect to a fuzzy sort subsumption lattice has the same complexity of deciding the problem in the crisp setting. This is due to the fact that GLBs in a fuzzy lattice and its crisp counterpart can be computed in the same way (also see Proposition 1.42), and the rules for fuzzy OSF constraint normalization and crisp OSF constraint normalization are essentially the same. The algorithm from [8] is based on the union-find problem [1] and has a worst-case complexity of $O(mG(m))$, where $m = |\mathit{Tags}(\psi_1) \cup \mathit{Tags}(\psi_2)|$ and the growth rate of the function G is of the order of an inverse of the Ackermann function ($G(m) \leq 5$ for all practical purposes) [8].

Partitioning the set $\mathit{Tags}(\phi)$ according to the equality constraints in ϕ (Lines 9 and 10) is an application of the union-find problem. The complexity is thus $O(mG(m))$, where G is as above and $m = |\mathit{Tags}(\psi_1) \cup \mathit{Tags}(\psi_2)|$.

Computing $\preceq(s, s')$ for two sorts $s, s' \in \mathcal{S}$ can be performed in $O(|\mathcal{S}|+e)$ time – where e is

the number of edges in the DAG representation of the fuzzy sort subsumption relation – with an approach that is analogous to solving the shortest paths problem in a DAG (presented in Section 4.8.3). The overall complexity of the computation of the subsumption degrees β_1 and β_2 in Lines 14 and 15 is thus $O(m(|\mathcal{S}| + e))$, where $m = |\mathit{Tags}(\psi_1) \cup \mathit{Tags}(\psi_2)|$.

4.8 Implementation

As discussed in Section 3.8, efficient OSF term unification relies on an efficient implementation of the GLB operation in the sort subsumption lattice. This can be achieved, for instance, by encoding the DAG representation of the sort lattice using binary vectors. In this section we show that the same encoding techniques of crisp OSF logic can be employed for computing GLBs in a fuzzy lattice, and we present algorithms for the computation of the subsumption degree $\preceq(s, s')$ between two sorts.

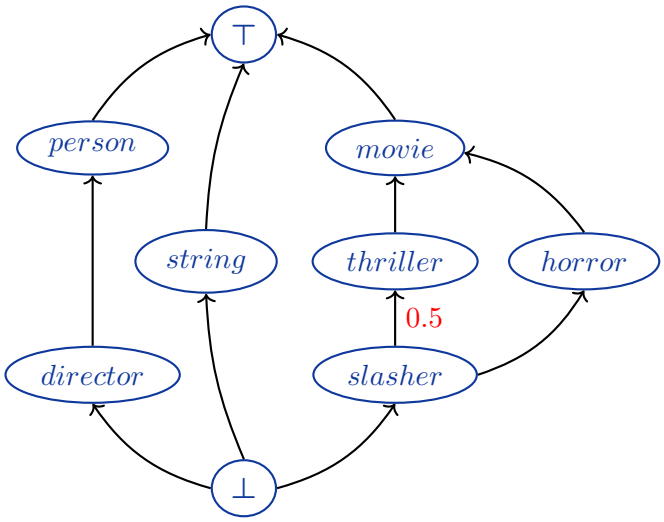
4.8.1 Encoding fuzzy lattices

A fuzzy bounded lattice $(\mathcal{S}, \preceq)$ of sort symbols can be obtained as the reflexive and transitive closure of fuzzy is-a declarations of shape $s \lhd_\beta s'$ ($\beta \in (0, 1]$), i.e., $\preceq \stackrel{\text{def}}{=} \lhd^{\circledast}$, where $\lhd : \mathcal{S}^2 \to [0, 1]$ is a fuzzy binary relation on $\mathcal{S}$. Because of Proposition 1.42 and the following proposition – stating that the support of the reflexive and transitive closure of a fuzzy binary relation is equal to the reflexive and transitive closure of its support –, GLBs in $(\mathcal{S}, \preceq)$ can equivalently be computed in $(\mathcal{S}, \preceq)$, where $\preceq \stackrel{\text{def}}{=} |\preceq|$.

Proposition 4.55 (Support and reflexive-transitive closure). *If $\lhd$ is a fuzzy binary relation on a set $\mathcal{S}$, then $|\lhd|^* = |\lhd^{\circledast}|$.*

By Propositions 1.42 and 4.55, it is possible to consider a fuzzy subsort declaration $s \lhd_\beta s'$ with $\beta > 0$ as a regular subsort declaration $s \,|\lhd|\, s'$ simply by ignoring the associated degree β. Algorithm 1 or Algorithm 2 can then be used to encode the (crisp) lattice $(\mathcal{S}, |\lhd|^*)$ by associating a binary code with each sort, and these codes can be used to compute GLBs with the encoding technique seen in Section 3.8.1.

Example 4.56 (Encoding a fuzzy lattice). Fig. 4.6b shows the encoding of the sorts of the fuzzy lattice represented in Fig. 4.6a, obtained by applying the same classification algorithm of Algorithm 1 or Algorithm 2, simply by ignoring the edge weights. The codes are computed as bit-vectors of variable length. Computing the GLB of *thriller* and *horror*, for instance, amounts to performing the bitwise AND of their codes, i.e., 100010AND10010 = 10. The result is the code of the sort *slasher*, which is indeed the GLB of *thriller* and *horror* in the lattice of Fig. 4.6b. ▷



(a)

Sort	Code
⊥	0
director	1
slasher	10
string	100
person	1001
thriller	10010
horror	100010
movie	1110110
⊤	11111111

(b)

Figure 4.6: A fuzzy lattice and its encoding.

4.8.2 Encoding fuzzy partial orders

The definition of an OSF signature (Definition 3.1) requires that the sort subsumption relation of crisp OSF logic must constitute a finite lattice. As seen in Section 3.8.2, however, whenever $(\mathcal{S}, \preceq)$ is just a partial order it is still possible to implicitly work on a *lattice completion* of $(\mathcal{S}, \preceq)$ defined on its antichains. From an implementation perspective, the completion does not need to be explicitly constructed. When the GLB $s \wedge s'$ of two sorts does not exists, the set of their MLBs is computed instead (for instance, through Algorithm 3), since $\{s, s'\}^{mlb}_{\preceq}$ corresponds exactly to the GLB of $\{s\}$ and $\{s'\}$ in the lattice of antichains of $\mathcal{S}$.

In this section we show that it is possible to adopt a similar strategy in fuzzy OSF logic when the fuzzy subsumption relation $(\mathcal{S}, \preceq)$ is not a fuzzy lattice, but only a fuzzy partial order. Like in the crisp case, rather than computing the GLB of two sorts of a fuzzy poset, it is possible to compute the set of their MLBs instead.

Definition 4.57 (MLBs in a fuzzy poset). Let $(\mathcal{S}, \preceq)$ be a fuzzy poset and $S \subseteq \mathcal{S}$. *The set of MLBs of* S *in* $(\mathcal{S}, \preceq)$ is defined as $S^{fmlb}_{\preceq} \stackrel{\text{def}}{=} \left\lceil S^{fl}_{\preceq} \right\rceil^{\cdot}$.

The fact that the same encoding and decoding strategy of Section 3.8.2 can be applied in the fuzzy case is guaranteed by Proposition 4.55 and the following proposition, which states that the MLBs in a fuzzy poset $(\mathcal{S}, \preceq)$ correspond to the MLBs in the poset $(\mathcal{S}, |\preceq|)$.

Proposition 4.58 (MLBs in crisp and fuzzy posets). *Let* $(\mathcal{S}, \preceq)$ *be a fuzzy poset and let* $\preceq \stackrel{\text{def}}{=} |\preceq|$. *For all* $S \subseteq \mathcal{S}$, $S^{fmlb}_{\preceq} = S^{mlb}_{\preceq}$.

This approach is formally validated by the construction of the completion of a fuzzy

partial order $(\mathcal{S}, \preceq)$ into a fuzzy lattice $(Antichains(\mathcal{S}), \preceq)$ built on the antichains of $(\mathcal{S}, \preceq)$. Like in the crisp case, the GLB of $\{s\}$ and $\{s'\}$ in $(Antichains(\mathcal{S}), \preceq)$ corresponds exactly to the set of maximal lower bounds of $\{s, s'\}$ in $(\mathcal{S}, \preceq)$. The construction of $(Antichains(\mathcal{S}), \preceq)$ is detailed in Section 5.4, in the context of the definition of similarity-based reasoning with OSF logic.

4.8.3 Computing the subsumption degree $\preceq(s, s')$

Besides finding the GLB of two sorts (or the set of their MLBs), another operation that needs to be performed efficiently is the computation of the subsumption degree $\preceq(s_0, s_1)$.

A possible approach is to generalize the bit-vector encoding strategy of Section 3.8.1 by encoding each sort $s \in \mathcal{S}$ with a vector of $N = |\mathcal{S}|$ real numbers rather than N bits, in such a way that the real number at position i in the vector encoding the sort s is the value of the subsumption degree $\preceq(s_i, s)$. In other words, the code $\varepsilon(s)$ of a sort s would be a vector $V \in [0,1]^N$ that corresponds to the fuzzy set $\{\beta/s_i \mid V[i] = \beta = \preceq(s_i, s)\}$. This encoding can be obtained by adapting the crisp classification algorithm (Algorithm 1) using the point-wise max and min operations on real vectors, rather than the bitwise OR and AND operations on bit vectors. The GLB of a pair s_0 and s_1 can be obtained by taking the point-wise min of their codes, thus obtaining a code corresponding to the set $L = \{\beta/s \mid \beta = \min(\preceq(s, s_0), \preceq(s, s_1))\}$, and then computing the maximal elements of $|L|$. This approach may however be too expensive for large fuzzy lattices. For instance, while encoding a lattice of $N = 100000$ sorts with binary codes of variable length would require around 596 MB, encoding a fuzzy lattice using 32-bit or 64-bit representations of real numbers in the unit interval would require around 38146 MB or 76293 MB, respectively. Moreover, although the binary encoding technique has been applied to a DAG of more than 900000 nodes in the CEDAR project, for even larger graphs in order to mitigate the space complexity it may be necessary to apply the optimizations presented in [13] or [6], which are not readily applicable to $[0,1]$-valued arrays.

An alternative solution is to compute the value of $\preceq(s_0, s_1)$ only when necessary (and possibly keep a cache for efficient subsequent computations), by traversing the DAG representation of the fuzzy lattice. This computation can be optimized by first encoding the fuzzy lattice, so that it is possible to efficiently decide whether a sort s is subsumed by another sort s' with a positive degree, by verifying whether $s \curlywedge s' = s$, allowing to avoid traversing portions of the DAG representation that are not needed for the computation of $\preceq(s_0, s_1)$. While encoding the fuzzy lattice with bit vectors still requires quadratic space, storing vectors of bits requires significant less space than an $|\mathcal{S}| \times |\mathcal{S}|$ table of real numbers.

Recall that a fuzzy lattice $(\mathcal{S}, \preceq)$ can be represented as a weighted DAG obtained by fuzzy is-a declarations of shape $s \lhd_\beta s'$ ($\beta \in (0,1]$), i.e., $\preceq \stackrel{\text{def}}{=} \lhd^{\circledast}$, where $\lhd : \mathcal{S}^2 \to [0,1]$ is a fuzzy binary relation on $\mathcal{S}$. The set of all paths from s to s' in this DAG is denoted

$\mathit{Paths}(s, s')$. The *value* of a path $\pi = s'_1 \lhd_{\beta_1} s'_2 \lhd_{\beta_2} \ldots \lhd_{\beta_{n-2}} s'_{n-1} \lhd_{\beta_{n-1}} s'_n$ is defined as $\mathit{Value}(\pi) \stackrel{\text{def}}{=} \min_{1 \leq i \leq n-1} \beta_i$. By max-min transitivity, the value of $\preceq(s_0, s_1)$ can thus be computed as $\max_{\pi \in \mathit{Paths}(s_0, s_1)} \mathit{Value}(\pi)$.

Concretely, the computation of $\preceq(s_0, s_0)$ can be achieved with Algorithm 6, which proceeds analogously to the shortest-paths algorithm for weighted DAGs, by visiting all the elements between s_0 and s_1 in topological order. For a sort $s \in \mathcal{S}$ we let $\mathit{Parents}^f(s) \stackrel{\text{def}}{=} \{\beta / s' \mid s \lhd_\beta s'\}$, and we assume that the procedure $\textsc{Topological}(s_0, s_1)$ returns an iterator of sorts between s_0 and s_1 in a fixed topological order of $(\mathcal{S}, \preceq)$. During the traversal, the value of the best path from s_0 to a sort s' found so far is stored as $\mathit{Value}(s')$. This value is initially undefined for every sort. The algorithm proceeds as follows.

1. First of all, the algorithm verifies whether $\preceq(s_0, s_1) > 0$ on Line 2, which can be achieved efficiently by encoding the weighted DAG representation of the fuzzy lattice (Section 4.8.1). If $s_0 \not\preceq s_1$, the algorithm returns the subsumption degree 0; otherwise, it continues with the traversal of the DAG.

2. The value $\mathit{Value}(s_0)$ of the source node s_0 is initialized to 1 on Line 5, since $\preceq(s_0, s_0) = 1$ by reflexivity. The algorithm then proceeds by visiting each sort s' in topological order from s_0 to s_1.

3. On Line 7 the algorithm verifies whether $\mathit{Value}(s')$ has been initialized. It may happen that s' is not a supersort of s', but still happens to be between s_0 and s_1 in the topological ordering, so that $\mathit{Value}(s')$ is left undefined. In this case the sort s' is skipped, and the loop of Line 6 resumes with the next sort.

4. Otherwise, if $\mathit{Value}(s')$ is defined, each parent s'' of s' is considered (Line 8). If s'' is not a subsort of s_1, then s'' is skipped, and its value $\mathit{Value}(s'')$ is not computed. Otherwise, $\beta = \min(\mathit{Value}(s'), \lhd(s', s''))$ is computed on Line 10. Finally, $\mathit{Value}(s'')$ is either initialized or updated to the value β, depending on whether it was undefined, or its value was less than β.

Because the nodes are visited in topological order, after the node s_1 has been visited it must be the case that $\mathit{Value}(s_1) = \preceq(s_0, s_1) = \max_{\pi \in \mathit{Paths}(s_0, s_1)} \mathit{Value}(\pi)$.

Encoding the DAG representation of the fuzzy lattice allows a few optimizations in Algorithm 6, by allowing to skip sorts that are not necessary for the computation of $\preceq(s_0, s_1)$. For instance, on Line 9 the sort s'' is skipped if it is not a subsort of s_1, so that $\mathit{Value}(s'')$ is not computed. At a later iteration of the loop of Line 6, the same sort s'' will be skipped on Line 7 since $\mathit{Value}(s'')$ is undefined, avoiding unnecessary computations on Lines 8 to 12.

Example 4.59 (Computing a subsumption degree). Consider the fuzzy partial order represented in Fig. 4.7, where the index of each sort corresponds to its position in a

Algorithm 6 Approximation degree in topological order.

```
1:  procedure DEGREE(s_0, s_1)                          ▷ Compute ⪯(s_0, s_1)
2:    if s_0 ⋠ s_1 then
3:      return 0
4:    else
5:      Value(s_0) ← 1
6:      for s' ← s_0 to s_1 in TOPOLOGICAL(s_0, s_1) do
7:        if Value(s') is defined then
8:          for β/s'' ∈ Parents^f(s') do
9:            if s'' ⪯ s_1 then
10:             β ← min(Value(s'), β)
11:             if Value(s'') is not defined or Value(s'') < β then
12:               Value(s'') ← β
13:     return Value(s_1)
```

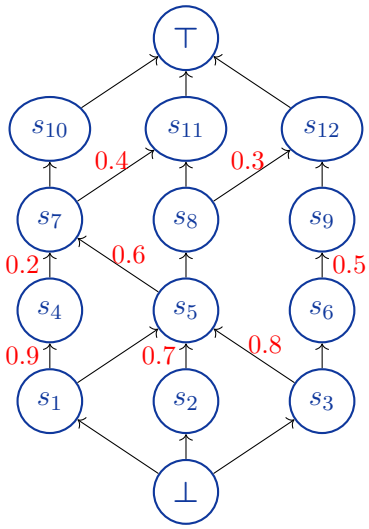


Figure 4.7: The fuzzy partial order of Example 4.59.

topological ordering of the graph. In order to compute $\preceq(s_3, s_9)$, the iteration starting on Line 6 will first consider the sort s_3 and initialize $\mathit{Value}(s_6) = 1$, but not $\mathit{Value}(s_5)$, since the latter is not a subsort of s_9. After discarding the sorts s_4 and s_5 on Line 7, the iteration will visit s_6 and initialize $\mathit{Value}(s_9) = 0.5$. The sorts s_7 and s_8 are then skipped on Line 7, and the next node visited by the iteration is the target s_9, where the iteration stops. The value of $\preceq(s_3, s_9)$ is thus 0.5. Without the optimizations implemented in Algorithm 6, $\mathit{Value}(s_5)$ would have been initialized instead, causing the loop of Line 6 to be executed for s_5, and consequently also for s_7 and s_8, thus initializing $\mathit{Value}(s_i)$ for $i \in \{5, 7, 8, 10, 11, 12\}$, traversing edges of the fuzzy partial order that are not necessary for the computation of $\preceq(s_3, s_9)$. ▷

```
% Subsumption and fuzzy subsumption
rock < music.
electronic < music.
post_rock < rock.
artist < person.
<(post_rock, electronic) = 0.5.

% Instances
{ alinda } < person.
{ yorke, greenwood } < artist.
{ radiohead } < band.
{ kid_a } < post_rock.
{ anima } < electronic.
{ 0.7/ok_computer } < rock.

% Facts
member_of(yorke, radiohead).
member_of(greenwood, radiohead).
plays_in(yorke, kid_a(year -> 2000)).
plays_in(yorke, anima(title -> "Anima")).
plays_in(greenwood, ok_computer).

% Rules
likes(alinda, X) :- plays_in(Z : artist, X : electronic),
                    member_of(Z, radiohead).
```

Figure 4.8: A logic program with OSF terms and a fuzzy subsumption relation.

4.9 Potential application: logic programming with fuzzy OSF logic

A possible application of fuzzy OSF logic is the implementation of a logic programming language (like LOGIN [8]) based on SLD resolution and fuzzy OSF term unification, i.e., where first-order term unification is replaced by the unification of OSF terms over a fuzzy sort subsumption lattice. As seen in Section 2.2, logic programming languages and weaker definitions of unification based on fuzzy relations such as similarities and proximities have been researched extensively (e.g., [9, 17, 51, 61, 64, 72, 74, 101]) and implemented in systems such as Bousi~Prolog [63] and FASILL [60].

There are several potential advantages to using OSF logic in this context. First of all, the unification algorithm for OSF terms takes into account a (fuzzy) sort subsumption relation, which can result in more efficient computations [8, 37]. Another advantage is the flexibility provided by OSF terms, which lack a fixed arity and can thus easily represent

partial information, and are moreover simpler to interpret thanks to their use of features rather than positions to specify arguments [8].

Consider, for example, the program of Fig. 4.8. The program begins by declaring a simple fuzzy subsumption relation (whose only non-crisp subsumption is the pair $\preceq$(`post_rock`, `electronic`) = 0.5), followed by the membership degrees of a few individuals (where the only non-crisp membership concerns `ok_computer`, which belongs to the sort `rock` with degree 0.7). Individuals can be thought of as singleton sorts (sorts that denote a single element), and thus as OSF terms themselves. The predicates `member_of`, `plays_in` and `likes` – whose arguments are OSF terms – are then used to specify the facts of the program and a rule stating that the individual `alinda` likes electronic music albums played by members of the band Radiohead. The query `?- likes(alinda, X : music)` would return a solution binding `Z` to `yorke` and `X` to `anima(title -> "Anima")`, and another solution binding `Z` to `yorke` and `X` to `kid_a(year -> 2000)`. However, since `kid_a` is an instance of `post_rock`, which is subsumed by `electronic` only with degree 0.5, the second solution is associated with a satisfaction degree of 0.5. As another example, the query `?- plays_in(greenwood, X : rock)` would return the solution mapping `X` to `ok_computer` with satisfaction degree 0.7, as this is the membership degree of `ok_computer` to `rock`.

Additional applications of fuzzy OSF logic will be explored in Chapter 5, focusing on similarity-based OSF logic, an approach grounded on fuzzy OSF logic.

Chapter 5

Similarity-Based Reasoning with Order-Sorted Feature Logic

Many approaches to approximate reasoning based on fuzzy relations have been proposed in the logic programming literature. As seen in Section 2.2, several approaches extend the syntax of a logic program with a similarity relation (or a proximity relation, or multiple such relations) on the set of functor and predicate symbols. This relation is then taken into account when two first-order terms (FOTs) are unified or when SLD resolution is performed, so as to tolerate mismatches in functor and predicate names, leading to approximate solutions to queries posed to a logic program.

As preliminary work towards the definition of similarity-based reasoning with Order-Sorted Feature (OSF) logic, the FOT unification procedure of Aït-Kaci and Pasi [9] not only tolerates mismatches in functor names, but also allows the unification of FOTs with a different number and possibly a different order of arguments. This approach allows the unification, for example, of FOTs such as `movie(Director, "Psycho")` and `film(Title, Year, "Hitchcock")`, provided that the functors `movie` and `film` are considered similar, and that a mapping between their argument positions is defined.

The interest in defining a similarity-based extension of OSF logic lies in the efficiency of this language for knowledge representation and reasoning. For instance, as shown in Section 3.7.1, one advantage of OSF logic is that its unification algorithm takes into account a *subsumption* (is-a) ordering between sorts, which enables a single unification step to potentially replace several resolution steps, possibly leading to more efficient computations [8, 37]. Moreover, as discussed in Section 3.7.2, the CEDAR Semantic Web reasoner based on OSF logic was, at the time of the experiments, consistently among the best reasoners in terms of concept classification, and several orders of magnitude more efficient in terms of terminological reasoning [6, 15]. Introducing a similarity relation to augment the flexibility of OSF logic, while preserving its efficiency, has the potential to significantly enhance the

effectiveness and applicability of this framework, particularly in domains like the Semantic Web, where efficiency is of paramount concern.

In this chapter we show how to make the OSF term unification algorithm more flexible by considering a *similarity relation between sorts* besides a sort subsumption ordering. Rather than devising ad-hoc unification rules that deal with the similarity relation and its interaction with the sort subsumption ordering, we propose to *combine the two relations into a single fuzzy subsumption relation*. Intuitively, this is achieved by applying the following informal inference, inspired by the similarity-based approaches to logic programming (e.g., [101]):

$$\frac{\begin{array}{l}\text{If the sort } s_0 \text{ is subsumed by the sort } s_1\\ \text{and } s_1 \text{ is similar to the sort } s_2 \text{ with degree } \beta\end{array}}{\text{then } s_0 \text{ is subsumed by } s_2 \text{ with degree } \beta.}$$

For example, if *slasher*s are *horror* movies, and *horror* movies are similar to *thriller*s with degree 0.5, then we can conclude that *slasher*s are also *thriller*s with subsumption degree 0.5. As a consequence, queries aimed to retrieve *thriller*s from a knowledge base may also retrieve instances of *slasher* (associated with an approximation degree), thus improving the flexibility of the retrieval process. Intuitively, the fuzzy subsumption relation encodes the information of both the crisp subsumption and the similarity. This procedure shifts the setting to that of *OSF logic with a fuzzy sort subsumption relation*, i.e., *fuzzy OSF logic*, whose semantics has been developed in Chapter 4. The advantage is that, as seen in Chapter 4, in fuzzy OSF logic it is possible to apply the same unification algorithm of (crisp) OSF logic, with essentially the same computational cost, thereby retaining the efficiency inherent to this logical framework.

In Section 5.1 we argue why a similarity-based unification procedure for OSF terms should take into account the interaction between the sort subsumption and the sort similarity relations, and why combining them into a single fuzzy subsumption relation is an effective way to address this issue.

We proceed by dealing with the issue of how to formally define a fuzzy subsumption relation – which should be a fuzzy partial order, or even a fuzzy lattice – starting from a (crisp) subsumption and a fuzzy similarity relation. Section 5.2 starts by combining the two relations into a *fuzzy subsumption preorder*, which may however contain fuzzy subsumption cycles. This matter is addressed in Section 5.3, where we present a construction of a fuzzy partial order from a fuzzy preorder that generalizes the well-known analogous result from order theory (Proposition 1.20). In Section 5.4 we introduce a definition of a completion of a fuzzy poset into a fuzzy lattice built on the antichains of the fuzzy poset, generalizing Proposition 1.17 to a fuzzy setting. The completion finalizes the transformation of a subsumption relation and a similarity into a fuzzy subsumption relation. In other words, Sections 5.2 to 5.4 ensure that the process of combining a crisp subsumption and a fuzzy

similarity according to the intuition outlined above is sound, i.e., it indeed leads to a fuzzy lattice, as required by the unification rules of fuzzy OSF logic.

Finally, in Section 5.5 we discuss two potential applications of similarity-based reasoning with OSF logic: (i) an extension of the CEDAR reasoner [6, 15] which relies on a sort similarity relation to approximately answer queries posed to a knowledge base, and (ii) a fuzzy logic programming language based on OSF terms, which leverages a similarity relation between sorts, comparable to a similarity-based extension of the language LOGIN [8].

The proofs of the main results of this chapter are reported in Appendix A.2.

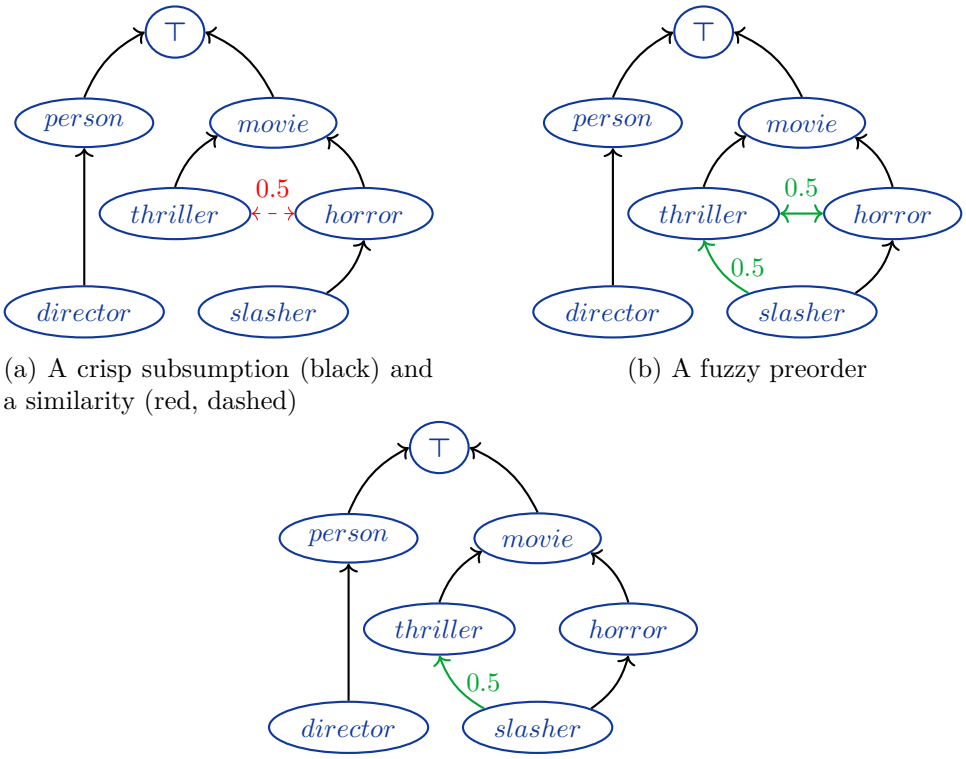
5.1 Similarity-based OSF logic, informally

In a standard setting, two FOTs can be unified if there is a substitution that makes them *equal*. A mismatch of functor symbols such as f and g in the terms $t_1 \stackrel{\text{def}}{=} f(X, a)$ and $t_2 \stackrel{\text{def}}{=} g(b, Y)$ causes their unification to fail. As discussed in Section 2.2, several approaches have been proposed to relax this equality constraint in order to make FOT unification more flexible. For instance, by considering a similarity relation between functor symbols (e.g., [101]), it is possible to *weakly unify* the terms t_1 and t_2 provided that $f \sim_\beta g$ with $\beta \in (0, 1]$. In this case the two terms *weakly unify with approximation degree* β. A similarity can also be considered when resolving clauses, leading to similarity-based SLD resolution [101] (see, e.g., Examples 2.6 and 2.7).

As seen in Section 3.6, the unification of two OSF terms aims to combine the constraints expressed by the two terms in a consistent way into a single term. This procedure takes a sort subsumption relation $\preceq$ into account, so that it is possible, for instance, to unify the term $t_1 \stackrel{\text{def}}{=} s(f_1 \to s_1)$ with the term $t_2 \stackrel{\text{def}}{=} s'(f_2 \to s_2)$ even if their root sorts s and s' are different, as long as $s \wedge s' \neq \bot$. Moreover, as shown in Section 3.7.1, this feature can allow a single unification step to replace several resolution steps, which can lead to more efficient computations.

Our goal is to define a more flexible, or *weaker*, unification procedure for OSF terms that also takes into account a *similarity relation* $\sim\colon \mathcal{S} \times \mathcal{S} \to [0, 1]$ on sort symbols. Using this additional information it could be possible, for example, to unify the terms t_1 and t_2 even if $s \wedge s' = \bot$.

Example 5.1 (Weak OSF term unification: similar sorts). Consider the set of sorts $\mathcal{S}$, the subsumption relation $\preceq \subseteq \mathcal{S}^2$, and the similarity $\sim : \mathcal{S}^2 \to [0, 1]$ specified in Fig. 5.1a. Additionally, consider the sort `"Psycho"`, which only subsumes $\bot$ and is only subsumed by $\top$, and the OSF terms $\psi_1 \stackrel{\text{def}}{=} \mathit{horror}(\mathit{title} \to \texttt{"Psycho"})$ and $\psi_2 \stackrel{\text{def}}{=} \mathit{thriller}(\mathit{title} \to X : \top)$. Clearly the two terms cannot be unified since $\mathit{horror} \wedge \mathit{thriller} = \bot$, but they could be *weakly unified* if we additionally consider that *horror* and *thriller* are similar to degree 0.5. Intuitively, as ψ_1 and ψ_2 denote movies of similar genres (and all of the other constraints



(a) A crisp subsumption (black) and a similarity (red, dashed)

(b) A fuzzy preorder

(c) A fuzzy lattice

Figure 5.1: Examples of (fuzzy) relations (the bottom element $\bot$ is left implicit).

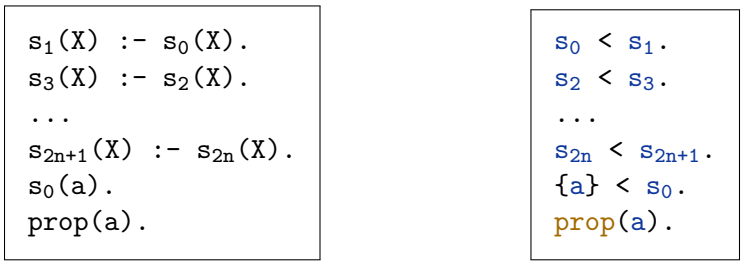
can be combined in a consistent way), then they could be weakly unified. ▷

Besides matching similar sorts, weak OSF term unification could also consider the interaction between the subsumption and the similarity relations, as the following example shows.

Example 5.2 (Weak OSF term unification: subsort of a similar sort). Continuing from Example 5.1, consider the term $\psi_3 \stackrel{\text{def}}{=} \mathit{slasher}(\mathit{title} \to \texttt{"Psycho"})$ which unifies with ψ_1 since $\mathit{slasher} \preceq \mathit{horror}$. Moreover, as $\mathit{horror} \sim_{0.5} \mathit{thriller}$, then ψ_3 should also *weakly unify* with ψ_2. In other words, since the genre of ψ_3 is subsumed by a genre (horror) which is similar to the genre of ψ_2 (and also all of the other constraints can be combined in a consistent way), then the two terms could be *weakly* unified, achieving a result analogous to Example 2.6, where similarity-based SLD resolution is employed. ▷

Additionally, similarly to how a single OSF term unification step can replace several SLD resolution steps (as in Example 3.67), similarity-based OSF term unification should be capable of replacing several similarity-based SLD resolution steps.

Example 5.3 (Similarity-based SLD resolution and weak OSF term unification). Consider the logic program of Fig. 5.2a and assume that $\sim$ is a similarity relation such



(a) Prolog version (b) OSF version

Figure 5.2: The logic programs of Example 5.3.

that, for each $1 \leq i \leq n$, $\sim(\mathtt{s_{2i-1}}, \mathtt{s_{2i}}) = \beta_i > 0$. The query `?- s2n+1(X), prop(X)` will then require several similarity-based SLD resolution steps before returning the solution `X = a`. Now consider the OSF version of the same program of Fig. 5.2b and the query `prop(X:s2n+1)`. Because `a`[1] is subsumed by `s0`, which is subsumed by `s1`, which is similar to `s2`, which is subsumed by `s3`, and so on, up to `s2n+1`, then the query should succeed after a single unification of the terms `X:s2n+1` and `a`. ▷

These examples show that a similarity-based notion of OSF term unification should consider *the interaction between the sort subsumption and the sort similarity relations*, besides allowing sorts that are equal, similar, or with a non-bottom GLB.

In order to achieve this, we propose to combine the (crisp) subsumption $\preceq$ and the similarity $\sim$ into a single fuzzy subsumption relation $\precsim$. The advantage is that it would then be possible to employ the fuzzy unification algorithm of OSF terms (Section 4.7), taking into account the fuzzy subsumption $\precsim$ which incorporates the information of both $\preceq$ and $\sim$ at the same time. Intuitively, the combination of $\preceq$ and $\sim$ is inspired by the following informal inference:

$$\frac{\begin{array}{l}\text{If the sort } s_0 \text{ is subsumed by the sort } s_1\\ \text{and } s_1 \text{ is similar to the sort } s_2 \text{ with degree } \beta\end{array}}{\text{then } s_0 \text{ is subsumed by } s_2 \text{ with degree } \beta.}$$

For example, considering the setting of Examples 5.1 and 5.2, since $\mathit{slasher} \preceq \mathit{horror}$ and $\mathit{horror} \sim_{0.5} \mathit{thriller}$ (Fig. 5.1a), then we could conclude that $\precsim(\mathit{slasher}, \mathit{thriller}) = 0.5$ (Fig. 5.1c). Thus, it would be possible to unify the terms ψ_1 and ψ_2, since, according to Fig. 5.1c, $\mathit{horror} \curlywedge \mathit{thriller} = \mathit{slasher}$ (where $\curlywedge$ denotes the GLB of two sorts in the fuzzy subsumption $\precsim$). The unification would result in the term $\mathit{slasher}(\mathit{title} \rightarrow \texttt{"Psycho"})$, which is subsumed by ψ_1 with degree 1, and by ψ_2 with degree 0.5 (and thus the overall unification degree is 0.5). In a similar manner, ψ_2 and ψ_3 can be unified with degree 0.5.

[1] Recall that constants such as `a` are treated as singleton sorts, i.e., sorts that denote a single element.

Analogously, in the context of Example 5.3, since

$$\mathtt{a} \preceq \mathtt{s}_0,\ \ \mathtt{s}_0 \preceq \mathtt{s}_1,\ \ \sim(\mathtt{s}_1, \mathtt{s}_2) = \beta_1,\ \ \mathtt{s}_2 \preceq \mathtt{s}_3,\ \ \ldots,\ \ \sim(\mathtt{s}_{2n-1}, \mathtt{s}_{2n}) = \beta_n,\ \ \mathtt{s}_{2n} \preceq \mathtt{s}_{2n+1}$$

then by repeatedly applying the same informal inference[2] we obtain $\preceq(\mathtt{a}, \mathtt{s}_{2n+1}) = \beta = \min_{1 \leq i \leq n} \beta_i$, thus enabling the weak unification of $\mathtt{a}$ with $\mathtt{X}{:}\mathtt{s}_{2n+1}$ with degree β.

The advantage of this approach is that it allows to seamlessly integrate the similarity relation into the unification rules of fuzzy OSF logic, taking the interaction between the similarity and the crisp subsumption into account, potentially allowing a single weak unification to replace several similarity-based SLD resolution steps, all the while maintaining the same computational complexity of crisp OSF logic for the computation of the unifier of two OSF terms. The only additional cost is the construction of the fuzzy subsumption, which is only required once, before the queries are processed.

Sections 5.2 to 5.4 are devoted to the formal validation of this procedure. Specifically, they demonstrate that the fuzzy subsumption defined by combining a crisp subsumption and a similarity leads indeed to a fuzzy lattice, as required by the unification rules of fuzzy OSF logic.

5.2 Fuzzy similarity-subsumption preorder

As a first step towards our formal definition of a fuzzy subsumption relation that combines a (crisp) subsumption relation and a sort similarity, we define a *fuzzy preorder* on sort symbols, which intuitively arises from the iterated application of the informal inference discussed above.

Definition 5.4 (Similarity-subsumption chain and $\precsim$). Let $(\mathcal{S}, \preceq)$ be a partial order, let $\sim\colon \mathcal{S}^2 \to [0,1]$ be a similarity relation, and let $s, s' \in \mathcal{S}$. A *similarity-subsumption chain of strength* β from s to s' is a sequence of sorts

$$s \sim_{\beta_0} s_0 \preceq s'_1 \sim_{\beta_1} s_1 \preceq s'_2 \cdots s_{n-1} \preceq s'_n \sim_{\beta_n} s'$$

such that $\beta = \min_{0 \leq i \leq n} \beta_n$. The fuzzy relation $\precsim$ on $\mathcal{S}$ is defined by letting, for all $s, s' \in \mathcal{S}$,

$$\precsim(s, s') \stackrel{\text{def}}{=} \max \left\{ \beta \;\middle|\; \begin{array}{l} \text{there is a similarity-subsumption} \\ \text{chain of strength } \beta \text{ from } s \text{ to } s' \end{array} \right\}.$$

Alternatively, it is possible to define $\precsim$ as the transitive closure of the composition of $\preceq$ and $\sim$.

[2] More precisely, a generalization of the rule where the subsumption in the antecedent is also fuzzy: if $\preceq(s_0, s_1) = \beta_0$ and $\sim(s_1, s_2) = \beta_1$, then $\preceq(s_0, s_2) = \beta_0 \wedge \beta_1$.

Proposition 5.5 (Equivalent definition of $\precsim$). *Let $(\mathcal{S}, \preceq)$ be a partial order*[3]*, let $\sim$ be a similarity relation on $\mathcal{S}$, and let $\precsim$ be as in Definition 5.4. Then $\precsim = (\preceq \circ \sim)^{\oplus}$.*

Example 5.6 (Similarity-subsumption chain). Fig. 5.1b represents the fuzzy preorder $\precsim$ obtained by combining the subsumption relation $\preceq$ and the similarity $\sim$ of Fig. 5.1a. The fuzzy subsumption edges added to $\preceq$ are represented in green. ▷

Proposition 5.7 (Similarity-subsumption preorder). *The fuzzy relation $\precsim$ of Definition 5.4 is a fuzzy preorder.*

5.3 Fuzzy similarity-subsumption partial order

It is clear from Example 5.6 that the fuzzy relation $\precsim$ in general is not a fuzzy lattice, and not even a fuzzy partial order, as it is not antisymmetric. One reason is that $|\sim| \subseteq |\precsim|$ and $|\sim|$ is symmetric, and thus $\precsim$ will contain symmetric links (such as $\precsim(\mathit{thriller}, \mathit{horror}) = \precsim(\mathit{horror}, \mathit{thriller}) > 0$ in Example 5.6) that are due directly to the similarity relation. A solution in this case simply consists in deleting such similarity links, which is also justified by the fact that these are not needed anymore for the purpose of unifying two terms (for example, the fact $\sim(\mathit{thriller}, \mathit{horror}) > 0$ has already been used to define $\precsim(\mathit{slasher}, \mathit{thriller}) > 0$, which is enough, for instance, to make the terms ψ_2 and ψ_3 of Examples 5.1 and 5.2 weakly unifiable). According to this intuition, we define the fuzzy relation $\precsim$ as follows.

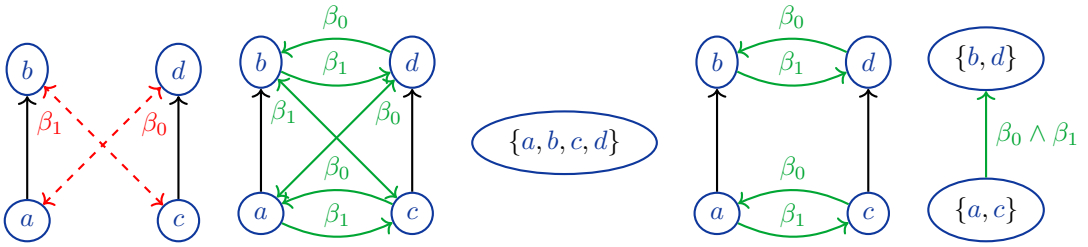
Definition 5.8 (Fuzzy preorder $\precsim$). Let $\preceq \subseteq \mathcal{S}^2$ be a partial order, $\sim : \mathcal{S}^2 \to [0, 1]$ be a similarity relation and $\precsim$ be as in Definition 5.4. The fuzzy relation $\precsim$ is defined by letting $\precsim \stackrel{\text{def}}{=} ((\precsim \dot{-} \sim) \cup \preceq)^{\oplus}$, where the difference $\dot{-}$ is defined by letting, for two fuzzy sets F and G,

$$\mu_{F \dot{-} G}(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } \mu_G(x) > 0 \\ \mu_F(x) & \text{otherwise.} \end{cases}$$

Note that the fuzzy union of $(\precsim \dot{-} \sim)$ with $\preceq$ is necessary in case $|\sim| \cap \preceq \neq \emptyset$, and the transitive closure is needed to ensure the transitivity of $\precsim$ in case transitive links are deleted by taking $(\precsim \dot{-} \sim)$.

Example 5.9 (Fuzzy preorder $\precsim$). Fig. 5.1c represents the fuzzy relation $\precsim$ obtained by combining the subsumption $\preceq$ and the similarity $\sim$ – resulting in the relation $\precsim$ represented in Fig. 5.1b – and then deleting the similarity links as per Definition 5.8. The result in this case is a fuzzy lattice which can be employed, for instance, for the unification of the terms ψ_2 and ψ_3 of Examples 5.1 and 5.2. ▷

[3]Note that the crisp relation $\preceq$ can be treated as a fuzzy relation by identifying its characteristic function $\mathbf{1}_{\preceq}$ with the membership function $\mu_{\preceq}$.



(a) $\preceq$ (black) and $\sim$ (red, dashed) (b) $\precsim$ (c) $\precsim$ on $\mathcal{S}_{/\approx}$ (d) $\precsim$ (e) $\precsim$ on $\mathcal{S}_{/\approx}$

Figure 5.3: The fuzzy relations of Examples 5.10 and 5.13.

Deleting symmetric links may work for simple cases such as the one of Fig. 5.1, but in general the relation $\precsim$ may still not be antisymmetric, as the next example shows.

Example 5.10 (Antisymmetry and $\precsim$). Consider Fig. 5.3a, where $\mathcal{S} = \{a, b, c, d\}$, $a \preceq b$, $c \preceq d$, $a \sim d = \beta_0 > 0$ and $b \sim c = \beta_1 > 0$. Then $a \preceq b \sim_{\beta_1} c$ and $c \preceq d \sim_{\beta_0} a$ so that $a \precsim_{\beta_1} c$ and $c \precsim_{\beta_0} a$ (Fig. 5.3d), but $a \neq c$, violating fuzzy antisymmetry. ▷

Proposition 5.11 (Fuzzy preorder $\precsim$). *The fuzzy relation $\precsim$ of Definition 5.8 is a fuzzy preorder.*

While cases such as the latter one could be blamed on an improper modeling of the subsumption and the similarity relations, we propose a more general solution that consists in the definition of a fuzzy partial order on equivalence classes of sorts starting from a fuzzy preorder on sorts. This construction generalizes to fuzzy set theory the well-known order theoretic transformation of a preorder into a partial order described in Proposition 1.20.

Definition 5.12 (Fuzzy partial order $\precsim$ on $\mathcal{S}_{/\approx}$). Let $(\mathcal{S}, \preceq)$ be a fuzzy preorder on $\mathcal{S}$ and $\approx$ be the equivalence relation on $\mathcal{S}$ defined by letting $s \approx s' \Leftrightarrow (s, s') \in |\preceq|$ and $(s', s) \in |\preceq|$. Let $\beta_s \stackrel{\text{def}}{=} \bigwedge_{s' \in [s]^{\approx}} (\preceq(s, s') \wedge \preceq(s', s))$ for each $s \in \mathcal{S}$. The fuzzy binary relation $\preceq$ on $\mathcal{S}_{/\approx}$ is defined by letting $\preceq([s_0]^{\approx}, [s_1]^{\approx}) \stackrel{\text{def}}{=} 1$ if $[s_0]^{\approx} = [s_1]^{\approx}$, and otherwise

$$\preceq([s_0]^{\approx}, [s_1]^{\approx}) \stackrel{\text{def}}{=} \beta_{s_0} \wedge \beta_{s_1} \wedge \bigvee_{s'_0 \in [s_0]^{\approx}, s'_1 \in [s_1]^{\approx}} \preceq(s'_0, s'_1).$$

Note that the same symbol $\preceq$ is used for the preorder on $\mathcal{S}$ and the fuzzy relation on $\mathcal{S}_{/\approx}$, since its meaning is clear from context.

Example 5.13 (Fuzzy partial order $\precsim$ on $\mathcal{S}_{/\approx}$). Continuing from Example 5.10, Fig. 5.3e depicts the fuzzy partial order $\precsim$ on the equivalence classes of $\mathcal{S}$ obtained from the fuzzy preorder $\precsim$ on $\mathcal{S}$ of Fig. 5.3d, as defined in Definition 5.12. Then, for example, the similarity-based unification of the OSF terms $b(f \rightarrow \top)$ and $c(f \rightarrow \top)$ could be performed by

considering the fuzzy subsumption on the equivalence classes of the sorts, resulting in the unifier $t_1 = \{a, c\}(f \to \top)$, where $\{a, c\}$ is a *disjunctive sort*, and t_1 is a *disjunctive OSF term* [10] (also see Section 3.8.2). Alternatively, the construction of Definition 5.12 could be applied to the fuzzy preorder $\precsim$ represented in Fig. 5.3b, leading to the partial order of Fig. 5.3c. In this case the similarity-based unification of the same two terms would result in $t_2 = \{a, b, c, d\}(f \to \top)$. ▷

Proposition 5.14 (Fuzzy partial order $\precsim$ on $\mathcal{S}_{/\approx}$). *The fuzzy relation $\preceq$ of Definition 5.12 is a fuzzy partial order on $\mathcal{S}_{/\approx}$.*

5.4 Fuzzy similarity-subsumption lattice

In crisp OSF logic, the sort subsumption relation is by definition required to be a finite lattice. In practice, however, it is enough for $(\mathcal{S}, \preceq)$ to be a partial order and to implicitly work on a *completion* of $(\mathcal{S}, \preceq)$ consisting of a lattice of sets of sorts, where singletons are treated as normal sorts, while sets of two or more sorts as *disjunctive sorts* [10]. Formally, the lattice is defined on the antichains of the sort subsumption partial order and, from an implementation standpoint, it does not need to be explicitly constructed [5] (also see Section 3.8.2).

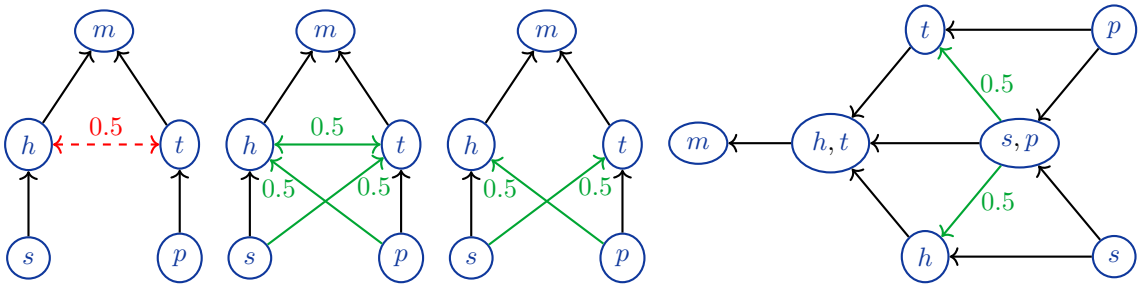
Analogously, the fuzzy sort subsumption relation of fuzzy OSF logic is required to be a fuzzy lattice, as seen in Chapter 4. In this section we present a generalization to fuzzy set theory of the antichain completion of a partial order into a lattice (Definition 1.15 and Proposition 1.17), i.e., we show how to construct a fuzzy lattice on the antichains of a fuzzy poset.

Definition 5.15 (Antichains in a fuzzy poset). Let $(\mathcal{S}, \preceq)$ be a fuzzy poset. Two elements $s, s' \in \mathcal{S}$ are said to be *incomparable* – denoted $s \parallel s'$ – if $\preceq(s, s') = 0$ and $\preceq(s', s) = 0$. An *antichain* is a subset $C \subseteq \mathcal{S}$ such that, for all $s, s' \in C$, $s \parallel s'$ if $s \neq s'$. The set of all antichains of $(\mathcal{S}, \preceq)$ is denoted $\mathit{Antichains}(\mathcal{S})$.

Definition 5.16 (Fuzzy antichain ordering). A fuzzy partial order $\preceq$ on $\mathcal{S}$ can be extended to a fuzzy relation $\preceq$ on $\mathit{Antichains}(\mathcal{S})$ by letting, for all $C, C' \in \mathit{Antichains}(\mathcal{S})$: $\preceq(C, C') \stackrel{\text{def}}{=} \min_{c \in C} \max_{c' \in C'} \preceq(c, c')$.

The same symbol $\preceq$ is used for the partial order on $\mathcal{S}$ and the fuzzy relation on $\mathit{Antichains}(\mathcal{S})$, since its meaning is always clear from context.

Proposition 5.17 (Fuzzy antichain lattice). *If $(\mathcal{S}, \preceq)$ is a fuzzy poset, then $(\mathit{Antichains}(\mathcal{S}), \preceq)$ from Definition 5.16 is a fuzzy lattice, where the GLB of $C, C' \in \mathit{Antichains}(\mathcal{S})$ is given by $C_1 \curlywedge C_2 \stackrel{\text{def}}{=} \lceil C_1{\downarrow} \cap C_2{\downarrow} \rceil^{\cdot}$. Moreover, $(\mathit{Antichains}(\mathcal{S}), \preceq)$ preserves the GLBs that exist in $(\mathcal{S}, \preceq)$, i.e., if $s_0 \curlywedge s_1 = s$, then $\{s_0\} \curlywedge \{s_1\} = \{s\}$.*



(a) $\preceq$ (black) and $\sim$ (red, dashed) (b) $\precsim$ (c) $\precnsim$ (d) Completion of $\precnsim$

Figure 5.4: The fuzzy relations of Example 5.18.

Example 5.18 (Greatest lower bounds in $\precnsim$). Consider the subsumption $\preceq$ and the similarity $\sim$ represented in Fig. 5.4a, and the fuzzy partial order $\precnsim$ obtained from their combination (according to Definition 5.8) represented in Fig. 5.4c. In this case $\precnsim$ is not a fuzzy lattice, since h and t do not have a GLB. Fig. 5.4d shows the completion of $\precnsim$ as defined in Definition 5.16, in which the GLB of $\{h\}$ and $\{t\}$ is $\{s, p\}$. We can thus use this fuzzy order for the unification, e.g., of the OSF terms $t_1 = m(f \to h)$ and $t_2 = m(f \to t)$, which would result in the disjunctive OSF term $m(f \to \{s, p\})$, which is subsumed by t_1 and t_2 with degree 0.5. ▷

Similarly to the crisp setting, the completion of a fuzzy partial order $\preceq : \mathcal{S} \to [0, 1]$ does not need to be computed explicitly in order to find the GLB of two sorts in $\mathit{Antichains}(\mathcal{S})$, as the same encoding and decoding strategies of [5, 13] can be employed in the fuzzy case, as seen in Section 4.8.2.

5.5 Potential applications

5.5.1 Similarity-based CEDAR

As seen in Section 3.7.2, CEDAR [6, 15] is a Semantic Web reasoner based on OSF logic and OSF term unification, whose implementation relies on techniques that exploit the specificity of concept taxonomies, particularly the fact that subsumption orderings are central to all ontologies. The main capabilities of CEDAR are concept classification, Boolean query answering (i.e., answering queries formed by sort symbols and Boolean connectives), and answering queries represented as OSF terms (CEDAR features an extended OSF term syntax that also supports a few Description Logic constructs [15]).

Before being executed, a query – expressed as an OSF term in which the variables of interest are marked by a question mark – is first optimized according to both the knowledge expressed in the given ontology and the OSF constraint normalization rules. The resulting query is then translated into a SPARQL query and executed by a SPARQL engine. This

```
SELECT ?x WHERE {
  ?x rdf:type :person .
  ?x :works_at ?y .
  ?y rdf:type owl:Thing . }
```

(a) Without the optimization step

```
SELECT ?x WHERE {
  ?x rdf:type :researcher .
  ?x :works_at ?y .
  ?y rdf:type :university . }
```

(b) After the optimization step

Figure 5.5: The SPARQL queries of Example 5.19.

process potentially leads to an optimized query that is more efficient to execute, by reducing the instance retrieval search space. Moreover, it also ensures the consistency of the input query against the ontology, so that no answer is provided if the query is inconsistent. The retrieval is further optimized thanks to a custom RDF triple indexing scheme based on OSF sort and attribute information [15].

Example 5.19 (CEDAR query normalization). Consider the subsumption $\preceq$ and similarity $\sim$ represented in Fig. 5.6 (adapted from an example of [15]) and the query $?X : person(works\_at \to \top)$, which aims to retrieve people that work somewhere. Additionally, assume that the feature $works\_at$ applies to objects of sort $researcher$ and points to objects of sort $university$. The CEDAR query optimization step [15] would transform the given query into the query $?X : researcher(works\_at \to university)$ (the SPARQL translations of these queries are depicted in Figs. 5.5a and 5.5b, respectively). The reason is that, since X is of sort $person$ and the feature $works\_at$ only applies to $researcher$s, then by the rule Sort Intersection the variable X must be of sort $person \wedge researcher = researcher$. An analogous reasoning applies to the value of the feature $works\_at$. The resulting query – which aims to retrieve researchers rather than people, and universities rather than entities of any sort – benefits from a much smaller search space, and is thus more efficient to execute.

On the other hand, CEDAR would find the query $?X : teacher(works\_at \to \top)$ to be inconsistent with the knowledge expressed in Fig. 5.6 (according to the subsumption $\preceq$, the sorts $teacher$ and $researcher$ are disjoint: $teacher \wedge researcher = \bot$), thus preventing the query from being translated into SPARQL and executed. ▷

While the consistency check is useful for preventing unnecessary computations, there may be scenarios where offering approximate answers to inconsistent queries becomes desirable. For instance, users often do not possess full knowledge of the extensive ontologies they query, making the inconsistency of a query, such as the second one in Example 5.19, potentially surprising. Furthermore, the inconsistency of this query is due to the closed world assumption of OSF logic, which states that, since $teacher \wedge researcher = \bot$, the two sorts must denote disjoint classes. However, it may be desirable to relax this consistency requirement in contexts where this assumption does not hold, such as settings where domain knowledge can evolve or be incomplete.

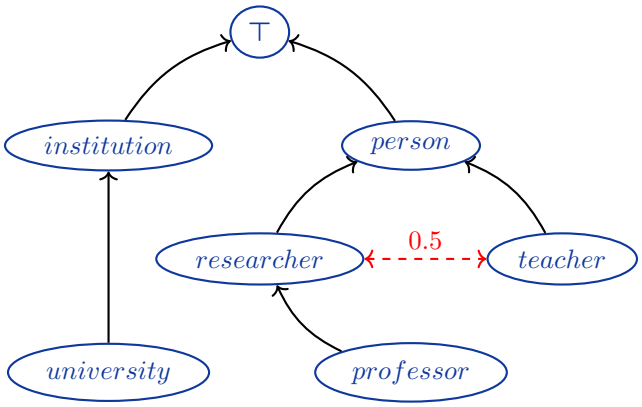


Figure 5.6: The subsumption (black) and similarity (red, dashed) relations of Examples 5.19 and 5.20.

With these considerations in mind, similarity-based reasoning with OSF logic could be implemented in an extension of CEDAR in order to relax the consistency requirement and provide approximate answers even when the input query is inconsistent with respect to the ontology. This would be achieved, for example, by enriching the subsumption relation of the given ontology with a similarity, which would then be taken into account during the query optimization phase, making query answering with CEDAR more flexible.

Example 5.20 (Similarity-based CEDAR query normalization). Continuing from Example 5.19, in a similarity-based extension of CEDAR we could consider the similarity relation $\sim$ of Fig. 5.6, which specifies that *researcher* and *teacher* are similar. The constraint normalization rules used in the query optimization phase could then be executed over the fuzzy subsumption $\precsim$ obtained by combining $\preceq$ and $\sim$ as described in the previous sections, where $professor$ becomes the GLB of *researcher* and *teacher*. The query $?X : teacher(works\_at \to \top)$, which was previously inconsistent, can now be simplified to $?X : professor(works\_at \to university)$. In this case, the answers to this query are associated with the approximation (satisfaction) degree 0.5. Alternatively, instead of providing approximate answers, the new query could be suggested to the user as a replacement to the inconsistent query. ▷

5.5.2 Logic programming with similarity-based OSF logic

Another possible application of our approach is the implementation of a logic programming language based on SLD resolution and OSF term unification (like LOGIN [8]), which additionally allows the specification of a sort similarity relation, similarly to how Bousi~Prolog [62, 63] allows the specification of a similarity (or a proximity) relation between functors, constants and predicates. The addition of a similarity relation allows the retrieval of approximate solutions to a query, improving the flexibility of query answering.

There are several potential advantages of using OSF logic in this context. First of all, the unification algorithm for OSF terms takes into account a sort subsumption relation, which can lead to more efficient computations [8, 37]. For instance, Example 3.67 shows how a single OSF term unification can replace several resolution steps, and analogously Example 5.3 discusses how a single weak OSF term unification can replace several similarity-based resolution steps. Another advantage is the flexibility provided by OSF terms, which lack a fixed arity and can thus easily represent partial information, and are moreover simpler to interpret thanks to their use of features rather than positions to specify arguments [8].

The following example, which extends Example 3.68, illustrates a similarity-based OSF logic program, showcasing the flexibility of our approach in retrieving approximate solutions to queries. The computation of a solution to a given query relies on the weak unification of OSF terms, and on SLD resolution on predicate symbols. Predicate symbols in this context are distinct from the sorts and features of OSF terms. They work exactly as in Prolog, with the only difference that they accept OSF terms rather than first-order terms as arguments.

Example 5.21 (Logic programming with OSF logic and a sort similarity). Consider the program of Fig. 5.7, which corresponds to the Bousi~Prolog program of Example 2.7. A sort subsumption relation is specified by declarations of shape s < s', while an expression like { c } < s means that the constant c is an instance of the sort s. Constants such as c are treated as singleton sorts (sorts that denote a single element), and thus as OSF terms themselves. A similarity declaration of shape $x \sim y = \beta$ specifies that the sorts `horror` and `thriller` are similar to degree 0.5. A few facts with the predicate `director_of` – whose arguments are OSF terms – are then specified and, finally, a rule involving the predicates `director_of` and `likes` states that `alinda` likes thriller movies. The query `?- likes(alinda, Y : movie)` is first reduced by resolution to the goal `director_of(X : person, Y : thriller)`, which is then resolved against the facts of the program, returning the following solutions:

- a solution binding `Y` to `memento(title -> "Memento")`, through the unification of `thriller` and `memento(title -> "Memento")`;
- a solution binding `Y` to `psycho` with approximation degree 0.5, through the weak unification of `thriller` and `psycho` (since `psycho` is a `horror`, which is similar to `thriller`);
- a solution binding `Y` to `halloween(year -> 1979)` with approximation degree 0.5, through the weak unification of `thriller` and `halloween(year -> 1979)` (since `halloween` is a `horror`, which is similar to `thriller`).

Behind the scenes, the computations would involve the unification of OSF terms over the fuzzy subsumption $\precsim$ obtained by combining, before the execution of the queries, the subsumption $\preceq$ and the similarity $\sim$ specified by the program. ▷

```
% Subsumption relation
director < person.
slasher < horror.
horror < movie.
thriller < movie.
% Similarity relation
thriller ~ horror = 0.5.
% Instances
{ alinda } < person.
{ hitchcock, carpenter, nolan  } < director.
{ psycho } < horror.
{ halloween } < slasher.
{ memento } < thriller.
% Facts
director_of(hitchcock, psycho).
director_of(carpenter, halloween(year -> 1979)).
director_of(nolan, memento(title -> "Memento")).
% Rules
likes(alinda, Y :thriller) :- director_of(X : person, Y).
```

Figure 5.7: A logic program with OSF terms and a similarity relation.

Note that in this example a similarity was only specified between sort symbols in order to perform weak OSF term unification, but it could be possible to further extend our approach by also considering a similarity between predicate symbols in order to perform similarity-based SLD resolution. This would be analogous, for instance, to Bousi~Prolog, which supports a similarity relation on predicate symbols (in order to perform similarity-based SLD resolution) and on functor symbols (in order to perform weak first-order term unification).

We conclude with an example that showcases another advantage of our approach with respect to other similarity-based logic programming languages such as Bousi~Prolog, in particular when dealing with cycles in the specification of the subsumption and similarity relations.

Example 5.22 (Cycles in Bousi~Prolog and similarity-based OSF logic). Consider the logic program enriched with a similarity relation of Fig. 5.8a, which follows the syntax of Bousi~Prolog [62, 63]. The program starts by describing the similarity and the subsumption relations of Fig. 5.3a (with $\beta_0 = 0.4$ and $\beta_1 = 0.5$) through declarations of shape $x \sim y = \beta$ and two Horn rules, followed by a few facts that specify the instances of the predicates `a`, `b`, `c` and `d`. Loading this program in Bousi~Prolog and querying `a(X)` leads to an infinite loop[4].

[4]The program of Fig. 5.8a was tested on Bousi~Prolog 3.6.1 (`https://dectau.uclm.es/bousi-prolog/2018/07/26/downloads/`).

```
a ~ d = 0.4 .
b ~ c = 0.5 .
b(X) :- a(X).
d(X) :- c(X).
a(alice).
b(bob).
c(carol).
d(david).
```

(a) Bousi~Prolog version

```
a ~ d = 0.4 .
b ~ c = 0.5 .
a < b.
c < d.
{ alice } < a.
{ bob   } < b.
{ carol } < c.
{ david } < d.
```

(b) OSF version

Figure 5.8: The logic programs with a similarity relation of Example 5.22.

This is due to the fact that, since `a` and `d` are similar, then the query `a(X)` resolves with the clause `d(X) :- c(X)` though similarity-based SLD resolution, and the goal becomes `c(X)`. Then, since `b` and `c` are similar, the new goal resolves with the clause `b(X) :- a(X)` by similarity-based SLD resolution, resulting in the goal `a(X)` again, so that the cycle starts over.

Now consider the similarity-based OSF logic program of Fig. 5.8b, which describes the same setting. The subsumption and similarity defined in the program would be combined into a fuzzy subsumption relation before the execution of the query `a(X)`, leading to the fuzzy partial order $\precsim$ of Fig. 5.3c, or the fuzzy partial order $\precapprox$ of Fig. 5.3e (this could depend on an additional parameter passed to the program). In the first case, the answers to the query would include the instances of `a`, `b`, `c` and `d` (i.e., `alice`, `bob`, `carol` and `david`), while in the second case they would consist of the instances of `a` and `c` (i.e., `alice` and `carol`). ▷

Conclusions

In this chapter we provide a brief summary of the thesis, followed by an overview of potential directions for future research.

Summary of the thesis

In **Chapter 2** we have presented a brief summary of Knowledge Representation and Reasoning (KRR) languages that are closely related to Order-Sorted Feature (OSF) logic, in order to provide the necessary background and a means of comparison of our work with the current literature. In particular, we have discussed Description Logics (DLs), one of the most prominent formalisms for KRR due to its application to the Semantic Web, and fuzzy DLs, the fuzzy generalization of DLs that aims to handle vagueness and real-valued truth degrees.

A brief overview is then provided of research on approximate reasoning with logic programming that relies on fuzzy relations such as proximities and similarities. Extending the syntax of Prolog with a similarity relation enables the definition of flexible notions of unification and SLD resolution, called weak unification and similarity-based SLD resolution. The aim is to enhance logic programming languages with the capability of providing approximate answers to queries.

Chapter 3 is a comprehensive overview of OSF logic [10]. It presents the main syntactic objects of this language – namely OSF terms, OSF clauses and OSF graphs – and how they are related to each other. The meaning of these objects is defined in structures called OSF algebras, which provide the denotation of an OSF term and the satisfiability of an OSF clause.

Structure-preserving mappings between OSF algebras, called OSF algebra homomorphisms, allow to establish an approximation partial ordering on the set of OSF graphs. A subsumption ordering on the set of normal OSF terms, and an implication ordering on rooted solved OSF clauses, are also defined, and it is proved that the three orderings are equivalent. In particular, the subsumption ordering on normal OSF terms constitutes a lattice such that the greatest lower bound (GLB) of two OSF terms can be computed through their unification, the operation on which reasoning with OSF logic is grounded.

Two applications based on OSF logic and OSF term unification are then discussed: the logic programming language LOGIN [8], and the Semantic Web reasoner CEDAR [6, 15].

The discussion of Chapter 3 sets the foundations for the development of fuzzy OSF logic in **Chapter 4**, which constitutes the first key contribution of our thesis.

Fuzzy OSF logic maintains the syntax of crisp OSF logic, but it generalizes its semantics to a fuzzy setting. In particular, each sort symbol is interpreted as a fuzzy subset of a domain of an interpretation, allowing real-valued degrees of membership of an object to a sort. Moreover, sort symbols are ordered in a fuzzy subsumption relation, which models a flexible notion of inclusion that generalizes Zadeh's definition of inclusion of fuzzy sets.

The syntactic objects of fuzzy OSF logic are interpreted in fuzzy OSF algebras (or interpretations). The denotation of an OSF term is a fuzzy subset of the domain, while OSF clauses are satisfiable in a fuzzy OSF interpretation with an associated satisfaction degree $\beta \in [0, 1]$. Similarly to the crisp case, the notions of fuzzy denotation of OSF terms and of graded satisfiability of OSF clauses are proved to be equivalent. The fuzzy OSF graph algebra, a generalization of the OSF graph algebra, is then introduced, and it is proved that any solved clause is satisfiable in this algebra with degree of satisfaction 1.

Fuzzy β-homomorphisms are then introduced as mappings between two fuzzy OSF interpretations $\mathcal{I}$ and $\mathcal{J}$ that preserve feature applications and, to some degree, the sorts of the elements of $\mathcal{I}$. These mappings enable the definition of a fuzzy ordering between OSF graphs, which is then proved to be equivalent to a fuzzy subsumption ordering between normal OSF terms, and a fuzzy implication ordering between rooted solved OSF clauses.

The fuzzy subsumption ordering between normal OSF terms generalizes the one on sort symbols, and it constitutes a fuzzy lattice. An equivalent syntactic definition of fuzzy OSF term subsumption is also provided, along with a result showing how fuzzy OSF term subsumption and crisp OSF term subsumption are related to each other.

The GLB of two normal OSF terms in the fuzzy OSF term subsumption lattice can be computed via their unification, through a constraint normalization process that is essentially the same as the one for crisp OSF logic. The fact that it is possible to employ the same unification procedure as crisp OSF logic constitutes a benefit in terms of developing fuzzy OSF logic reasoners, since it is possible to take advantage of existing implementation techniques.

The generalization to a fuzzy semantics enables approximate reasoning within the framework of OSF logic. In particular, we have provided an algorithm for OSF term unification that can be employed to decide whether two OSF terms are subsumed by each other, and to which degree. The complexity of this algorithm is discussed, followed by the implementation of two core operations it is based on: finding the GLB of two sorts in the fuzzy sort subsumption lattice, and computing the subsumption degree of a sort with respect to another sort.

Finally, we have presented an application of fuzzy OSF logic consisting of a fuzzy logic programming language based on fuzzy OSF term unification. Fuzzy OSF logic also enables the development of similarity-based reasoning with OSF logic, which is discussed in Chapter 5.

In **Chapter 5** we have presented the second key contribution of our thesis, consisting of an approach to define approximate reasoning with OSF logic by extending its language with a similarity relation on sort symbols. By combining this similarity relation with the usual sort subsumption relation of this language it is possible to define a fuzzy subsumption relation which intuitively combines the information of both relations and their interaction.

We start by combining a sort subsumption relation and a sort similarity into a *fuzzy subsumption preorder*, which may however contain cycles. To address this issue, we present a construction of a fuzzy partial order from a fuzzy preorder that generalizes a well-known result from order theory. A definition of a completion of a fuzzy poset into a fuzzy lattice is introduced next, which finalizes the transformation of a subsumption relation and a similarity into a fuzzy subsumption relation.

The fuzzy subsumption is then taken into account when unifying two OSF terms, according to the constraint normalization rules of fuzzy OSF logic. The advantage of this approach is that it allows to seamlessly integrate the similarity relation into the unification rules of fuzzy OSF logic, taking the interaction between the similarity and the crisp subsumption into account, and potentially allowing a single unification to replace several similarity-based SLD resolution steps.

Two potential applications of similarity-based reasoning with OSF logic have been presented: a fuzzy logic programming language and an extension of the CEDAR reasoner which are capable of returning approximate answer to queries.

Future work

In this section we present in distinct paragraphs some promising research directions based on the research work developed in this thesis.

Disjunction and negation The language of OSF logic adopted in this thesis consists of conjunctions of simple constraints. It would be interesting to study a fuzzy semantics of the language of OSF logic that also includes, for instance, *negations* and *disjunctions*. Versions of crisp OSF logic (and of feature logics in general) that include these operations are studied, for instance, in [4, 34, 106].

Partial features and fuzzy features Like in crisp OSF logic, feature symbols are interpreted as total functions in fuzzy OSF logic as well. Other feature logics have been studied

where features are interpreted as *partial functions* instead [4, 34, 106]. Besides investigating an extension of fuzzy OSF logic that interprets features as partial functions, another research direction would be the study of OSF logic where feature symbols are also given a fuzzy interpretation, for instance as fuzzy binary relations satisfying some functionality condition, or along the lines of the fuzzy functions of [89].

Other t-norms The definition of the fuzzy semantics of OSF logic in Chapter 4 relies on the choice of the Gödel t-norm and t-conorm, and the same is true for the development of similarity-based reasoning with OSF logic in Chapter 5. The study of the semantics of OSF logic under alternative t-norms, and the investigation of whether the results of Chapters 4 and 5 are preserved when employing these operations, is left for future research.

Proximity relations Besides similarity relations, *proximity relations* have also been considered when defining weaker notions of unification with the goal of extending logic programming languages with the capability to perform approximate reasoning (e.g., [61, 64, 74, 90]). Proximity relations generalize similarity relations by dropping the max-min transitivity requirement, which can be more appropriate in some modeling contexts [61, 67, 103], so that it would also be valuable to define a proximity-based notion of weak unification of OSF terms.

Description Logic constructs DLs offer a great variety of concept and role constructors that can be employed to build complex expressions. Depending on the choice of these constructors, a different trade-off can be achieved between the expressivity of a DL and its computational complexity. Strategies to express DL constructors in OSF logic are discussed, for instance, in [4, 15]. An analogous investigation into whether it is possible to extend fuzzy OSF logic with DL concept constructors will be the subject of future work.

Anti-unification Another direction for future work would be investigating the fuzzy *anti-unification* (or generalization) of OSF terms, i.e., the operation dual to unification. Fuzzy definitions of the anti-unification of first-order terms are provided, for instance, in [9, 71, 72, 73], and anti-unification of OSF terms is studied in [12].

Fuzzy OSF term unification modulo a theory OSF theory unification is the problem of checking the consistency of an OSF term modulo a hierarchy of sort definitions, enforcing structural constraints imposed by an OSF theory [11]. Sort definitions are comparable to classes imposing structural constraints on objects, and OSF theory unification allows objects to inherit and abide by constraints from their classes. The integration of sort definitions in fuzzy OSF logic, and the study of fuzzy OSF term unification modulo a theory of sort definitions, are left for future research.

Implementation and experimental comparisons Finally, the approaches to approximate reasoning with OSF logic developed in this thesis should be implemented in applications that are capable of providing approximate answers to queries posed to a knowledge base, by relying on a fuzzy sort subsumption relation or on a sort similarity relation. Examples of these applications are the fuzzy logic programming languages of Sections 4.9 and 5.5.2, and the extension of the CEDAR reasoner of Section 5.5.1. The efficiency and the effectiveness of systems based on the outcomes of our research could then be experimentally compared with other approaches, such as Bousi∼Prolog, or DL and fuzzy DL reasoners.

Appendix A

Proofs

A.1 Proofs for Chapter 4: Fuzzy Order-Sorted Feature Logic

A.1.1 Proofs for Section 4.5: Fuzzy OSF algebra homomorphisms

Proposition 4.25 (Fuzzy homomorphisms). *Let $\gamma : \mathcal{I} \to \mathcal{J}$ be a β-morphism.*

1. *If $\gamma' : \mathcal{J} \to \mathcal{K}$ is a β'-morphism, then $\gamma' \circ \gamma : \mathcal{I} \to \mathcal{K}$ is a $\beta \wedge \beta'$-morphism.*
2. *For all $\beta' \leq \beta$: γ is a β' morphism.*
3. *There is a maximum β' such that γ is a β'-morphism.*

Proof of Proposition 4.25. The first two points are easy to show.

Let $B \stackrel{\text{def}}{=} \{\beta \in [0,1] \mid \gamma : \mathcal{I} \to \mathcal{J} \text{ is a } \beta\text{-morphism}\}$ and $\beta_s = \sup(B)$. Note that

$$\begin{aligned} B &\stackrel{\text{def}}{=} \{\beta \in [0,1] \mid \gamma : \mathcal{I} \to \mathcal{J} \text{ is a } \beta\text{-morphism}\} \\ &= \{\beta \in [0,1] \mid \forall s \in \mathcal{S}, \forall d \in \Delta^{\mathcal{I}} : s^{\mathcal{I}}(d) \wedge \beta \leq s^{\mathcal{J}}(\gamma(d))\} \end{aligned}$$

and that $B \neq \emptyset$ by assumption. The fact that $\beta_s \in B$ and thus β_s is the maximum β' such that γ is a β'-morphism follows from the following claim.

Claim A.1. For each $1 \leq i \leq n$, let $f_i : X \to [0,1]$ and $g_i : X \to [0,1]$ be functions. Let

$$B \stackrel{\text{def}}{=} \{\beta \in [0,1] \mid \forall x \in X, \forall 1 \leq i \leq n : \min(f_i(x), \beta) \leq g_i(x)\}.$$

Then $\sup(B) \in B$. □

Theorem 4.29 (Fuzzy morphisms into $\mathcal{G}$). *For any fuzzy OSF interpretation $\mathcal{I}$ there exists a β-homomorphism into the fuzzy OSF graph algebra $\mathcal{G}$ for some $\beta \in (0,1]$.*

Proof of Theorem 4.29. For each element $d \in \Delta^{\mathcal{I}}$ we construct an OSF graph $\gamma(d) \in \Delta^{\mathcal{G}}$.

Let us start with a few preliminary definitions. Let $d \in \Delta^{\mathcal{I}}$:

- the set of sorts to which d belongs to with degree greater than 0 is denoted[1]

$$|d|^{\mathcal{I}}_{\mathcal{S}} \stackrel{\text{def}}{=} \{s \in \mathcal{S} \mid s^{\mathcal{I}}(d) > 0\};$$

- the most specific sort to which d belongs to with degree greater than 0 is denoted $s_d \stackrel{\text{def}}{=} \bigwedge |d|^{\mathcal{I}}_{\mathcal{S}}$.

We construct a labeled graph $g = (N, E, \lambda_N, \lambda_E)$ with $N \subseteq \mathcal{V}$ as follows:

- for each $d \in \Delta^{\mathcal{I}}$ we choose a variable $X_d \in \mathcal{V}$ to denote a node in g;
- the label of X_d is defined as $\lambda_N(X_d) \stackrel{\text{def}}{=} s_d$;
- for each $f \in \mathcal{F}$, if $f^{\mathcal{I}}(d) = d'$, then g contains the edge $(X_d, X_{d'})$ labeled f.

For each $d \in \Delta^{\mathcal{I}}$, let $\beta_d = \min_{s \in |d|^{\mathcal{I}}_{\mathcal{S}}} \preceq(s_d, s)$. Note that $\beta_d > 0$ for any $d \in \Delta^{\mathcal{I}}$, and that because the set $\mathcal{S}$ of sorts is finite, then the set $\{\beta_d \mid d \in \Delta^{\mathcal{I}}\} = \{\min_{s \in |d|^{\mathcal{I}}_{\mathcal{S}}} \preceq(s_d, s) \mid d \in \Delta^{\mathcal{I}}\}$ is also finite. Let $\beta = \min\{\beta_d \mid d \in \Delta^{\mathcal{I}}\} > 0$ and define $\gamma(d)$ to be the maximally connected subgraph of g rooted in X_d. It is left to show that $\gamma : \mathcal{I} \to \mathcal{G}$ is a β-homomorphism.

- Let $f \in \mathcal{F}$ and $d \in \Delta^{\mathcal{I}}$. If $f^{\mathcal{I}}(d) = d'$, then $\gamma(d)$ has an edge labeled f from X_d to $X_{d'}$, which is the root of $\gamma(d')$. Hence $\gamma(f^{\mathcal{I}}(d)) = f^{\mathcal{G}}(\gamma(d))$.
- Let $s \in \mathcal{S}$ and $d \in \Delta^{\mathcal{I}}$. If $s^{\mathcal{I}}(d) = 0$, then $s^{\mathcal{G}}(\gamma(d)) \geq \beta \wedge s^{\mathcal{I}}(d)$. Otherwise, $s \in |d|^{\mathcal{I}}_{\mathcal{S}}$ and thus $s^{\mathcal{G}}(\gamma(d)) = \preceq(s_d, s) \geq \min_{s' \in |d|^{\mathcal{I}}_{\mathcal{S}}} \preceq(s_d, s') = \beta_d \geq \beta \geq \beta \wedge s^{\mathcal{I}}(d)$. □

Theorem 4.31 (Extracting solutions through fuzzy homomorphisms). *For any solved OSF clause ϕ, fuzzy interpretation $\mathcal{I}$, assignment $\alpha : \mathcal{V} \to \Delta^{\mathcal{I}}$ and $\beta \in (0, 1]$ such that $\mathcal{I}, \alpha \models_\beta \phi$ there exists an OSF algebra β-homomorphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ such that $\alpha(X) = \gamma(G(\phi(X)))$ for each $X \in \mathit{Tags}(\phi)$.*

Proof of Theorem 4.31. Let us abbreviate $G(\phi(X))$ as g_X. The β-morphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ is defined as follows. Let $g \in \mathcal{G}[\Delta^{\mathcal{G},\phi}]$, so that $g = w^{\mathcal{G}}(g_X)$ for some $w \in \mathcal{F}^*$ and $X \in \mathit{Tags}(\phi)$: we define $\gamma(g) = \gamma(w^{\mathcal{G}}(g_X)) \stackrel{\text{def}}{=} w^{\mathcal{I}}(\alpha(X))$. Note that $\gamma(g_X) = \alpha(X)$ for each $X \in \mathit{Tags}(\phi)$. First of all, we need to show that γ is well-defined. Let $g \in \mathcal{G}[\Delta^{\mathcal{G},\phi}]$ and note that there are two cases to consider.

- Suppose $g = g_X$ for some $X \in \mathit{Tags}(\phi)$ and let $w, w' \in \mathcal{F}^*$ and $Y, Z \in \mathit{Tags}(\phi)$ be such that $g = w^{\mathcal{G}}(g_Y) = w'^{\mathcal{G}}(g_Z)$. We want to prove that $w^{\mathcal{I}}(\alpha(Y)) = w'^{\mathcal{I}}(\alpha(Z))$. By construction of g_X it must be the case that ϕ contains constraints of shape

$$\begin{array}{llll} Y.f_0 \doteq Y_1, & Y_1.f_1 \doteq Y_2, & \ldots, & Y_n.f_n \doteq X \quad \text{and} \\ Z.f'_0 \doteq Z_1, & Z_1.f'_1 \doteq Z_2, & \ldots, & Z_m.f'_m \doteq X \end{array}$$

[1] Recall that we assume that $(\mathcal{S}, \preceq)$ is finite, and thus $|d|^{\mathcal{I}}_{\mathcal{S}}$ is a finite subset of $\mathcal{S}$.

such that $w = f_0 \cdot f_1 \cdots f_n$ and $w' = f'_0 \cdot f'_1 \cdots f'_m$. Because $\mathcal{I}, \alpha \models_\beta \phi$ and $\beta > 0$, then indeed $w^{\mathcal{I}}(\alpha(Y)) = w'^{\mathcal{I}}(\alpha(Z))$ as desired.

- Suppose $g = G(Z : \top)$ for some variable $Z \notin \mathit{Tags}(\phi)$. Then $g = w'^{\mathcal{G}}(g_X)$ for some $X \in \mathit{Tags}(\phi)$ and some $w' \in \mathcal{F}^*$ such that g_X and w' uniquely determine Z (see Definition 4.17). This case now reduces to the previous one.

We now prove that the function γ is indeed a β-morphism.

- Let $f \in \mathcal{F}$ and $g \in \mathcal{G}[\Delta^{\mathcal{G},\phi}]$, so that g has shape $g = w^{\mathcal{G}}(g_X)$ as before. Let $w' = w \cdot f \in \mathcal{F}^*$. Then $\gamma(f^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]}(g)) = \gamma(f^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]}(w^{\mathcal{G}}(g_X))) = \gamma(f^{\mathcal{G}}(w^{\mathcal{G}}(g_X))) = \gamma(w'^{\mathcal{G}}(g_X)) = w'^{\mathcal{I}}(\alpha(X)) = f^{\mathcal{I}}(w^{\mathcal{I}}(\alpha(X))) = f^{\mathcal{I}}(\gamma(g))$.

- Now let $s \in \mathcal{S}$ and we want to show that $s^{\mathcal{I}}(\gamma(g)) \geq \beta \wedge s^{\mathcal{G}}(g)$ for all $g \in \mathcal{G}[\Delta^{\mathcal{G},\phi}]$. Let $g \in \mathcal{G}[\Delta^{\mathcal{G},\phi}]$ be arbitrary. There are two cases to consider.

  - Suppose $g = G(Z : \top)$ for some variable $Z \notin \mathit{Tags}(\phi)$. If $s^{\mathcal{G}}(g) = 0$ then the result holds. Otherwise, if $s^{\mathcal{G}}(g) > 0$, then g is labeled by a sort s' such that $\preceq(s', s) > 0$, but since g is labeled by $s' = \top$, then $s = \top$, so that $s^{\mathcal{I}}(\gamma(g)) = \top^{\mathcal{I}}(\gamma(g)) = 1$ and the result holds.
  - Suppose $g = g_X$ for some $X \in \mathit{Tags}(\phi)$ and let s' be the label of the root of g. By construction of g_X then ϕ must contain a constraint of the form $X : s'$, and by the assumption that $\mathcal{I}, \alpha \models_\beta \phi$ we have that $s'^{\mathcal{I}}(\alpha(X)) \geq \beta$. By Definition 4.4 then we have that $s^{\mathcal{I}}(\gamma(g_X)) = s^{\mathcal{I}}(\alpha(X)) \geq s'^{\mathcal{I}}(\alpha(X)) \wedge \preceq(s', s) \geq \beta \wedge \preceq(s', s) = \beta \wedge s^{\mathcal{G}}(g)$. □

Theorem 4.32 (Denotation of ψ-terms via fuzzy morphisms). *Let ψ be a normal OSF term, let $\phi = \phi(\psi)$, and let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a fuzzy OSF interpretation. For all $d \in \Delta^{\mathcal{I}}$ and $\beta \in (0, 1]$:*

$$[\![\psi]\!]^{\mathcal{I}}(d) \geq \beta \;\Leftrightarrow\; \textit{there is a } \beta\textit{-morphism } \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \textit{ such that } d = \gamma(G(\psi))$$

and thus $[\![\psi]\!]^{\mathcal{I}}(d) = \sup(\{\beta \in [0,1] \mid \exists \beta\textit{-morphism } \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \textit{ such that } d = \gamma(G(\psi))\})$.

Proof of Theorem 4.32. Let X be the root variable of ψ and let $d \in \Delta^{\mathcal{I}}$.

Suppose that $[\![\psi]\!]^{\mathcal{I}}(d) \geq \beta$. By Definition 4.6 and Remark 5 then there is some α such that $[\![\psi]\!]^{\mathcal{I},\alpha}(d) \geq \beta$ and $d = \alpha(X)$. By Proposition 4.10 then $\mathcal{I}, \alpha \models_\beta \phi$, so that by Theorem 4.31 there is a β-homomorphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ such that $d = \alpha(X) = \gamma(G(\phi(X))) = \gamma(G(\psi))$.

Now suppose $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ is a β-homomorphism such that $d = \gamma(G(\psi))$. By Proposition 4.21 we know that $\mathcal{G}[\Delta^{\mathcal{G},\phi}], \alpha \models_1 \phi$ where $\alpha : \mathcal{V} \to \Delta^{\mathcal{G}[\Delta^{\mathcal{G},\phi}]}$ is such that $\alpha(Y) = G(\phi(Y))$

for all $Y \in \mathit{Tags}(\phi)$, and thus $d = \gamma(G(\psi)) = \gamma(G(\phi(X))) = \gamma(\alpha(X))$. Then by Proposition 4.28 $\mathcal{I}, \gamma \circ \alpha \models_\beta \phi$, and thus by Proposition 4.10 it holds that $[\![\psi]\!]^{\mathcal{I},\gamma\circ\alpha}(\gamma(\alpha(X))) \geq \beta$, i.e., $[\![\psi]\!]^{\mathcal{I},\gamma\circ\alpha}(d) \geq \beta$, so that $[\![\psi]\!]^{\mathcal{I}}(d) \geq \beta$.

Finally, note that by what we just proved (and the fact that, for any $S \subseteq [0,1]$, $\sup(S) = \sup(S \setminus \{0\})$):

$$\begin{aligned}
[\![\psi]\!]^{\mathcal{I}}(d) &= \sup(\{\beta \in [0,1] \mid [\![\psi]\!]^{\mathcal{I}}(d) \geq \beta\}) \\
&= \sup(\{\beta \in (0,1] \mid [\![\psi]\!]^{\mathcal{I}}(d) \geq \beta\}) \\
&= \sup(\{\beta \in (0,1] \mid \exists\ \beta\text{-morphism } \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \text{ such that } d = \gamma(G(\psi))\}) \\
&= \sup(\{\beta \in [0,1] \mid \exists\ \beta\text{-morphism } \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I} \text{ such that } d = \gamma(G(\psi))\}). \quad \square
\end{aligned}$$

A.1.2 Proofs for Section 4.6: Fuzzy OSF orderings and subsumption

Proposition 4.35 (Endomorphic approximation fuzzy preorder). *For all fuzzy OSF interpretations $\mathcal{I}$, the fuzzy binary relation $\sqsubseteq^{\mathcal{I}}$ is a fuzzy preorder.*

Proof of Proposition 4.35. Let $\mathcal{I}$ be a fuzzy OSF interpretation. For all $d \in \mathcal{I}$ the identity function on $\mathcal{I}[d]$ is clearly a 1-morphism, so that $\sqsubseteq^{\mathcal{I}}(d,d) = 1$. Now suppose $\sqsubseteq^{\mathcal{I}}(d_0,d_1) = \beta_0$ and $\sqsubseteq^{\mathcal{I}}(d_1,d_2) = \beta_1$. We want to prove that $\sqsubseteq^{\mathcal{I}}(d_0,d_2) \geq \beta_0 \wedge \beta_1$. Let $\beta_0 > 0$ and $\beta_1 > 0$ (otherwise, the desired result follows immediately) and let $\gamma_0 : \mathcal{I}[d_0] \to \mathcal{I}[d_1]$ and $\gamma_1 : \mathcal{I}[d_1] \to \mathcal{I}[d_2]$ be, respectively, a β_0-morphism and a β_1-morphism. Note that these exist because of Proposition 4.27. Then by Proposition 4.25 $\gamma_1 \circ \gamma_0$ is a $\beta_0 \wedge \beta_1$ morphism, and $\sqsubseteq^{\mathcal{I}}(d_0,d_2) \geq \beta_0 \wedge \beta_1$ follows by Definition 4.33. $\square$

Proposition 4.36 (Fuzzy morphisms between two OSF graphs). *Let g_0 and g_1 be two OSF graphs. If $\gamma_0 : \mathcal{G}[g_0] \to \mathcal{G}[g_1]$ is a β_0-morphism ($\beta_0 > 0$) and $\gamma_1 : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$ is a β_1-morphism ($\beta_1 > 0$) such that $\gamma_0(g_0) = g_1$ and $\gamma_1(g_1) = g_0$, then:*

1. *for all $g \in \mathcal{G}[g_0]$, g and $\gamma_0(g)$ are labeled by the same sort;*
2. *for all $g \in \mathcal{G}[g_1]$, g and $\gamma_1(g)$ are labeled by the same sort;*
3. *$\gamma_0 \circ \gamma_1 = id_{\Delta^{\mathcal{G}[g_1]}}$ and $\gamma_1 \circ \gamma_0 = id_{\Delta^{\mathcal{G}[g_0]}}$, and thus γ_0 and γ_1 are bijections;*
4. *γ_0 and γ_1 are 1-morphisms.*

Proof of Proposition 4.36. According to the statement of the proposition, let

- $\gamma_0 : \mathcal{G}[g_0] \to \mathcal{G}[g_1]$ be a β_0-morphism such that $\gamma_0(g_0) = g_1$, and
- $\gamma_1 : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$ be a β_1-morphism such that $\gamma_1(g_1) = g_0$

with $\beta_0 > 0$ and $\beta_1 > 0$.

Let $g \in \mathcal{G}[g_1]$, so that $g = w^{\mathcal{G}}(g_1)$ for some $w \in \mathcal{F}^*$. Then

$$\begin{array}{rclcl} \gamma_0(\gamma_1(g)) & = & \gamma_0(\gamma_1(w^{\mathcal{G}}(g_1))) & = & \gamma_0(w^{\mathcal{G}}(\gamma_1(g_1))) \\ & = & w^{\mathcal{G}}(\gamma_0(\gamma_1(g_1))) & = & w^{\mathcal{G}}(\gamma_0(g_0)) = w^{\mathcal{G}}(g_1) = g, \end{array}$$

i.e., $\gamma_0 \circ \gamma_1 = id_{\Delta^{\mathcal{G}[g_1]}}$.

Let s be the sort labeling g and s' be the sort labeling $g' \stackrel{\text{def}}{=} \gamma_1(g)$. By Definition 4.23 then

$$s^{\mathcal{G}}(g) \wedge \beta_1 \leq s^{\mathcal{G}}(g') \text{ and } s'^{\mathcal{G}}(g') \wedge \beta_0 \leq s'^{\mathcal{G}}(\gamma_0(g')) = s'^{\mathcal{G}}(g).$$

Note that $s^{\mathcal{G}}(g) = 1$ and $s'^{\mathcal{G}}(g') = 1$, so that $s^{\mathcal{G}}(g') > 0$ and $s'^{\mathcal{G}}(g) > 0$. Thus

$$\preceq(s', s) = s^{\mathcal{G}}(g') > 0 \text{ and } \preceq(s, s') = s'^{\mathcal{G}}(g) > 0,$$

so that by antisymmetry of $\preceq$ it follows that $s = s'$, meaning that g and $\gamma_1(g)$ are labeled by the same sort.

Now let s be an arbitrary sort, and $g \in \mathcal{G}[g_1]$ be arbitrary. By what we just showed, g and $\gamma_1(g)$ are labeled by the same sort, so that $s^{\mathcal{G}}(g) = s^{\mathcal{G}}(\gamma_1(g))$, and thus $s^{\mathcal{G}}(g) \wedge 1 \leq s^{\mathcal{G}}(\gamma_1(g))$, i.e., γ_1 is a 1-morphism.

In a very similar way it can be shown that (i) γ_0 is a 1-morphism, (ii) for all $g \in \mathcal{G}[g_0]$, g and $\gamma_0(g)$ are labeled by the same sort, and (iii) $\gamma_1 \circ \gamma_0 = id_{\Delta^{\mathcal{G}[g_0]}}$. □

Lemma A.2 (Equivalence of fuzzy OSF orderings). *If the normal OSF terms ψ and ψ' (with roots Y and X, respectively, and no common variables), the OSF graphs g and g', and the rooted solved OSF clauses ϕ_Y and ϕ'_X respectively correspond to one another though the syntactic mappings, then the following are equivalent, for all $\beta \in (0, 1]$.*

1. *There is a β-morphism $\gamma : \mathcal{G}[g] \to \mathcal{G}[g']$ such that $\gamma(g) = g'$.*
2. *For all OSF interpretations $\mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$: $[\![\psi']\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![\psi]\!]^{\mathcal{I}}(d)$.*
3. *For all OSF interpretations $\mathcal{I}$ and assignments α such that $\mathcal{I}, \alpha \models_{\beta'} \phi'$, there is an assignment α' such that $\alpha'(X) = \alpha(X)$ and $\mathcal{I}, \alpha' \models_{\beta' \wedge \beta} \phi[X/Y]$.*
4. *For all $g \in \Delta^{\mathcal{G}}$: $[\![\psi']\!]^{\mathcal{G}}(g) \wedge \beta \leq [\![\psi]\!]^{\mathcal{G}}(g)$.*

Proof of Lemma A.2.

(1) implies (2) Suppose that there is a β-homomorphism $\gamma : \mathcal{G}[g] \to \mathcal{G}[g']$ such that $g' = \gamma(g)$. We want to show that for all $\mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$:

$$[\![\psi']\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![\psi]\!]^{\mathcal{I}}(d).$$

Let $\mathcal{I}$ be an arbitrary OSF interpretation, let $d \in \Delta^{\mathcal{I}}$ and suppose $[\![\psi']\!]^{\mathcal{I}}(d) = \beta' > 0$ (otherwise the desired result immediately follows). Then by Theorem 4.32 there is a β'-morphism $\gamma' : \mathcal{G}[\Delta^{\mathcal{G},\phi'}] \to \mathcal{I}$ such that $d = \gamma'(G(\psi')) = \gamma'(g')$. Note that $\mathcal{G}[g] = \mathcal{G}[\Delta^{\mathcal{G},\phi}]$ and $\mathcal{G}[g'] = \mathcal{G}[\Delta^{\mathcal{G},\phi'}]$, thus we can also write $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{G}[\Delta^{\mathcal{G},\phi'}]$. Then by Proposition 4.25 $\gamma' \circ \gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{I}$ is a $\beta \wedge \beta'$-morphism and $d = \gamma'(\gamma(g)) = (\gamma' \circ \gamma)(g) = (\gamma' \circ \gamma)(G(\psi))$. By Theorem 4.32 then $[\![\psi]\!]^{\mathcal{I}}(d) \geq \beta \wedge \beta' = \beta \wedge [\![\psi']\!]^{\mathcal{I}}(d)$.

(2) implies (1) Suppose that, for any $\mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$: $[\![\psi']\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![\psi]\!]^{\mathcal{I}}(d)$. Then in particular for all $g'' \in \Delta^{\mathcal{G}}$: $[\![\psi']\!]^{\mathcal{G}}(g'') \wedge \beta \leq [\![\psi]\!]^{\mathcal{G}}(g'')$.

Since $g' = G(\psi')$, then $[\![\psi']\!]^{\mathcal{G}}(g') = 1$, so that $[\![\psi]\!]^{\mathcal{G}}(g') \geq \beta > 0$. Thus by Theorem 4.32 there is a β-morphism $\gamma : \mathcal{G}[\Delta^{\mathcal{G},\phi}] \to \mathcal{G}$ such that $g' = \gamma(G(\psi)) = \gamma(g)$. Note that $\mathcal{G}[g] = \mathcal{G}[\Delta^{\mathcal{G},\phi}]$, since $g = G(\phi)$. Then by Proposition 4.27 $\gamma : \mathcal{G}[g] \to \mathcal{G}[\gamma(g)]$, i.e., $\gamma : \mathcal{G}[g] \to \mathcal{G}[g']$ as desired.

The following claim will be needed in the next two directions.

Claim A.3. Let ϕ be an OSF clause, $\beta \in [0,1]$, $Y \in \mathit{Tags}(\phi)$ and $X \notin \mathit{Tags}(\phi)$: if $\mathcal{I}, \alpha \models_{\beta'} \phi$, then $\mathcal{I}, \alpha' \models_{\beta'} \phi[X/Y]$, where

$$\alpha'(Z) = \begin{cases} \alpha(Y) & \text{if } Z = X \\ \alpha(Z) & \text{otherwise.} \end{cases}$$

(2) implies (3) Suppose that $\mathcal{I}, \alpha \models_{\beta'} \phi'$. Then $[\![\psi']\!]^{\mathcal{I}}(\alpha(X)) \geq \beta'$ by Proposition 4.10, so that by assumption (2) $[\![\psi]\!]^{\mathcal{I}}(\alpha(X)) \geq \beta \wedge \beta'$. If $\beta' = 0$ the desired result follows immediately, so suppose otherwise. By Remark 5 and Proposition 4.10 then there is some α' such that $\mathcal{I}, \alpha' \models_{\beta' \wedge \beta} \phi$ and $\alpha(X) = \alpha'(Y)$ (recall that Y is the root tag of ϕ and ψ). But then α'' defined by letting

$$\alpha''(Z) = \begin{cases} \alpha'(Y) & \text{if } Z = X \\ \alpha'(Z) & \text{otherwise} \end{cases}$$

is such that $\mathcal{I}, \alpha'' \models_{\beta' \wedge \beta} \phi[X/Y]$ by Claim A.3, and $\alpha''(X) = \alpha'(Y) = \alpha(X)$ as desired.

(3) implies (2) Let $\mathcal{I}$ and $d \in \Delta^{\mathcal{I}}$ be arbitrary and we want to show that $[\![\psi']\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![\psi]\!]^{\mathcal{I}}(d)$. Suppose that $[\![\psi']\!]^{\mathcal{I}}(d) = \beta' > 0$ (otherwise the result follows immediately). By Remark 5 and Proposition 4.10 then there is some α such that $d = \alpha(X)$ and $\mathcal{I}, \alpha \models_{\beta'} \phi'$.

By assumption (3) then there is some α' such that $\alpha'(X) = \alpha(X)$ and $\mathcal{I}, \alpha' \models_{\beta \wedge \beta'} \phi[X/Y]$. Let α'' be defined by letting

$$\alpha''(Z) = \begin{cases} \alpha'(X) & \text{if } Z = Y \\ \alpha'(Z) & \text{otherwise.} \end{cases}$$

Then $\mathcal{I}, \alpha'' \models_{\beta \wedge \beta'} \phi[X/Y][Y/X]$ by Claim A.3, i.e., $\mathcal{I}, \alpha'' \models_{\beta \wedge \beta'} \phi$. Since $\alpha''(Y) = \alpha'(X) = \alpha(X) = d$, then by Proposition 4.10 $[\![\psi]\!]^{\mathcal{I}}(d) \geq \beta \wedge \beta' = \beta \wedge [\![\psi']\!]^{\mathcal{I}}(d)$ as desired.

(2) implies (4) Obvious.

(4) implies (1) Similar to ***(2) implies (1)***. □

Theorem 4.45 (Equivalence of fuzzy OSF orderings). *If the normal OSF terms ψ and ψ' (with roots Y and X, respectively, and no common variables), the OSF graphs g and g', and the rooted solved OSF clauses ϕ_Y and ϕ'_X respectively correspond to one another though the syntactic mappings, then the following are equivalent: (1) $g \sqsubseteq^{\mathcal{G}}_{\beta} g'$, (2) $\psi' \preceq_{\beta} \psi$, and (3) $\phi'_X \models_{\beta} \phi_Y$.*

Proof of Theorem 4.45. Note that, by Lemma A.2 (and the fact that, for any $S \subseteq [0,1]$, $\sup(S) = \sup(S \setminus \{0\})$):

$$
\begin{aligned}
& \sup(\{\beta \in [0,1] \mid \gamma(g) = g' \text{ for some } \beta\text{-homomorphism } \gamma : \mathcal{G}[g] \to \mathcal{G}[g']\}) \\
= \; & \sup\left(\left\{ \beta \in [0,1] \;\middle|\; \begin{array}{l} \forall \mathcal{I}, \forall d \in \Delta^{\mathcal{I}} : (\mathcal{I}, \alpha \models_{\beta'} \phi') \Rightarrow (\exists \alpha'.\alpha'(X) = \alpha(X) \\ \text{and } \mathcal{I}, \alpha' \models_{\beta' \wedge \beta} \phi[X/Y] \end{array} \right\}\right) \\
= \; & \sup(\{\beta \in [0,1] \mid \forall \mathcal{I}, \forall d \in \Delta^{\mathcal{I}} : [\![\psi']\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![\psi]\!]^{\mathcal{I}}(d)\}).
\end{aligned}
$$

□

Lemma A.4 (Semantic and syntactic subsumption). *Let ψ_0 and ψ_1 be two normal OSF terms. Then, for all $\beta \in (0,1]$: $\preceq(\psi_0, \psi_1) \geq \beta$ if and only if there are two (normal) OSF terms ψ'_0 and ψ'_1 such that $\psi_0 \equiv \psi'_0$, $\psi_1 \equiv \psi'_1$, and $\trianglelefteq(\psi'_0, \psi'_1) \geq \beta$.*

Proof of Lemma A.4.

($\Leftarrow$) Let ψ'_0 and ψ'_1 be as in the statement of the lemma, with $h : \mathit{Tags}(\psi'_1) \to \mathit{Tags}(\psi'_0)$ witnessing $\trianglelefteq(\psi'_0, \psi'_1) \geq \beta$. By Proposition 4.40 it holds that $[\![\psi_0]\!]^{\mathcal{I}} = [\![\psi'_0]\!]^{\mathcal{I}}$ and $[\![\psi_1]\!]^{\mathcal{I}} = [\![\psi'_1]\!]^{\mathcal{I}}$ for all $\mathcal{I}$. We show that $\preceq(\psi_0, \psi_1) \geq \beta$ by showing that $\forall \mathcal{I}, \forall d \in \Delta^{\mathcal{I}} : [\![\psi_0]\!]^{\mathcal{I}}(d) \wedge \beta \leq [\![\psi_1]\!]^{\mathcal{I}}(d)$.

Let $\mathcal{I}$ be an arbitrary fuzzy interpretation, let $d \in \Delta^{\mathcal{I}}$ and suppose $[\![\psi_0]\!]^{\mathcal{I}}(d) = [\![\psi'_0]\!]^{\mathcal{I}}(d) = \beta_0$. Assume $\beta_0 > 0$, as otherwise the result follows immediately. Then by Proposition 4.10 and Remark 5 there is some $\alpha_0 : \mathcal{V} \to \Delta^{\mathcal{I}}$ such that $\mathcal{I}, \alpha_0 \models_{\beta_0} \phi'_0$, where $\phi'_0 = \phi(\psi'_0)$, $d = \alpha_0(X'_0)$, and $X'_0 = \mathit{RootTag}(\psi'_0)$. Define $\alpha_1 : \mathcal{V} \to \Delta^{\mathcal{I}}$ as follows, for any $X \in \mathcal{V}$:

$$
\alpha_1(X) = \begin{cases} \alpha_0(h(X)) & \text{if } X \in \mathit{Tags}(\psi'_1), \\ \alpha_0(X) & \text{otherwise.} \end{cases}
$$

Then $\mathcal{I}, \alpha_1 \models_{\beta \wedge \beta_0} \phi'_1$, where $\phi'_1 = \phi(\psi'_1)$:

- Suppose ϕ_1' contains a constraint of the form $X_1 : s_1$. Since ϕ_1' is a rooted solved clause constructed from ψ_1', then it must be the case that $s_1 = \mathit{Sort}_{\psi_1'}(X_1)$. Let $s_0 = \mathit{Sort}_{\psi_0'}(h(X_1))$ and note that by assumption that $\trianglelefteq(\psi_0', \psi_1') \geq \beta$ we have that (a) $\preceq(s_0, s_1) \geq \beta$. By construction ϕ_0' contains a constraint of the form $h(X_1) : s_0$, and since $\mathcal{I}, \alpha_0 \models_{\beta_0} \phi_0'$, then (b) $s_0^{\mathcal{I}}(\alpha_0(h(X_1))) \geq \beta_0$. Thus

$$\begin{array}{rcll} s_1^{\mathcal{I}}(\alpha_1(X_1)) & = & s_1^{\mathcal{I}}(\alpha_0(h(X_1))) & \\ & \geq & s_0^{\mathcal{I}}(\alpha_0(h(X_1))) \wedge \preceq(s_0, s_1) & \text{(Definition 4.4)} \\ & \geq & \beta_0 \wedge \beta. & \text{(by (a) and (b))} \end{array}$$

- Suppose ϕ_1' contains a constraint of the form $X.f \doteq Y$. Then $X \xrightarrow{f}_{\psi_1'} Y$ and thus by assumption that $\trianglelefteq(\psi_0', \psi_1') \geq \beta$ we have that $h(X) \xrightarrow{f}_{\psi_0'} h(Y)$, so that ϕ_0' must contain a constraint of the form $h(X).f \doteq h(Y)$. Since $\mathcal{I}, \alpha_0 \models_{\beta_0} \phi_0'$ and $\beta_0 > 0$, then $f^{\mathcal{I}}(\alpha_0(h(X))) = \alpha_0(h(Y))$, and thus $f^{\mathcal{I}}(\alpha_1(X)) = \alpha_1(Y)$, so that $\mathcal{I}, \alpha_1 \models_{\beta \wedge \beta_0} X.f \doteq Y$.

Let $X_1' = \mathit{RootTag}(\psi_1')$. By Definition 4.47 then $h(X_1') = X_0'$, so that $\alpha_1(X_1') = \alpha_0(h(X_1')) = \alpha_0(X_0') = d$. Since $\mathcal{I}, \alpha_1 \models_{\beta \wedge \beta_0} \phi_1'$, then by Proposition 4.10 $[\![\psi_1']\!]^{\mathcal{I}}(d) = [\![\psi_1]\!]^{\mathcal{I}}(d) \geq \beta_0 \wedge \beta = [\![\psi_0]\!]^{\mathcal{I}}(d) \wedge \beta$ as desired.

($\Rightarrow$) **(Sketch)** Let $\phi_0 = \phi(\psi_0)$ and $\phi_1 = \phi(\psi_1)$ and $g_0 = G(\psi_0)$ and $g_1 = G(\psi_1)$. In the following, we abbreviate $G(\phi_0(X))$ as g_0^X and $G(\phi_1(X))$ as g_1^X (in other words, g_i^X is the subgraph of g_i rooted at the node corresponding to the variable $X \in \mathit{Tags}(\psi_i)$).

Since $\preceq(\psi_0, \psi_1) \geq \beta$, then $\sqsubseteq^{\mathcal{G}}(g_1, g_0) \geq \beta$, so there is a β-morphism $\gamma : \mathcal{G}[g_1] \to \mathcal{G}[g_0]$ such that $g_0 = \gamma(g_1)$. The following steps almost provide a mapping $h : \mathit{Tags}(\psi_1) \to \mathit{Tags}(\psi_0)$ that witnesses $\trianglelefteq(\psi_0, \psi_1) \geq \beta$:

1. for $Y \in \mathit{Tags}(\psi_1)$, consider $g_1^Y \in \mathcal{G}[g_1]$ and $\gamma(g_1^Y) \in \mathcal{G}[g_0]$;
2. let X be the variable in ψ_0 such that $g_0^X = \gamma(g_1^Y)$; and
3. define $h(Y) \stackrel{\text{def}}{=} X$.

Unfortunately, nothing guarantees that such a variable $X \in \mathit{Tags}(\psi_0)$ actually exists, because of cases such as the following:

$$s_0 \preceq_\beta s_1,\ \psi_0 = X_0 : s_0, \text{ and } \psi_1 = Y_0 : s_1(f \to Y_1 : \top)$$

where $\gamma(G(\phi_1(Y_1))) = \gamma(G(Y_1 : \top))$ would be the trivial graph (with main sort $\top$) obtained by applying $f^{\mathcal{G}}$ to $G(\psi_0)$ (recall Definition 4.17). Clearly $\psi_0 \preceq_\beta \psi_1$[2], but we cannot define an h that satisfies Definition 4.47. This is why the statement of the lemma mentions a term

[2]Note that, for any OSF term $\psi = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$, $[\![\psi]\!]^{\mathcal{I}} = [\![X : s(f_1 \to t_1, \ldots, f_n \to t_n, f \to \top)]\!]^{\mathcal{I}}$, for any $f \in \mathcal{F}$.

ψ_0' that is equivalent to ψ_0, where the required subterms of ψ_0 corresponding to such trivial graphs would be made explicit (in the latter case ψ_0' would be $X_0 : s_0(f \to X : \top)$). The details concerning the general construction of such ψ_0' are left to the reader. In the following we simply assume that ψ_0 contains the variables needed to define h as above.

Now we show that the function h witnesses the fact that $\trianglelefteq(\psi_0, \psi_1) \geq \beta$.

- Clearly $h(\mathit{RootTag}(\psi_1)) = \mathit{RootTag}(\psi_0)$, since $\gamma(g_1) = g_0$, and h has been defined accordingly.
- Suppose that $Y_0 \xrightarrow{f}_{\psi_1} Y_1$. According to our definition of h:
  - let X_0 be the variable in ψ_0 such that $g_0^{X_0} = \gamma(g_1^{Y_0}))$, so that $h(Y_0) = X_0$; and
  - let X_1 be the variable in ψ_0 such that $g_0^{X_1} = \gamma(g_1^{Y_1})$, so that $h(Y_1) = X_1$.

  Since $Y_0 \xrightarrow{f}_{\psi_1} Y_1$, then $g_1^{Y_1} = f^{\mathcal{G}}(g_1^{Y_0})$ and, since γ is a β-morphism, then $\gamma(g_1^{Y_1}) = \gamma(f^{\mathcal{G}}(g_1^{Y_0})) = f^{\mathcal{G}}(\gamma(g_1^{Y_0}))$, and thus $g_0^{X_1} = f^{\mathcal{G}}(g_0^{X_0})$. By construction of g_0 this means that $X_0 \xrightarrow{f}_{\psi_0} X_1$, i.e.,. $h(Y_0) \xrightarrow{f}_{\psi_0} h(Y_1)$ as desired.
- Now let $Y \in \mathit{Tags}(\psi_1)$ be arbitrary. According to our definition of h, let X be the variable in ψ_0 such that $g_0^X = \gamma(g_1^Y)$, so that $h(Y) = X$. Let $s_0 = \mathit{Sort}_{\psi_0}(X)$ and $s_1 = \mathit{Sort}_{\psi_1}(Y)$. Since γ is a β-morphism, then $s_1^{\mathcal{G}}(g_1^Y) \wedge \beta \leq s_1^{\mathcal{G}}(\gamma(g_1^Y)) = s_1^{\mathcal{G}}(g_0^X)$. Note that $s_1^{\mathcal{G}}(g_1^Y) = \preceq(s_1, s_1) = 1$ and $s_1^{\mathcal{G}}(g_0^X) = \preceq(s_0, s_1)$, so the previous inequation simplifies to $\beta \leq \preceq(s_0, s_1)$.

  Since $Y \in \mathit{Tags}(\psi_1)$ was arbitrary, we have shown that

  $$\preceq(\mathit{Sort}_{\psi_0}(h(Y)), \mathit{Sort}_{\psi_1}(Y)) \geq \beta$$

  is true for all $Y \in \mathit{Tags}(\psi_1)$, and thus $\min(\{\preceq(\mathit{Sort}_{\psi_0}(h(Y)), \mathit{Sort}_{\psi_1}(Y)) \mid Y \in \mathit{Tags}(\psi_1)\}) \geq \beta$, showing indeed that $\trianglelefteq(\psi_0, \psi_1) \geq \beta$. □

Theorem 4.48 (Semantic and syntactic fuzzy subsumption). *Let ψ_0 and ψ_1 be two normal OSF terms. Then, for all $\beta \in (0, 1]$: $\psi_0 \preceq_\beta \psi_1$ if and only if there are two (normal) OSF terms ψ_0' and ψ_1' such that $\psi_0 \equiv \psi_0'$, $\psi_1 \equiv \psi_1'$, and $\psi_0' \trianglelefteq_\beta \psi_1'$.*

Proof of Theorem 4.48. This is a consequence of Lemma A.4 and the following fact.

Claim A.5. Let ψ_0 and ψ_1 be normal OSF terms. If

- there exist ψ_0' and ψ_1' such that $\psi_0' \equiv \psi_0$ and $\psi_1' \equiv \psi_1$ and $\psi_0' \trianglelefteq_{\beta'} \psi_1'$ with $\beta' > 0$, and
- there exist ψ_0'' and ψ_1'' such that $\psi_0'' \equiv \psi_0$ and $\psi_1'' \equiv \psi_1$ and $\psi_0'' \trianglelefteq_{\beta''} \psi_1''$ with $\beta'' > 0$,

then $\beta' = \beta''$.

The reason why this holds is that, if ψ_i, ψ_i' and ψ_i'' ($i \in \{0,1\}$) are equivalent, then they are essentially the same term, except that possibly one term may contain a subterm with a feature f pointing at the sort $\top$, and this subterm is not present in the other equivalent terms (see also Definition 4.37 and the proof of Lemma A.4). The existence of such trivial subterms does not however affect the degree of the syntactic subsumption.

Let us now prove the statement of the theorem. If $\preceq(\psi_0, \psi_1) = \beta$ then by Lemma A.4 there exist $\psi_0' \equiv \psi_0$ and $\psi_1' \equiv \psi_1$ such that $\trianglelefteq(\psi_0', \psi_1') \geq \beta$, i.e., $\psi_0' \trianglelefteq_{\beta'} \psi_1'$ and $\beta' \geq \beta$. Then also $\trianglelefteq(\psi_0', \psi_1') \geq \beta'$, and thus by Lemma A.4 $\preceq(\psi_0, \psi_1) \geq \beta'$, i.e., $\beta' \leq \beta$, and thus $\beta' = \beta$.

Now suppose that there exist $\psi_0' \equiv \psi_0$ and $\psi_1' \equiv \psi_1$ such that $\psi_0' \trianglelefteq_\beta \psi_1'$, so that $\trianglelefteq(\psi_0', \psi_1') \geq \beta$. Then by Lemma A.4 $\preceq(\psi_0, \psi_1) \geq \beta$. To prove that $\preceq(\psi_0, \psi_1) \leq \beta$, let us suppose otherwise, i.e., that $\preceq(\psi_0, \psi_1) = \beta' > \beta$. Then by Lemma A.4 there exist $\psi_0'' \equiv \psi_0$ and $\psi_1'' \equiv \psi_1$ such that $\trianglelefteq(\psi_0'', \psi_1'') \geq \beta'$, i.e., $\psi_0'' \trianglelefteq_{\beta''} \psi_1''$ and $\beta'' \geq \beta' > \beta$. But by Claim A.5 $\beta'' = \beta$, which is the desired contradiction. □

Theorem 4.49 (Crisp and fuzzy subsumption). *For all normal OSF terms ψ_1 and ψ_2: $\psi_1 \preceq \psi_2$ if and only if $\preceq(\psi_1, \psi_2) > 0$.*

Proof of Theorem 4.49. We employ the following notation for the rest of the proof: for an OSF graph $g = (N, E, \lambda_N, \lambda_E, X)$, we let $s_g \stackrel{\text{def}}{=} \lambda_N(X)$, i.e., s_g is the label of the root of g.

In the rest of the proof we rely on definitions, results and notation from crisp OSF logic [10]. In particular, $\preceq$ denotes the crisp subsumption ordering on sorts and OSF terms, and $\sqsubseteq^{\mathcal{J}}$ is the crisp approximation ordering on OSF graphs.

To better distinguish between the crisp and fuzzy contexts in the proof, we use $\mathcal{J}$ for the *crisp* OSF graph algebra [10], and $\mathcal{G}$ for the *fuzzy* OSF graph algebra. Note that the domains of these two structures and the definition of $f^{\mathcal{J}}$ and $f^{\mathcal{G}}$ are the same. The only difference is the interpretation of sort symbols, since, for each $s \in \mathcal{S}$: $s^{\mathcal{J}}$ is the set of graphs g such that $s_g \preceq s$, while $s^{\mathcal{G}}$ is a fuzzy set defined by letting $s^{\mathcal{G}}(g) = \preceq(s_g, s)$. Note that it follows that $g \in s^{\mathcal{J}}$ if and only if $s^{\mathcal{G}}(g) > 0$.

Also note that, given a graph g, the subalgebra $\mathcal{J}[g]$ and the fuzzy subalgebra $\mathcal{G}[g]$ have the same domain and feature symbols are interpreted in the same way. The only difference is again the interpretation of sort symbols, since, for each $s \in \mathcal{S}$: $s^{\mathcal{J}[g]} = s^{\mathcal{J}} \cap \mathcal{F}^*(g)$ and $s^{\mathcal{G}[g]} = s^{\mathcal{G}} \cap \mathbf{1}_{\mathcal{F}^*(g)}$. Note that it holds that $g' \in s^{\mathcal{J}[g]}$ if and only if $s^{\mathcal{G}[g]}(g') > 0$.

Let $g_1 = G(\psi_1)$ and $g_2 = G(\psi_2)$. Suppose that $\psi_1 \preceq \psi_2$, so that $g_2 \sqsubseteq^{\mathcal{J}} g_1$ [10], meaning that there is a function $\gamma : \Delta^{\mathcal{J}[g_2]} \to \Delta^{\mathcal{J}[g_1]}$ such that

- $\gamma(g_2) = g_1$;
- $\forall f \in \mathcal{F}$, $\forall g \in \mathcal{J}[g_2]$: $\gamma(f^{\mathcal{J}}(g)) = f^{\mathcal{J}}(\gamma(g))$; and
- $\forall s \in \mathcal{S}$, $\forall g \in \mathcal{J}[g_2]$: if $g \in s^{\mathcal{J}}$, then $\gamma(g) \in s^{\mathcal{J}}$.

Note that, equivalently, the last condition expresses that $\forall s \in \mathcal{S}$, $\forall g \in \mathcal{G}[g_2]$:

$$\text{if } s^{\mathcal{G}}(g) > 0, \text{ then } s^{\mathcal{G}}(\gamma(g)) > 0. \tag{A.1}$$

We prove that there is a $\beta > 0$ such that $\forall s \in \mathcal{S}$ and $\forall g \in \mathcal{G}[g_2]$: $s^{\mathcal{G}}(g) \wedge \beta \leq s^{\mathcal{G}}(\gamma(g))$.

Note that $\mathcal{G}[g_2]$ contains a finite number of graphs with root label different from $\top$, i.e., at most the ones corresponding to the variables of $\phi(\psi_2)$. Thus the set $B \stackrel{\text{def}}{=} \{\preceq(s_{\gamma(g)}, s_g) \mid s_g \neq \top, g \in \mathcal{G}[g_2]\}$ is finite, so that $\beta \stackrel{\text{def}}{=} \min(B)$ exists. Note that $\beta = \min(\{\preceq(s_{\gamma(g)}, s_g) \mid g \in \mathcal{G}[g_2]\})$, since if $s_g = \top$, then $\preceq(s_{\gamma(g)}, s_g) = 1$ by definition. Moreover, for every $g \in \mathcal{G}[g_2]$, $\preceq(s_{\gamma(g)}, s_g) > 0$: indeed $s_g^{\mathcal{G}}(g) = \preceq(s_g, s_g) = 1 > 0$ so that by (A.1) $s_g^{\mathcal{G}}(\gamma(g)) = \preceq(s_{\gamma(g)}, s_g) > 0$. It follows that $\beta > 0$. Finally, let $g \in \mathcal{G}[g_2]$ and $s \in \mathcal{S}$. Then

$$\begin{array}{rcccl} s^{\mathcal{G}}(g) \wedge \beta & = & \preceq(s_g, s) \wedge \beta & \leq & \preceq(s_g, s) \wedge \preceq(s_{\gamma(g)}, s_g) \\ & \leq & \preceq(s_{\gamma(g)}, s) & = & s^{\mathcal{G}}(\gamma(g)) \end{array}$$

by transitivity of $\preceq$. Thus γ is a β-morphism $\gamma : \mathcal{G}[g_2] \to \mathcal{G}[g_1]$ such that $\gamma(g_2) = g_1$. Then $\sqsubseteq^{\mathcal{G}}(g_2, g_1) = \beta' \geq \beta > 0$, so that by Theorem 4.45 $\preceq(\psi_1, \psi_2) > 0$.

For the other direction, assume that $\psi_1 \preceq_\beta \psi_2$ for some $\beta > 0$. Then $g_2 \sqsubseteq^{\mathcal{G}}_\beta g_1$, meaning that there is a function $\gamma : \Delta^{\mathcal{G}[g_2]} \to \Delta^{\mathcal{G}[g_1]}$ such that

- $\gamma(g_2) = g_1$;
- $\forall f \in \mathcal{F}$, $\forall g \in \mathcal{G}[g_2]$: $\gamma(f^{\mathcal{G}}(g)) = f^{\mathcal{G}}(\gamma(g))$; and
- $\forall s \in \mathcal{S}$, $\forall g \in \mathcal{G}[g_2]$: $s^{\mathcal{G}}(g) \wedge \beta \leq s^{\mathcal{G}}(\gamma(g))$.

Let s and g be arbitrary and suppose that $g \in s^{\mathcal{J}}$, so that $s^{\mathcal{G}}(g) > 0$. Since $s^{\mathcal{G}}(g) \wedge \beta \leq s^{\mathcal{G}}(\gamma(g))$, then $s^{\mathcal{G}}(\gamma(g)) > 0$, and so $\gamma(g) \in s^{\mathcal{J}}$. Thus γ is an OSF algebra morphism [10] $\gamma : \mathcal{J}[g_2] \to \mathcal{J}[g_1]$ such that $\gamma(g_2) = g_1$, i.e., $g_2 \sqsubseteq^{\mathcal{J}} g_1$, and equivalently $\psi_1 \preceq \psi_2$ as desired. □

A.1.3 Proofs for Section 4.7: Fuzzy OSF term unification

Theorem 4.53 (Correctness of Algorithm 5). *Let ψ_1 and ψ_2 be normal OSF terms. If (ψ, β) is the output of the procedure* UNIFY(ψ_1, ψ_2) *of Algorithm 5, then $\psi = \psi_1 \curlywedge_\beta \psi_2$.*

Proof of Theorem 4.53. The fact that $\psi = \psi_1 \curlywedge \psi_2$ is given by Theorem 4.52. Let us assume that ψ is not the inconsistent clause, as otherwise $\beta = 1$ and the result follows immediately.

For $i \in \{1, 2\}$, let $h_i : \mathit{Tags}(\psi_i) \to \mathit{Tags}(\psi)$ be the mapping defined by letting $h_i(X) = Z_{[X]}$ for all $X \in \mathit{Tags}(\psi_i)$. Then h_1 witnesses the syntactic subsumption $\psi \trianglelefteq_{\beta_1} \psi_1$, as it satisfies the three conditions of Definition 4.47.

1. Let $\mathit{RootTag}(\psi_1) = X_1$. Then by the construction of $\psi = \psi(\phi')$ on Line 13 and the fact that ϕ' is rooted in $Z_{[X_1]}$ it holds that $h_1(\mathit{RootTag}(\psi_1)) = h_1(X_1) = Z_{[X_1]} = \mathit{RootTag}(\psi)$.

2. Suppose that $X \xrightarrow{f}_{\psi_1} Y$. Then ϕ on Line 2 contains the constraint $X.f \doteq Y$. Throughout the application of the constraint normalization rules of Fig. 4.2 the variables X and Y of this constraint may be substituted with other variables, possibly multiple times. All of these substitutions are witnessed by equality constraints contained in ϕ after the loop of Lines 3 and 4. By the definition of Eq on Line 9, after the loop of Lines 11 and 12 the clause ϕ' must contain the constraint $Z_{[X]}.f \to Z_{[Y]}$. Finally, by the construction of $\psi = \psi(\phi')$ on Line 13, it holds that $Z_{[X]} \xrightarrow{f}_{\psi} Z_{[Y]}$, i.e., $h_1(X) \xrightarrow{f}_{\psi} h_1(Y)$, as required.

3. Since $h_1(X) = Z_{[X]}$ for all $X \in \mathit{Tags}(\psi_1)$, the computation of β_1 on Line 14 is carried out in the same way as the definition of the syntactic subsumption degree in Definition 4.47.

It can be proved analogously that h_2 witnesses the syntactic subsumption $\psi \trianglelefteq_{\beta_2} \psi_2$.

It follows by Theorem 4.48 that $\preceq(\psi, \psi_1) = \beta_1$ and $\preceq(\psi, \psi_2) = \beta_2$, and, by Definition 4.51, that $\beta = \min(\beta_1, \beta_2)$ is the unification degree of ψ_1 and ψ_2. □

A.1.4 Proofs for Section 4.8: Implementation

Proposition 4.55 (Support and reflexive-transitive closure). *If $\lhd$ is a fuzzy binary relation on a set $\mathcal{S}$, then $|\lhd|^* = |\lhd^{\circledast}|$.*

Proof of Proposition 4.55. Note that $|\lhd|^* \stackrel{\text{def}}{=} |\lhd|^+ \cup \{(x,x) \mid x \in X\}$, while $\lhd^{\circledast}(x,y) \stackrel{\text{def}}{=} 1$ if $x = y$, and $\lhd^{\circledast}(x,y) \stackrel{\text{def}}{=} \lhd^{\oplus}(x,y)$ otherwise. To prove the statement of the proposition it is then sufficient to show that $|\lhd^{\oplus}| = |\lhd|^+$. This is a consequence of the fact that, for all $i \geq 1$, $|\lhd^i| = |\lhd|^i$, which can be proved by a simple induction. □

Proposition 4.58 (MLBs in crisp and fuzzy posets). *Let $(\mathcal{S}, \preceq)$ be a fuzzy poset and let $\preceq \stackrel{\text{def}}{=} |\preceq|$. For all $S \subseteq \mathcal{S}$, $S^{fmlb}_{\preceq} = S^{mlb}_{\preceq}$.*

Proof of Proposition 4.58. The statement follows from the following facts, which are immediate consequences of Definitions 1.6, 1.11, 1.37 and 1.38: for all $S \subseteq \mathcal{S}$, (i) $S^{l}_{\preceq} = S^{fl}_{\preceq}$, and (ii) $\lceil S \rceil = \lceil S \rceil^{\cdot}$. □

A.2 Proofs for Chapter 5: Similarity-Based Reasoning with Order-Sorted Feature Logic

A.2.1 Proofs for Section 5.2: Fuzzy similarity-subsumption preorder

Proposition 5.5 (Equivalent definition of $\precsim$). *Let $(\mathcal{S}, \preceq)$ be a partial order[3], let $\sim$ be a similarity relation on $\mathcal{S}$, and let $\precsim$ be as in Definition 5.4. Then $\precsim = (\preceq \circ \sim)^{\oplus}$.*

Proof of Proposition 5.5. Let $s, s' \in \mathcal{S}$, and, for brevity, let $R \stackrel{\text{def}}{=} \preceq \circ \sim$. The fact that $\precsim(s, s') = R^{\oplus}(s, s')$ is given by the following equivalences:

$$
\begin{aligned}
\precsim(s, s') = \beta \quad &\Leftrightarrow_1 \quad \beta \text{ is the maximum such that there exists a sequence of sorts} \\
&\qquad s \sim_{\beta_0} s_0 \preceq t_1 \sim_{\beta_1} s_1 \preceq t_2 \cdots s_{p-1} \preceq t_p \sim_{\beta_p} s' \\
&\qquad \text{with } \beta = \min_{0 \le i \le p} \beta_i \\
&\Leftrightarrow_2 \quad \beta \text{ is the maximum such that there exists a sequence of sorts} \\
&\qquad s \; R_{\beta'_0} \; s'_0 \; R_{\beta'_1} \; s'_1 \ldots s'_{n-1} \; R_{\beta'_n} \; s' \text{ with } \beta = \min_{0 \le i \le n} \beta'_i \\
&\Leftrightarrow_3 \quad (\exists n \; R^n(s, s') = \beta) \text{ and } (\forall m \ge 1, \; R^m(s, s') \le \beta) \\
&\Leftrightarrow_4 \quad R^{\oplus} \stackrel{\text{def}}{=} \dot{\bigcup}_{m \ge 1} R^m = \beta
\end{aligned}
$$

The equivalences are given by:

1. Definition 5.4;
2. $R \stackrel{\text{def}}{=} (\preceq \circ \sim)$;
3. Definition 1.29 and $\mathcal{S}$ is finite;
4. Definition 1.30 and $\mathcal{S}$ is finite. □

Proposition 5.7 (Similarity-subsumption preorder). *The fuzzy relation $\precsim$ of Definition 5.4 is a fuzzy preorder.*

Proof of Proposition 5.7. Reflexivity holds since $s \sim_1 s$ for all $s \in \mathcal{S}$.

For transitivity, let $s_0, s_1, s_2 \in \mathcal{S}$ be such that

$$s_0 \precsim_{\beta_0} s_1, \;\; s_1 \precsim_{\beta_1} s_2, \text{ and } s_0 \precsim_{\beta} s_2.$$

We want to prove that $\beta \ge \min(\beta_0, \beta_1)$. By Definition 5.4 there exists chains

$$
\begin{aligned}
s_0 \sim_{\beta_{(0,0)}} s_{(0,0)} \preceq \cdots \preceq s'_{(0,n)} \sim_{\beta_{(0,n)}} s_1, \\
s_1 \sim_{\beta_{(1,0)}} s_{(1,0)} \preceq \cdots \preceq s'_{(1,m)} \sim_{\beta_{(1,m)}} s_2
\end{aligned}
$$

such that $\beta_0 = \min_{0 \le i \le n} \beta_{(0,i)}$ and $\beta_1 = \min_{0 \le i \le m} \beta_{(1,i)}$. By joining them, these two chains constitute a single similarity-subsumption chain of strength $\min(\beta_0, \beta_1)$ from s_0 to s_2. By Definition 5.4, $\beta \ge \min(\beta_0, \beta_1)$. □

A.2.2 Proofs for Section 5.3: Fuzzy similarity-subsumption partial order

Proposition 5.11 (Fuzzy preorder $\precsim$). *The fuzzy relation $\precsim$ of Definition 5.8 is a fuzzy preorder.*

Proof of Proposition 5.11. Reflexivity is given by $\preceq$, and transitivity by the transitive closure. □

Proposition 5.14 (Fuzzy partial order $\precsim$ on $\mathcal{S}_{/\approx}$). *The fuzzy relation $\preceq$ of Definition 5.12 is a fuzzy partial order on $\mathcal{S}_{/\approx}$.*

Proof of Proposition 5.14. First of all, note that, for all $x, x' \in \mathcal{S}$, whenever $x \in [x']$, then $\preceq(x, x') > 0$, $\preceq(x', x) > 0$, and $\beta_x = \beta_{x'}$. Also note that $\beta_x > 0$ for all $x \in \mathcal{S}$.

Reflexivity holds by definition.

For antisymmetry, let (a) $\preceq([x],[y]) > 0$ and (b) $\preceq([y],[x]) > 0$. We want to show that $[x] = [y]$. Let $x' \in [x]$, and we want to show that $x' \in [y]$. By (a), there are some $x'' \in [x]$ and $y'' \in [y]$ such that $\preceq(x'', y'') > 0$. By (b), there are some $x''' \in [x]$ and $y''' \in [y]$ such that $\preceq(y''', x''') > 0$. Then $x' \preceq x'' \preceq y'' \preceq y$ and $y \preceq y''' \preceq x''' \preceq x'$. Thus $x' \in [y]$. The fact that $[y] \subseteq [x]$ is proved similarly.

For transitivity, let

$$\preceq([x],[y]) = \beta_x \wedge \beta_y \wedge \bigvee_{x' \in [x], y' \in [y]} \preceq(x', y') = \beta_0, \text{ and}$$
$$\preceq([y],[z]) = \beta_y \wedge \beta_z \wedge \bigvee_{y' \in [y], z' \in [z]} \preceq(y', z') = \beta_1.$$

We want to show that $\preceq([x],[z]) \geq \beta_0 \wedge \beta_1$.

Since $\mathcal{S}$ is finite, let $x_0 \in [x]$, $y_0, y_1 \in [y]$, and $z_1 \in [z]$ be such that

$$\text{(c) } \preceq(x_0, y_0) = \bigvee_{x' \in [x], y' \in [y]} \preceq(x', y') \text{ and (d) } \preceq(y_1, z_1) = \bigvee_{y' \in [y], z' \in [z]} \preceq(y', z').$$

Then:

$$\begin{aligned}
\bigvee_{x' \in [x], z' \in [z]} \preceq(x', z') &\geq_1 \preceq(x_0, z_1) \geq_2 \preceq(x_0, y_0) \wedge \preceq(y_0, z_1) \\
&\geq_3 \preceq(x_0, y_0) \wedge \preceq(y_0, y_1) \wedge \preceq(y_1, z_1) \\
&\geq_4 \preceq(x_0, y_0) \wedge \beta_y \wedge \preceq(y_1, z_1) \\
&\geq_5 \bigvee_{x' \in [x], y' \in [y]} \preceq(x', y') \wedge \beta_y \wedge \bigvee_{y' \in [y], z' \in [z]} \preceq(y', z').
\end{aligned}$$

The inequalities are given by:

1. Definition of the t-conorm $\vee$, $x_0 \in [x]$, and $z_1 \in [z]$;
2. Max-min transitivity of $\preceq$;
3. Max-min transitivity of $\preceq$;
4. Definition of β_y (Definition 5.12), $y_0, y_1 \in [y]$;
5. (c) and (d).

Finally:

$$\begin{aligned}\preceq([x],[z]) &\overset{\text{def}}{=} \bigvee_{x'\in[x],z'\in[z]} \preceq(x',z') \wedge \beta_x \wedge \beta_z \\ &\geq \bigvee_{x'\in[x],y'\in[y]} \preceq(x',y') \wedge \beta_y \wedge \bigvee_{y'\in[y],z'\in[z]} \preceq(y',z') \wedge \beta_x \wedge \beta_z = \beta_0 \wedge \beta_1. \qquad \square\end{aligned}$$

A.2.3 Proofs for Section 5.4: Fuzzy similarity-subsumption lattice

Proposition 5.17 (Fuzzy antichain lattice). *If $(\mathcal{S}, \preceq)$ is a fuzzy poset, then $(Antichains(\mathcal{S}), \preceq)$ from Definition 5.16 is a fuzzy lattice, where the GLB of $C, C' \in Antichains(\mathcal{S})$ is given by $C_1 \curlywedge C_2 \overset{\text{def}}{=} \lceil C_1{\downarrow} \cap C_2{\downarrow} \rceil^{\cdot}$. Moreover, $(Antichains(\mathcal{S}), \preceq)$ preserves the GLBs that exist in $(\mathcal{S}, \preceq)$, i.e., if $s_0 \curlywedge s_1 = s$, then $\{s_0\} \curlywedge \{s_1\} = \{s\}$.*

Proof of Proposition 5.17. First of all, note that the following holds by Definition 5.16:

$$\preceq(C, C') \geq \beta \quad \Leftrightarrow \quad \forall c \in C \ \exists c' \in C' \text{ such that } \preceq(c, c') \geq \beta.$$

This fact will be used throughout the proof.

Reflexivity of $(Antichains(\mathcal{S}), \preceq)$ is obvious, since, for any $C \in Antichains(\mathcal{S})$, for all $c \in C$: $\preceq(c, c) = 1$.

For transitivity, suppose that (i) $\preceq(C_0, C_1) = \beta_0$ and (ii) $\preceq(C_1, C_2) = \beta_1$. To show that $\preceq(C_0, C_2) \geq \min(\beta_0, \beta_1)$ it suffices to show that for any $c_0 \in C_0$ there exists a $c_2 \in C_2$ such that $\preceq(c_0, c_2) \geq \min(\beta_0, \beta_1)$. Thus, let $c_0 \in C_0$ be arbitrary. Then by (i) there is some $c_1 \in C_1$ such that $\preceq(c_0, c_1) \geq \beta_0$, and by (ii) there is some $c_2 \in C_2$ such that $\preceq(c_1, c_2) \geq \beta_1$. By max-min transitivity of $(\mathcal{S}, \preceq)$ then

$$\preceq(c_0, c_2) \geq \min(\preceq(c_0, c_1), \preceq(c_1, c_2)) \geq \min(\beta_0, \beta_1).$$

For antisymmetry, suppose that (iii) $\preceq(C_0, C_1) > 0$ and (iv) $\preceq(C_1, C_0) > 0$. To show that $C_0 = C_1$, let $c_0 \in C_0$. Then by (iii) there is some $c_1 \in C_1$ such that $\preceq(c_0, c_1) > 0$ and by (iv) there is some $c_2 \in C_0$ such that $\preceq(c_1, c_2) > 0$, implying $\preceq(c_0, c_2) > 0$. Because C_0 is an antichain, then $c_0 = c_2$, which implies $c_1 = c_0 = c_2$ by fuzzy antisymmetry of $(\mathcal{S}, \preceq)$, so that $c_0 \in C_1$. The other direction is proved similarly.

Now let $C_0, C_1 \in \mathit{Antichains}(\mathcal{S})$, and $C \stackrel{\text{def}}{=} \lceil C_0{\downarrow} \cap C_1{\downarrow} \rceil$. Then C is the GLB of C_0 and C_1 in $(\mathit{Antichains}(\mathcal{S}), \preceq)$.

- To prove that $C \preceq C_0$ it suffices to show that for any $c \in C$ there is some $c_0 \in C_0$ such that $\preceq(c, c_0) > 0$. This immediately follows from the fact that if $c \in C$, then in particular $c \in C_0{\downarrow}$. The fact that $C \preceq C_1$ is proved similarly.

- Suppose $C' \in \mathit{Antichains}(\mathcal{S})$ is such that (v) $C' \preceq C_0$ and (vi) $C' \preceq C_1$. To show that $C' \preceq C$, it is sufficient to show that for any $c' \in C'$ there is some $c \in C$ such that $\preceq(c', c) > 0$. Hence let $c' \in C'$ be arbitrary. By (v) and (vi) there are some $c_0 \in C_0$ and $c_1 \in C_1$ such that $c' \preceq c_0$ and $c' \preceq c_1$. Then $c' \in C_0{\downarrow} \cap C_1{\downarrow}$. Then by definition of C there is some $c \in C$ (possibly equal to c') such that $c' \preceq c$.

The map $s_0 \mapsto \{s_0\}$ also satisfies the property whereby, if $\preceq(s, s') = \beta$ in $(\mathcal{S}, \preceq)$, then $\preceq(\{s\}, \{s'\}) = \beta$ in $(\mathit{Antichains}(\mathcal{S}), \preceq)$.

Finally, suppose s is the GLB of s_0, s_1 in $(\mathcal{S}, \preceq)$. Then $\{s_0\} \curlywedge \{s_1\} = \lceil s_0{\downarrow} \cap s_1{\downarrow} \rceil = \{s\}$. □

Bibliography

[1] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[2] Hassan Aït-Kaci. "A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures". PhD thesis. University of Pennsylvania, 1984.

[3] Hassan Aït-Kaci. *A Set-Complete Construction for Order-Sorted Set-Valued Features*. Tech. rep. CEDAR Technical Report Number 11, CEDAR Project. Villeurbanne, France: LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, 2014.

[4] Hassan Aït-Kaci. "Data models as constraint systems: a key to the semantic web". In: *Constraint Processing Letters* 1 (2007).

[5] Hassan Aït-Kaci and Samir Amir. *Classifying and Querying Very Large Taxonomies*. Tech. rep. CEDAR Technical Report Number 2, CEDAR Project. Villeurbanne, France: LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, 2013.

[6] Hassan Aït-Kaci and Samir Amir. "Classifying and querying very large taxonomies with bit-vector encoding". In: *J. Intell. Inf. Syst.* 48.1 (2017), pp. 1–25. DOI: 10.1007/s10844-015-0383-2. URL: https://doi.org/10.1007/s10844-015-0383-2.

[7] Hassan Aït-Kaci and Patrick Lincoln. "LIFE — A Natural Language for Natural Language". In: *TAL Traitement Automatique des Langues* 30 (Jan. 1989), pp. 37–67.

[8] Hassan Aït-Kaci and Roger Nasr. "Login: a logic programming language with built-in inheritance". In: *J. Log. Program.* 3.3 (1986), pp. 185–215. ISSN: 0743-1066. DOI: https://doi.org/10.1016/0743-1066(86)90013-0. URL: http://www.sciencedirect.com/science/article/pii/0743106686900130.

[9] Hassan Aït-Kaci and Gabriella Pasi. "Fuzzy lattice operations on first-order terms over signatures with similar constructors: A constraint-based approach". In: *Fuzzy Sets Syst.* 391 (2020). Computer Science, pp. 1–46. ISSN: 0165-0114. DOI: https:

//doi.org/10.1016/j.fss.2019.03.019. URL: http://www.sciencedirect.com/science/article/pii/S016501141930199X.

[10] Hassan Aït-Kaci and Andreas Podelski. "Towards a meaning of life". In: *J. Log. Program.* 16.3 (1993), pp. 195–234. ISSN: 0743-1066. DOI: https://doi.org/10.1016/0743-1066(93)90043-G. URL: http://www.sciencedirect.com/science/article/pii/074310669390043G.

[11] Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. "Order-sorted feature theory unification". In: *The Journal of Logic Programming* 30.2 (1997), pp. 99–124. ISSN: 0743-1066. DOI: https://doi.org/10.1016/S0743-1066(96)00053-2. URL: http://www.sciencedirect.com/science/article/pii/S0743106696000532.

[12] Hassan Aït-Kaci and Yutaka Sasaki. "An axiomatic approach to feature term generalization". In: *European Conference on Machine Learning*. Springer. 2001, pp. 1–12.

[13] Hassan Aït-Kaci et al. "Efficient Implementation of Lattice Operations". In: *ACM Trans. Program. Lang. Syst.* 11.1 (Jan. 1989), pp. 115–146. ISSN: 0164-0925. DOI: 10.1145/59287.59293. URL: https://doi.org/10.1145/59287.59293.

[14] Hassan Aït-Kaci et al. "The Wild LIFE Handbook". In: 1994.

[15] Samir Amir and Hassan Aït-Kaci. "An Efficient and Large-Scale Reasoning Method for the Semantic Web". In: *J. Intell. Inf. Syst.* 48.3 (June 2017), pp. 653–674. ISSN: 0925-9902. DOI: 10.1007/s10844-016-0435-2. URL: https://doi.org/10.1007/s10844-016-0435-2.

[16] Samir Amir and Hassan Aït-Kaci. *Design and Implementation of an Efficient Semantic Web Reasoner*. Tech. rep. CEDAR Technical Report Number 12, CEDAR Project. Villeurbanne, France: LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, 2014.

[17] Francesca Arcelli-Fontana and Ferrante Formato. "A Similarity-Based Resolution Rule". In: *International Journal of Intelligent Systems* 17 (2002), pp. 853–872.

[18] Franz Baader, Ian Horrocks, and Ulrike Sattler. "Description Logics". In: *Handbook of Knowledge Representation*. Ed. by Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. Vol. 3. Foundations of Artificial Intelligence. Elsevier, 2008, pp. 135–179. DOI: https://doi.org/10.1016/S1574-6526(07)03003-9. URL: http://www.sciencedirect.com/science/article/pii/S1574652607030039.

[19] Franz Baader and Werner Nutt. "Basic description logics". In: *The description logic handbook: theory, implementation, and applications*. 2003, pp. 43–95.

[20] Franz Baader and Wayne Snyder. "Unification theory". In: *Handbook of automated reasoning* 1 (2001), pp. 445–532.

[21] Franz Baader et al. *Introduction to description logic*. Cambridge University Press, 2017.

[22] Michael A Bender and Martin Farach-Colton. "The LCA problem revisited". In: *LATIN 2000: Theoretical Informatics: 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000 Proceedings 4*. Springer. 2000, pp. 88–94.

[23] Michael A Bender et al. "Finding least common ancestors in directed acyclic graphs". In: *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. 2001, pp. 845–854.

[24] Michael A Bender et al. "Lowest common ancestors in trees and directed acyclic graphs". In: *Journal of Algorithms* 57.2 (2005), pp. 75–94.

[25] Tim Berners-Lee, James Hendler, and Olli Lassila. "The Semantic Web". In: *Scientific American* 284 (2001), p. 35.

[26] Fernando Bobillo, Miguel Delgado, and Juan Gómez-Romero. "DeLorean: A reasoner for fuzzy OWL 2". In: *Expert Systems with Applications* 39.1 (2012), pp. 258–272. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2011.07.016. URL: http://www.sciencedirect.com/science/article/pii/S095741741100978X.

[27] Fernando Bobillo and Umberto Straccia. "Fuzzy ontology representation using OWL 2". In: *International journal of approximate reasoning* 52.7 (2011), pp. 1073–1094.

[28] Fernando Bobillo and Umberto Straccia. "fuzzyDL: An Expressive Fuzzy Description Logic Reasoner". In: *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*. 2008, pp. 923–930.

[29] Fernando Bobillo and Umberto Straccia. "Optimising fuzzy description logic reasoners with general concept inclusion absorption". In: *Fuzzy Sets and Systems* 292 (2016). Special Issue in Honor of Francesc Esteva on the Occasion of his 70th Birthday, pp. 98–129. ISSN: 0165-0114. DOI: https://doi.org/10.1016/j.fss.2014.10.029. URL: http://www.sciencedirect.com/science/article/pii/S0165011414004850.

[30] Stefan Borgwardt and Rafael Peñaloza. "Fuzzy Description Logics – A Survey". In: *International Conference on Scalable Uncertainty Management*. Springer. 2017, pp. 31–45.

[31] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Elsevier, 2004.

[32] Esra Çakir and H Ziya Ulukan. "A Fuzzy Logic Programming Environment for Recycling Facility Selection." In: *IJCCI*. 2019, pp. 367–374.

[33] Esra Çakır and Ziya Ulukan. "A fuzzy linguistic programming for sustainable ecotourism activities". In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2020, pp. 0121–0126.

[34] Robert L. Carpenter. *The Logic of Typed Feature Structures: With Applications to Unification Grammars, Logic Programs and Constraint Resolution*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992. DOI: 10.1017/CBO9780511530098.

[35] Yves Caseau et al. "Encoding of multiple inheritance hierarchies and partial orders". In: *Computational Intelligence* 15.1 (1999), pp. 50–62.

[36] Inheung Chon. "Fuzzy partial order relations and fuzzy lattices". In: *Korean J. Math.* 17.4 (2009), pp. 361–374.

[37] Anthony G Cohn. "Taxonomic reasoning with many-sorted logics". In: *Artif. Intell. Rev.* 3.2-3 (1989), pp. 89–128.

[38] M. E. Cornejo, J. Medina Moreno, and C. Rubio-Manzano. "Towards a Full Fuzzy Unification in the Bousi∼Prolog system". In: *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2018, pp. 1–7. DOI: 10.1109/FUZZ-IEEE.2018.8491514.

[39] Artur Czumaj, Mirosław Kowaluk, and Andrzej Lingas. "Faster algorithms for finding lowest common ancestors in directed acyclic graphs". In: *Theoretical Computer Science* 380.1-2 (2007), pp. 37–46.

[40] Santanu Kumar Dash et al. "A scalable approach to computing representative lowest common ancestor in directed acyclic graphs". In: *Theoretical Computer Science* 513 (2013), pp. 25–37.

[41] Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[42] Kathrin Dentler et al. "Comparison of Reasoners for Large Ontologies in the OWL 2 EL Profile". In: *Semant. Web* 2.2 (Apr. 2011), pp. 71–87. ISSN: 1570-0844.

[43] J. Dörre and W. C. Rounds. "On subsumption and semiunification in feature algebras". In: *Fifth Annual IEEE Symposium on Logic in Computer Science*. 1990, pp. 300–310.

[44] Didier Dubois and Henri Prade. *Fuzzy Sets and Systems. Theory and Applications*. Vol. 144. Mathematics in Science and Engineering. Elsevier, 1980. URL: https://www.sciencedirect.com/bookseries/mathematics-in-science-and-engineering/vol/144/.

[45] Besik Dundua et al. "Constraint Solving over Multiple Similarity Relations". In: *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*. Ed. by Zena M. Ariola. Vol. 167. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 30:1–30:19. ISBN: 978-3-95977-155-9. DOI: 10.4230/LIPIcs.FSCD.2020.30. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12352.

[46] Martin C Emele and Rémi Zajac. "Typed unification grammars". In: *COLING 1990 Volume 3: Papers presented to the 13th International Conference on Computational Linguistics*. 1990.

[47] Johannes Fischer and Volker Heun. "Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE". In: *Annual Symposium on Combinatorial Pattern Matching*. Springer. 2006, pp. 36–48.

[48] Ferrante Formato, Giangiacomo Gerla, and Maria I Sessa. "Similarity-based unification". In: *Fundamenta Informaticae* 41.4 (2000), pp. 393–414. URL: http://www.dipmat2.unisa.it/people/gerla/www/Down/similarity-based.pdf.

[49] Deb Dutta Ganguly, Chilukuri K. Mohan, and Sanjay Ranka. "A space-and-time-efficient coding algorithm for lattice computations". In: *IEEE Transactions on Knowledge and Data Engineering* 6.5 (1994), pp. 819–829.

[50] Mingxia Gao and Chunnian Liu. "Extending OWL by fuzzy description logic". In: *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*. IEEE. 2005, 6–pp.

[51] Giangiacomo Gerla and Maria I. Sessa. "Similarity in Logic Programming". In: *Fuzzy Logic and Soft Computing*. Ed. by Guoqing Chen, Mingsheng Ying, and Kai-Yuan Cai. Boston, MA: Springer US, 1999, pp. 19–31. ISBN: 978-1-4615-5261-1. DOI: 10.1007/978-1-4615-5261-1_2. URL: https://doi.org/10.1007/978-1-4615-5261-1_2.

[52] Birte Glimm et al. "HermiT: an OWL 2 reasoner". In: *Journal of automated reasoning* 53 (2014), pp. 245–269.

[53] Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. "The foundational model of anatomy in OWL: Experience and perspectives". In: *Journal of Web Semantics* 4.3 (2006), pp. 181–195.

[54] John Goodwin. "Experiences of Using OWL at the Ordnance Survey". In: *In Proc. of OWL: Experiences and Directions*. 2005.

[55] Fabrizio Grandoni et al. "All-Pairs LCA in DAGs: Breaking through the O (n 2.5) barrier". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2021, pp. 273–289.

[56] Volker Haarslev et al. “The RacerPro knowledge representation and reasoning system”. In: *Semantic Web* 3.3 (2012), pp. 267–277.

[57] Frank van Harmelen et al. *Handbook of Knowledge Representation*. San Diego, CA, USA: Elsevier Science, 2007. ISBN: 0444522115.

[58] Frank W Hartel et al. “Modeling a description logic vocabulary for cancer research”. In: *Journal of biomedical informatics* 38.2 (2005), pp. 114–129.

[59] Jacques Herbrand. “Recherches sur la théorie de la démonstration”. PhD thesis. Faculté des scien es de l’université de Paris, Paris, France, 1930.

[60] Pascual Julián-Iranzo, Ginés Moreno, and José Antonio Riaza. “The Fuzzy Logic Programming language FASILL: Design and implementation”. In: *Int. J. Approx. Reason.* 125 (2020), pp. 139–168.

[61] Pascual Julián-Iranzo and Clemente Rubio-Manzano. “Proximity-based unification theory”. In: *Fuzzy Sets Syst.* 262 (2015). Theme: Logic and Computer Science, pp. 21–43. ISSN: 0165-0114. DOI: https://doi.org/10.1016/j.fss.2014.07.006. URL: http://www.sciencedirect.com/science/article/pii/S0165011414003133.

[62] Pascual Julián-Iranzo, Clemente Rubio-Manzano, and Juan Gallardo-Casero. “Bousi ∼Prolog: a Prolog Extension Language for Flexible Query Answering”. In: *Electronic Notes in Theoretical Computer Science* 248 (2009). Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008), pp. 131–147. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2009.07.064. URL: http://www.sciencedirect.com/science/article/pii/S1571066109002874.

[63] Pascual Julián-Iranzo and Fernando Sáenz-Pérez. “Bousi∼Prolog: Design and implementation of a proximity-based fuzzy logic programming language”. In: *Expert Syst. Appl.* 213 (2023), p. 118858. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.118858. URL: https://www.sciencedirect.com/science/article/pii/S0957417422018760.

[64] Pascual Julián-Iranzo and Fernando Sáenz-Pérez. “Proximity-based unification: an efficient implementation method”. In: *IEEE Trans. Fuzzy Syst.* 29.5 (2020), pp. 1238–1251.

[65] William R. Keller. *Feature Logics, Infinitary Descriptions and Grammar*. School of Cognitive and Computing Sciences, University of Sussex, 1993.

[66] Bernd Kiefer and Hans-Ulrich Krieger. “A bag of useful techniques for efficient and robust parsing”. In: (1998).

[67] Frank Klawonn. "Should fuzzy equality and similarity satisfy transitivity? Comments on the paper by M. De Cock and E. Kerre". In: *Fuzzy Sets and Systems* 133.2 (2003), pp. 175–180. ISSN: 0165-0114. DOI: https://doi.org/10.1016/S0165-0114(02)00243-9. URL: http://www.sciencedirect.com/science/article/pii/S0165011402002439.

[68] George Klir and Bo Yuan. *Fuzzy sets and fuzzy logic*. Vol. 4. Prentice hall New Jersey, 1995.

[69] Kevin Knight. "Unification: A multidisciplinary survey". In: *ACM Computing Surveys (CSUR)* 21.1 (1989), pp. 93–124.

[70] Hans-Ulrich Krieger and Ulrich Schäfer. "DL meet FL: a bidirectional mapping between ontologies and linguistic knowledge". In: *Coling 2010: Posters*. 2010, pp. 588–596.

[71] Temur Kutsia and Cleo Pau. "A Framework for Approximate Generalization in Quantitative Theories". In: *Automated Reasoning*. Ed. by Jasmin Blanchette, Laura Kovács, and Dirk Pattinson. Cham: Springer International Publishing, 2022, pp. 578–596. ISBN: 978-3-031-10769-6.

[72] Temur Kutsia and Cleo Pau. "Matching and Generalization Modulo Proximity and Tolerance Relations". In: *Language, Logic, and Computation*. Ed. by Aybüke Özgün and Yulia Zinova. Cham: Springer International Publishing, 2022, pp. 323–342. ISBN: 978-3-030-98479-3.

[73] Temur Kutsia and Cleo Pau. "Proximity-based generalization". In: *Proceedings of the 32nd International Workshop on Unification, UNIF*. Vol. 2018. 2018.

[74] Temur Kutsia and Cleo Pau. "Solving Proximity Constraints". In: *Logic-Based Program Synthesis and Transformation*. Ed. by Maurizio Gabbrielli. Cham: Springer International Publishing, 2020, pp. 107–122. ISBN: 978-3-030-45260-5.

[75] Michael J Lawley and Cyril Bousquet. "Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner". In: *Proc. 6th Australasian Ontology Workshop (IAOA'10). Conferences in Research and Practice in Information Technology*. Vol. 122. 2010, pp. 45–49.

[76] Vladimir Lifschitz, Leora Morgenstern, and David Plaisted. "Knowledge Representation and Classical Logic". In: *Handbook of Knowledge Representation*. Ed. by Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. Vol. 3. Foundations of Artificial Intelligence. Elsevier, 2008, pp. 3–88. DOI: https://doi.org/10.1016/S1574-6526(07)03001-5. URL: http://www.sciencedirect.com/science/article/pii/S1574652607030015.

[77] Thomas Lukasiewicz and Umberto Straccia. "Managing uncertainty and vagueness in description logics for the Semantic Web". In: *Journal of Web Semantics* 6.4 (2008). Semantic Web Challenge 2006/2007, pp. 291–308. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2008.04.001. URL: http://www.sciencedirect.com/science/article/pii/S1570826808000395.

[78] Alberto Martelli and Ugo Montanari. "An efficient unification algorithm". In: *ACM Trans. Program. Lang. Syst.* 4.2 (1982), pp. 258–282.

[79] I. Mezzomo, B. Bedregal, and R. H. N. Santiago. "Operations on bounded fuzzy lattices". In: *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*. 2013, pp. 151–156. DOI: 10.1109/IFSA-NAFIPS.2013.6608391.

[80] I. Mezzomo and B. C. Bedregal. "On fuzzy α-lattices". In: *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2016, pp. 775–781. DOI: 10.1109/FUZZ-IEEE.2016.7737766.

[81] Gian Carlo Milanese and Gabriella Pasi. "Conjunctive Reasoning on Fuzzy Taxonomies with Order-Sorted Feature Logic". In: *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. Luxembourg, Luxembourg, 2021, pp. 1–7. DOI: 10.1109/FUZZ45933.2021.9494474.

[82] Gian Carlo Milanese and Gabriella Pasi. "Fuzzy order-sorted feature logic". In: *Fuzzy Sets and Systems* 477 (2024), p. 108800. ISSN: 0165-0114. DOI: https://doi.org/10.1016/j.fss.2023.108800.

[83] Gian Carlo Milanese and Gabriella Pasi. "Fuzzy Order-Sorted Feature Term Unification". In: *Federated Logic Conference. 36th International Workshop on Unification*. Haifa, Israel, 2022. URL: https://www.cs.cas.cz/unif-2022/Papers/UNIF_2022_paper_3.pdf.

[84] Gian Carlo Milanese and Gabriella Pasi. "Similarity-Based Reasoning with Order-Sorted Feature Logic". In: *IEEE Transactions on Fuzzy Systems* (2024), pp. 1–14. DOI: 10.1109/TFUZZ.2024.3362897.

[85] Marvin Minsky. *A Framework for Representing Knowledge*. Tech. rep. USA, 1974.

[86] Kuniaki Mukai. *Anadic tuples in Prolog*. Tech. rep. TR-239. Tokyo, Japan: ICOT, 1987.

[87] Bernhard Nebel and Gert Smolka. "Attributive description formalisms ... and the rest of the world". In: *Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence Final Report on the IBM Germany LILOG-Project*. Ed. by Otthein Herzog and Claus-Rainer Rollinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 439–452. ISBN: 978-3-540-38493-9. DOI: 10.1007/3-540-54594-8_74. URL: https://doi.org/10.1007/3-540-54594-8_74.

[88] Bernhard Nebel and Gert Smolka. "Representation and reasoning with attributive descriptions". In: *Sorts and Types in Artificial Intelligence*. Ed. by Karl Hans Bläsius, Ulrich Hedtstück, and Claus-Rainer Rollinger. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 111–139. ISBN: 978-3-540-46965-0.

[89] Vilém Novák. "First-order fuzzy logic". In: *Studia logica* 46 (1987), pp. 87–109.

[90] Cleo Pau and Temur Kutsia. "Proximity-Based Unification and Matching for Fully Fuzzy Signatures". In: *2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2021. DOI: 10.1109/FUZZ45933.2021.9494438.

[91] Gordon D Plotkin. "A note on inductive generalization". In: *Mach. Intell.* 5 (1969), pp. 153–163.

[92] Alan Rector and Jeremy Rogers. "Ontological and practical issues in using a description logic to represent medical concept systems: Experience from GALEN". In: *Reasoning Web International Summer School*. Springer, 2006, pp. 197–231.

[93] John C Reynolds. "Transformational systems and algebraic structure of atomic formulas". In: *Mach. Intell.* 5 (1970), pp. 135–151.

[94] Francisco P Romero et al. "Classifying unlabeled short texts using a fuzzy declarative approach". In: *Language resources and evaluation* 47 (2013), pp. 151–178.

[95] Clemente Rubio-Manzano and Pascual Julián-Iranzo. "A fuzzy linguistic prolog and its applications". In: *Journal of Intelligent & Fuzzy Systems* 26.3 (2014), pp. 1503–1516.

[96] Clemente Rubio-Manzano and Pascual Julián-Iranzo. "Incorporation of abstraction capability in a logic-based framework by using proximity relations". In: *Journal of Intelligent & Fuzzy Systems* 29.4 (2015), pp. 1671–1683.

[97] Clemente Rubio-Manzano and Gracian Trivino. "Improving player experience in Computer Games by using players' behavior analysis and linguistic descriptions". In: *International Journal of Human-Computer Studies* 95 (2016), pp. 27–38.

[98] Alan Ruttenberg, Jonathan Rees, and Joanne Luciano. "Experience Using OWL DL for the Exchange of Biological Pathway Information." In: *OWLED*. Vol. 188. 2005.

[99] Sami H Al-Sayadi et al. "A fuzzy declarative approach to classify unlabeled short texts based on automatically constructed WordNet ontologies". In: *Computational Intelligence and Mathematics for Tackling Complex Problems 3*. Springer, 2022, pp. 157–164.

[100] Manfred Schmidt-Schauß and Gert Smolka. "Attributive concept descriptions with complements". In: *Artificial Intelligence* 48.1 (1991), pp. 1–26. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(91)90078-X. URL: http://www.sciencedirect.com/science/article/pii/000437029190078X.

[101] Maria I. Sessa. "Approximate reasoning by similarity-based SLD resolution". In: *Theor. Comput. Sci.* 275.1 (2002), pp. 389–426. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(01)00188-8. URL: http://www.sciencedirect.com/science/article/pii/S0304397501001888.

[102] Robert DC Shearer, Boris Motik, and Ian Horrocks. "Hermit: A highly-efficient OWL reasoner". In: *Owled*. Vol. 432. 2008, p. 91.

[103] Sujeet Shenoi and Austin Melton. "Proximity relations in the fuzzy relational database model". In: *Fuzzy Sets Syst.* 100 (1999), pp. 51–62. ISSN: 0165-0114. DOI: https://doi.org/10.1016/S0165-0114(99)80006-2. URL: http://www.sciencedirect.com/science/article/pii/S0165011499800062.

[104] Nikolaos Simou and Stefanos Kollias. "Fire: A fuzzy reasoning engine for impecise knowledge". In: *K-Space PhD Students Workshop, Berlin, Germany*. Vol. 14. Citeseer. 2007.

[105] Evren Sirin et al. "Pellet: A practical owl-dl reasoner". In: *Journal of Web Semantics* 5.2 (2007), pp. 51–53.

[106] Gert Smolka. "A Feature Logic with Subsorts". In: *LILOG-Report* 33 (1988).

[107] Gert Smolka. "Feature-constraint logics for unification grammars". In: *The Journal of Logic Programming* 12.1 (1992), pp. 51–87. ISSN: 0743-1066. DOI: https://doi.org/10.1016/0743-1066(92)90039-6. URL: https://www.sciencedirect.com/science/article/pii/0743106692900396.

[108] J. F. Sowa. "Conceptual Graphs for a Data Base Interface". In: *IBM Journal of Research and Development* 20.4 (1976), pp. 336–357.

[109] Kavitha Srinivas. "OWL Reasoning in the Real World: Searching for Godot". In: *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*. Ed. by Bernardo Cuenca Grau et al. Vol. 477. CEUR Workshop Proceedings. CEUR-WS.org, 2009. URL: http://ceur-ws.org/Vol-477/invited%5C_2.pdf.

[110] G. Stoilos et al. "Uncertainty and the Semantic Web". In: *IEEE Intelligent Systems* 21.5 (2006), pp. 84–87. DOI: 10.1109/MIS.2006.105.

[111] Giorgos Stoilos and Giorgos Stamou. "Reasoning with fuzzy extensions of OWL and OWL 2". In: *Knowledge and information systems* 40.1 (2014), pp. 205–242.

[112] Giorgos Stoilos, Giorgos Stamou, and Jeff Z Pan. "Fuzzy extensions of OWL: Logical properties and reduction to fuzzy description logics". In: *International journal of approximate reasoning* 51.6 (2010), pp. 656–679.

[113] Umberto Straccia. *Foundations of Fuzzy Logic and Semantic Web Languages*. New York: Chapman and Hall, 2014. DOI: https://doi.org/10.1201/b15460.

[114] Umberto Straccia. "Softfacts: A top-k retrieval engine for ontology mediated access to relational databases". In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2010, pp. 4115–4122.

[115] Edward Thomas, Jeff Z Pan, and Yuan Ren. "TrOWL: Tractable OWL 2 reasoning infrastructure". In: *The Semantic Web: Research and Applications: 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30–June 3, 2010, Proceedings, Part II 7*. Springer. 2010, pp. 431–435.

[116] Dmitry Tsarkov and Ian Horrocks. "FaCT++ Description Logic Reasoner: System Description". In: *Automated Reasoning*. Ed. by Ulrich Furbach and Natarajan Shankar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 292–297. ISBN: 978-3-540-37188-5.

[117] Dorothea Tsatsou et al. "LiFR: A Lightweight Fuzzy DL Reasoner". In: *The Semantic Web: ESWC 2014 Satellite Events*. Ed. by Valentina Presutti et al. Cham: Springer International Publishing, 2014, pp. 263–267. ISBN: 978-3-319-11955-7.

[118] Mingsheng Ying. "A logic for approximate reasoning". In: *The Journal of Symbolic Logic* 59.3 (1994), pp. 830–837.

[119] Rémi Zajac. "Inheritance and constraint-based grammar formalisms". In: *Computational Linguistics* 18.2 (1992), pp. 159–182.