Fuzzy Lattice–Theoretic Operations over Data and Knowledge Structures

Hassan AÏT-KACI Gabriella PASI

April 10, 2020

DRAFT—PLEASE DO NOT DISTRIBUTE

DRAFT—PLEASE DO NOT DISTRIBUTE

What is this about?

This explores the lattice-theoretic properties of the fuzzy processing of data and knowledge structures, such as First-Order Terms ($\mathcal{FOT}$s) and Order-Sorted Feature ($\mathcal{OSF}$) graphs. These objects are ordered with endomorphic structure subsumption which characterizes consistent instantiation of subterms, and more generally inheritance of features from a sort to its subsorts. Constraint systems consisting of declarative rules and axioms formalizing these operations as syntax-driven constraint normalization also provide effective operational interpretations. Extending these notions to assimilate distinct but similar objects can then be done by loosening semantic congruence among constructors. This is achieved by fuzzifying the constraint system. This kind of fuzzification can be defined and used as well for approximate data and knowledge representation and processing thanks to richer attributed object and concept structures such as $\mathcal{OSF}$ graphs.

Why Lattice Theory?

Because it is the mathematics of consistent approximation. Indeed, when (fuzzy) approximation of $\mathcal{FOT}$s or $\mathcal{OSF}$ graphs is defined as (fuzzy) structural subsumption, (fuzzy) unification is the Greatest Lower Bound (**glb**) operation, and (fuzzy) generalization is the Least Upper Bound (**lub**) operation. This provides a (fuzzy) **glb** operation over $\mathcal{FOT}$s and $\mathcal{OSF}$ object structures to act as, *e.g.*, a kind of "*(fuzzy) object join*" to specify approximate retrieval patterns over an object database. Dually, the **lub** operation is the computation of the most specific $\mathcal{FOT}$ or $\mathcal{OSF}$ graph up to a fuzzy approximation degree that is their most specific approximate generalization at that fuzzy approximation degree. Such could be used, *e.g.*, for fuzzy object schema inference or Machine Learning by fuzzy inductive reasoning. Therefore, understanding the formal operational aspects of (fuzzy) structure unification and its dual (fuzzy) structure generalization are invaluable pursuits most suitably formalized using Lattice Theory.

What is our objective?

It is to extend to fuzzy operations (to "*fuzzify*") both lattice operations on $\mathcal{FOT}$s and $\mathcal{OSF}$ graphs. Calibrating such structures with fuzzy truth levels as approximation degrees can then exploit more expressive lattice-theoretic operations (fuzzy unification, but also fuzzy generalization). Our pragmatic motivation is that such fuzzy lattice operations on $\mathcal{FOT}$s and $\mathcal{OSF}$ graphs are very convenient in structured data and knowledge representation and processing, such as *approximate* Information Mining and Retrieval.

Is this still a bit too fuzzy?

Be that as it may, we — *the authors* — hope that you — *the reader* — will be prompted to muse further into this book's contents to understand what the foregoing techno-gibberish actually means. Who knows? You may just share the frizzy fuzzy fun we felt defrizzing tangled fuzz, and perhaps even be enticed to use, or extend, our ideas and results.

Keywords: Approximate Information Processing; Lattice Algebra; First-Order Terms; Order-Sorted Feature Graphs; Fuzzy Unification; Fuzzy Generalization; Fuzzy Knowledge Representation; Automated Fuzzy Reasoning; Pattern-directed Fuzzy Reasoning; Fuzzy Pattern Induction; Fuzzy Machine Learning.

DRAFT—PLEASE DO NOT DISTRIBUTE

DRAFT—PLEASE DO NOT DISTRIBUTE

Version of April 10, 2020

Preamble

Our model of the world is, at best, fuzzy. What we hold to be true or false, as far as reasoning is concerned, is a partial approximation of ideal concepts and relations among such. Yet, far from suffering from this seeming imperfection, we are actually quite clever at many tasks involving reasoning (including recognition and learning) precisely *because* we can efficiently make sense out of approximate knowledge. It is, therefore, natural to wish that all AI tools be given the capability of approximate reasoning as a practical means to make efficient pragmatic sense out of the abundance of knowledge fed by the current supernova-size explosion of data — whether extracting knowledge, learning from it, or using what is learned to render intelligent services for useful aims; or all of that.[1]

The ideas reported in this book are some mathematical and computational reflections on approximate reasoning with data and knowledge represented as algebraic trees and, more generally, labeled rooted graphs (*i.e.*, most models). We view approximation as a partial order on object structures composed of symbols, where some symbols may denote more or less similar concepts. The ideas we have undertaken to develop here arose during the visit of the first author to Milan, Italy, in the Fall of 2016, at the invitation of the second author. They are the consequence of a congruence of minds intrigued at fuzzifying the power of lattice theory for data and knowledge processing. They are the result of our discussions trying to give simple answers to simple questions. The initial question was simple indeed: "*What happens to the Reynolds-Plotkin lattice of $\mathcal{FOT}$s with fuzzy unification and generalization?*" But since this lattice is itself only a special case of the lattice of rooted order-sorted feature graphs: "*What happens when we fuzzify $\mathcal{OSF}$ lattice operations?*" We looked for intuitive, formal, and operational, answers to these questions. This led, after some methodic and laborious research and a few initial but unsatisfying answers, to this current collection of technical thoughts. These constitute, in our opinion, just *a start.*

We wish to share this, should there be anyone interested in the same or related topics. We hope that the reader will draw satisfaction, even if only partially, in seeing a presentation of a comprehensive, yet simple and coherent, family of algebraic structures for fuzzy deduction and induction. It is our further wish that these ideas beget new ones in the reader's mind since, as we try to illustrate, we believe that there is a high potential for further work and applications.

We also took up the challenge of making this work destined to a wide audience, yet be self-contained. Most of the needed background and vocabulary for the technical notions we use and/or build upon are reviewed, summed-up in an appendix, where we cover all that is essential to understand the rest of this book, along with more examples and a quick review of the Generalized Distributive Law, a generic efficient implementation technique for commutative semirings which uses distributivity to optimize evaluation.

However, although this work may first appear as just one more theoretical niche for idle math-

[1] The expression "*fuzzy logic*" occurs 30 times in WIPO Technology Trends 2019 – Artificial Intelligence [130].

ematicians, it has in fact a universal pragmatic purpose. A longer, however more accurate, title should have perhaps been worded as, "***Lattice-Theoretic Operations for Fuzzy Inference by Deduction and Induction over Similar Data and Knowledge Structures***." Indeed, this investigation has led us to understand a universal model for efficiently *implementing* a powerful fuzzy reasoning algebra over approximate subsumption-ordered object structures.

Developing a mathematically formal answer to our questions appeared indeed as a necessity to us, but it was not our objective. Rather, the latter was to build formal bases for the correct and efficient implementation of such lattice operations capable of approximate pattern recognition of data and knowledge structures. Further still, even that was to constitute only the initial steps towards building adequate tools for our ultimate longer-term aim: using these tools as well-honed libraries for *applications* showing the benefits of our ideas in improving information retrieval and approximate reasoning in general. So we adjoined, following the initial formal part, a discussion in the form of a non-exhaustive grab-bag of several issues that have come to our attention in the light of our understanding, reflecting on the consequences of our formal contribution on a few connected domains, reviewing the state of art in these domains as related to our work. This has also allowed us to be more accurate in situating our contribution in the context of otherwise interesting, although different, works, and open the way to potentially fruitful connections.

In summary, we believe that this work has barely scratched open what appears like a bountiful vein of ore in AI research that may very well turn out to be a mother lode of pragmatically valuable and formally clean ideas. Indeed, we think that this is just a beginning.

Acknowledgements The most difficult task for all authors is to ensure that what they wrote is understood by their intended readership. So, first and foremost, we wish to thank the few colleagues and peers that have been kind enough to find the time in their busy schedules to read through partial drafts of this opus and to make useful comments that led to improve it in both form and contents. They are, in alphatical order, François Bry, Peter Buneman, Yves Caseau, Patrick Cousot, Leonidas Fegaras, Bart Kosko, Nabil Layaïda, Patrick Lincoln, Nicolas Nicolov, Fernando Pereira, Gordon Plotkin, and Francesca Rossi. Although our thanking them here does not necessarily entail that they find either form or contents of this book ideal or conform to their own scientific perspectives,[2] it has greatly benefitted from their kind attention. We have indicated wherever a specific technical point has been emended in the light of their feedback.

Also, each author is indebted to the other for being foolish enough to discuss the issues in sufficient detail as to be formally convincing. It became all the more intriguing when we realized that this may have further consequences over variant structures and operations.

And of course, each author is more indebted to their respective families as they each were making even lesser sense than usual while rambling about crazy fuzzy lattices when working in the garden tending sprouting ivy crawling up and down a criss-crossing mesh of guiding trellis.[3]

It is hoped that our ideas will be the seeds of several creative sprawling upshots.

Authors' comment: *N.B.: Reader please be advised that several parts of this book are still in a state of draft. It is hence* ***not to be distributed*** *at this stage, and is entrusted to you by the authors for comments and/or suggestions that may inspire us to improve it (any shall be duly acknowledged).*

[2] One would need to ask them.

[3] Like trying to parse, let alone make sense out of, this very sentence!

DRAFT—PLEASE DO NOT DISTRIBUTE

The parts that have been published have been indicated, where and when, and links given. The rest is constituted of parts being elaborated and therefore still in a draft state. These are essentially sets of notes and are meant to evolve into finished form as the writing is being completed. Thus, some of their current contents is likely to be modified, or could be synthesized more succinctly, or may diseappear altogether in a later update. Such parts are incomplete and/or possibly inconsistent in their current form. Also, new parts may appear later.

DRAFT—PLEASE DO NOT DISTRIBUTE

Table of Contents

DRAFT—PLEASE DO NOT DISTRIBUTE

List of Figures

List of Examples

DRAFT—PLEASE DO NOT DISTRIBUTE

Chapter 1

Version of April 10, 2020

Generalities

Our expected readership is assumed to be at ease with, or at least not averse to, advanced senior-level or graduate-level Symbolic Logic and Algebra;[1] more specifically, basic Lattice Theory.[2] Although not required, familiarity with some software specification and/or programming languages should be a plus. In particular, one should be comfortable with the syntax, data structures, and operations of Logic Programming (*e.g.*, Prolog).[3] Also, one should not mind our making use of formal notation as we find it helpful in conveying accurately what we mean. However, we strive to keep this notation simple and intuitive, and also "easy to program." Indeed, our *leitmotiv* is deriving implementable methods from declarative specifications.

What follows in this introduction explains our motivation, provides an overview of the ideas we discuss, and summarizes the book's organization.

1.1 Motivation

In this work, we do not mean to deal with fuzzy concepts or fuzzy properties thereof. This would require to encode existing knowledge and data bases to be populated with fuzzy entities. This would not only be a formidable task to undertake but also an unrealistic assumption. Rather, we wish to take the world of knowledge and data as it exists with possibly some additional information relating various concept or data constructor symbols. This information consists in a fuzzy measure of truth approximation derived from the meaning of symbols. For example, in a knowledge base dealing with people records and supporting approximate-pattern information retrieval, such a measure may indicate how "*semantically close*," the symbols "*person*" and "*individual*" (say) are to denote the same concepts (say with a .9 approximation degree).[4] Additionally, relating some attributes of so similar concepts could identify, at a given approximation degree, which attributes of a concept correspond to what attributes of a similar concept.

[1] https://www.encyclopediaofmath.org/index.php/Algebra_of_logic
[2] https://en.wikipedia.org/wiki/Lattice_(order)
[3] https://en.wikipedia.org/wiki/Prolog
[4] Degree 0.0 meaning distinct; degree 1.0 meaning identical.

Authors' comment: *We need some general comparative discussion about structurally vs. numerically assessed approximation — viz., First-Order Term ($\mathcal{FOT}$) or Order-Sorted Feature ($\mathcal{OSF}$) structure vs. Fuzzy Set ($\mathcal{FS}$) — lattice-theoretic calculi, and why it would be interesting to combine both.*

Such an example should contain realistically fuzzy concepts and expressions to help to motivate the need for fuzzy order-sorted featured $\mathcal{OSF}$ patterns. For example (quoting all fuzzy concepts or expressions), say we wish to retrieve:

> *Books on "**advanced**" "**algebra**" authored by a "**not so well-known**" mathematician from "**Eastern Europe**" "**around the end of the 18th century**" which have had "**a high impact**" on the topic of "**decision making**" "**today.**"*

This query pattern is vague. Indeed, several components in it (emphasized as quoted, boldfaced) are specified that have an approximate interpretation which could be rendered as fuzzy. On the other hand, it also corresponds to some typed attributed object structure (book, with author, subject, etc.). It therefore comes naturally to express such a vague query as some fuzzy $\mathcal{OSF}$ structure — say, with the following shape:

```
book
( author → mathematician
           ( fame → low
           , origin → east-european
           , period → end-of-18th-century
           )
, subject → algebra
            ( level → advanced
            )
, topic → decision-making
, impact → high
)
```

[To be completed later. . .]

Other points to discuss/elaborate:

- *Use fuzzy $\mathcal{OSF}$ generalization to aid, for example, in Machine Learning as a precious initial focusing step prior to exploiting number-analytical techniques such as Bayesian Nets or SVMs [150].*
- *While Propositional Logic ($\mathcal{PL}$) is a Boolean lattice, Fuzzy Propositional Logic ($\mathcal{FL}$) is a Brouwer lattice (also referred to as a Heyting algebra) [121].*[5]

  $\mathcal{OSF}$ Logic is also a lattice algebra: it has an infimum operation ($\mathcal{OSF}$ unification) and a supremum operation ($\mathcal{OSF}$ generalization). However, it is not a Boolean lattice. It is not even distributive [4], [6]. Seeing an $\mathcal{OSF}$ term as a logical constraint, $\mathcal{OSF}$ unification corresponds to Boolean conjunction, but $\mathcal{OSF}$ generalization is more approximate than Boolean disjunction. This is true as well for the lattice of first-order terms ordered

[5] See Appendix Section 2.2.1.

by subsumption,[6] as was first shown by Reynolds [141] and, simultaneously and independently, by Plotkin [135]. The lattice of ψ-terms (i.e., $\mathcal{OSF}$ terms in normal form) can be extended with a disjunction operation. So-extended ψ-terms, called ϵ-terms in [4] and [6], form a distributive lattice. It is not Boolean as it does not have complements. Further extending ϵ-terms with a restricted (constructive) form of complementation can provide a structure of Brouwer lattice.[7]

- *cite relevant references and give BibTEX format and with a public pdf link (see to enrich existing file `main.bib` — see BibTEX templates in `bibtex-templates.bib`). N.B.: Not all current citations in this file are to be used, nor appropriate. They will be eventually reviewed and cleaned.*
- *potential applications (approximate information retrieval, ...)*

1.2 Objective

We seek to explore the "*fuzzification*" of operations on $\mathcal{FOT}$s and $\mathcal{OSF}$ constraints as used in Logic Programming (*e.g.*, [61, 111], and [19]). We proceed by conjugating the lattice-theoretic properties of $\mathcal{FOT}$s and $\mathcal{OSF}$-constraint graph algebras ordered by structure subsumption with a fuzzy interpretation of equality. We start with the Reynolds-Plotkin original characterization of first-order algebraic term subsumption as a lattice ordering, further extended into an $\mathcal{OSF}$ constraint subsumption lattice, and further enhanced with fuzzy lattice operations.

We approach this study from the perspective of information approximation. In this context, we focus essentially on fuzzifying lattice-theoretic operations on $\mathcal{FOT}$s and $\mathcal{OSF}$ constraints. These formal (strict) structures and the fuzzified lattice operations thereon may then be used in, *e.g.*, approximate Data and Knowledge representation and processing. As it has been demonstrated in all areas they have been applied to, Fuzzy Logic and Algebra offer greater flexibility and expressivity for performing approximate deduction (inference) and induction (abstraction). Fuzzy inference and abstraction operations over $\mathcal{FOT}$s and $\mathcal{OSF}$ constraints are therefore bound to offer an appreciated improvement of "smarts" in approximate pattern-based retrieval, mining, and learning. In addition, these fuzzy operations are effective, efficient, and conservative extensions of their crisp versions.

We shall always insist on formulating formal lattice-theoretic operations following *declarative*, as opposed to *procedural*, specifications in the form of syntax-driven transformational rules which can be applied in any order. Besides greatly simplifying proving their correctness by structural induction, this eliminates irrelevant control issues and side-effectable environments which typically clutter procedural specifications.

We also review some literature we felt relevant to our pursuit (even if only as potential topics for further research) dealing with related formal notions from general correspondances between arguments (positional or keyword) attributed structures based on syntactic terms, graph data structures, finite-state automata, and their fuzzification. We also provide a few examples of how these

[6] That is, by $\mathcal{FOT}$ matching modulo variable renaming.
[7] See [6], Section 6.1, Page 336: "*Negative information.*"

structures may be put to use for approximate Knowledge Representation as they offer a more flexible means to perform fuzzy deduction and induction over abstract attributed objects and concepts represented as order-sorted feature constraints.

1.3 Organization of Contents

The rest of this book is organized as follows.

Chapter 2 reviews and defines the basic background notions and notations in Lattice Theory and Fuzzy Set Theory that we use in the rest of this book.

Chapter 3 focuses on first-order terms and fuzzifying their lattice operations. Section 3.2 covers the necessary background on $\mathcal{FOT}$s and Section 3.3 on $\mathcal{FOT}$ substitutions. Section 3.4 overviews background on the lattice of $\mathcal{FOT}$s, and offers an original declarative approach to $\mathcal{FOT}$ generalization that will later ease for us the task of fuzzifying this operation. Section 3.5 reviews the rules for $\mathcal{FOT}$ unification, while Section 3.6 develops an original approach for declarative $\mathcal{FOT}$ generalization. In Section 3.7, we proceed to fuzzify the Reynolds-Plotkin $\mathcal{FOT}$ subsumption lattice. We start with a specific formal fuzzification of $\mathcal{FOT}$ unification due to Maria Sessa. We then show how to make it more expressive by extending it to tolerate arity and/or argument-order mismatch in addition to just similar functors, and proceed to define their respective dual fuzzy generalization operations.[8] Finally, Section 3.9 recapitulates the contents of this chapter.

Chapter 4 focuses on order-sorted feature constraints and fuzzifying their lattice operations. Section 4.1 presents basic vocabulary and properties of order-sorted feature terms describing data and knowledge structures, and exposes the $\mathcal{OSF}$ term lattice operations. Section 4.2 continues with the fuzzification of the lattice operations on $\mathcal{OSF}$ terms.

Chapter 5 further extends the fuzzy-lattice constructions of the previous two chapters to be able to define fuzzy unification and generalization of term structure over constructors that are similar modulo *partial* schema-realignment. Section 5.1 motivates the issue with the case of partial argument-aligment maps, and proposes a solution. Section 5.2 carries this over to general feature alignment.

Chapter 6 puts this work in context. We review extant work that has some potential relation to the work presented here. Section 6.1 overviews related work. Section 6.2 discusses further work. Section 6.3 looks at implementations. Section 6.4 considers pragmatic upshots of using some ideas in this book in proof-of-concept software realizations. It also goes into some details regarding the implementation of a fuzzy partial order's lattice operations. Section 6.5 speculates about potential applications. Section 6.6 is a proposal for a convincing use case in the area of intelligent information retrieval mixing fuzzy and Bayesian reasoning.

Chapter 7 concludes with some comments on the usefulness and future evolution of this work: Section 7.1 recapitulates, Section 7.2 shares some thoughts on fuzziness, Section 7.3 indicates further work.

We have also adjoined an appendix to review quickly some background material defining basic notions useful to understand more easily the notions presented in this book, additional $\mathcal{OSF}$ constraint-solving rules and examples of $\mathcal{OSF}$ lattice operations.

[8] Parts of this section appeared in [20]; see presentation slides in [21].

Chapter 2

Version of April 10, 2020

Preliminaries

> Monism is the theory that anything less than everything is nothing.
>
> SAUL GORN—*Self-Annihilating Sentences [88]*

In this chapter, we provide a succinct summary of elementary formal notions, terminology, and notation constituting background for the issues developed in this book. Section 2.1 reviews basic terminology, defines formal set-theoretic properties of relations as sets of pairs. Section 2.2 extends the properties defined in Section 2.1 to their corresponding fuzzy notions.

2.1 Basic Algebra Terminology

This section is a brief review of essential terminology and facts in Set Theory and Set Algebra. Section 2.1.1 defines relations, and functions as specific relations. Section 2.1.2 reviews properties of relations. Section 2.1.3 reviews properties of specific structure-preserving mappings between algebras called homomorphisms. Section 2.1.4 reviews elementary notions and notations in specific algebras called lattices.

2.1.1 Relations and functions

We assume known the following elementary set-theoretic notions and their usual mathematical notation: set, empty set ($\emptyset$), subset ($\subseteq$); set operations: intersection ($A \cap B$), union ($A \cup B$), disjoint union ($A \uplus B$), difference ($A \setminus B$), Cartesian product ($A \times B$), complement ($\overline{A}$), powerset ($\mathcal{P}(A)$), cardinality ($|A|$); Boolean values and operations: $\mathfrak{true}$, $\mathfrak{false}$, $\mathfrak{and}$, $\mathfrak{or}$, $\mathfrak{not}$. We shall use the notation $a \stackrel{\text{def}}{=} b$ as an indication that the expression a is defined as expression b.

A binary *relation* r_{AB} on two sets A and B is a subset of their Cartesian product; *i.e.*, $r_{AB} \subseteq A \times B$. Given $n \in \mathbb{N}$, $n \geq 2$, an n-ary *relation*, $r_{A_1 \ldots A_n}$ on n sets A_i, $i = 1, \ldots, n$, is a subset of their Cartesian product; *i.e.*, $r_{A_1 \ldots A_n} \subseteq A_1 \times A_n$. For any set A, the relation $\mathbb{1}_A \stackrel{\text{def}}{=} \{ \langle x, x \rangle \,|\, x \in A \}$ is called the *identity* relation (or simply the identity) on A. The *inverse* relation (or simple the inverse) of a relation $r_{AB} \subseteq A \times B$ the relation $r_{AB}^{-1} \subseteq B \times A$ the relation defined as: $r_{AB}^{-1} \stackrel{\text{def}}{=} \{ \langle y, x \rangle \mid \langle x, y \rangle \in r_{AB} \}$. Given three sets A, B, and C, and two relations $r_{AB} \subseteq A \times B$

and $r_{BC} \subseteq B \times C$, the *composition* $r_{AB} \circ r_{BC}$ of two relations is defined as $r_{AB} \circ r_{BC} \stackrel{\text{def}}{=} \{ \langle x, z \rangle \mid \exists y \in B \text{ s.t. } \langle x, y \rangle \in r_{AB} \text{ and } \langle y, z \rangle \in r_{BC} \}$. Note that any relation $r_{AB} \subseteq A \times B$ satisfies $\mathbb{1}_A \circ r_{AB} = r_{AB}$ and $r_{AB} \circ \mathbb{1}_B = r_{AB}$.

A *function* f from a set A to a set B is a relation on A and B that associates to every element of A a *unique* element of B (its *image*). Formally, $\forall x \in A, \forall y \in B, \forall y' \in B, (\langle x, y \rangle \in f \text{ and } \langle x, y' \rangle \in f) \Rightarrow y = y'$. For this reason, we use the conventional notation $y = f(x)$ rather than $\langle x, y \rangle \in f$, and the notation $f : A \to B$ to denote that f is a function *from* A (the *domain* of f), *to* B (the *range* of f).[1] Note that the identity relation $\mathbb{1}_A$ on any set A is a also a function $\mathbb{1}_A : A \to A$ from A to itself.

A *function* f from a set A to a set B is:

- *injective* (or *one-to-one*) *iff* different elements in the domain are mapped to different elements in the range (formally, $x \neq y \Rightarrow f(x) \neq f(y)$);[2]
- *surjective* (or *onto*) *iff* all elements in the range are images of some element in the domain (formally, $\forall y \in B, \exists x \in A, \text{ s.t. } y = f(x)$);
- *bijective* (or *one-to-one onto*) *iff* it is both injective and surjective.

If a function $f : A \to B$ is bijective, then f's *inverse* is also a function $f^{-1} : B \to A$ since, by definition of its inverse as a relation, $\forall y \in B, \ f^{-1}(y) = x$ such that $x \in A$ and $y = f(x)$. The inverse of f satisfies $f^{-1}(f(x)) = x$ for all $x \in A$, and $f(f^{-1}(y)) = y$ for all $y \in B$. This is because $f^{-1} \circ f = \mathbb{1}_A$ and $f \circ f^{-1} = \mathbb{1}_B$. Note that the inverse of a bijection is also a bijection. Essentially, a bijection between two sets A and B indicates that these two sets can be identified as each distinct element in one set corresponds to a unique distinct element in the other set.

2.1.2 Properties of relations

A binary relation r on a set S is a subset of the Cartesian product $S \times S$; *i.e.*, it is a set of pairs of elements of S. We say that r is:

- *reflexive* iff:

$$\mathbb{1}_{S \times S} \subseteq r \tag{2.1}$$

  where $\mathbb{1}_S$ is the identity relation on S;

- *symmetric* iff:

$$r = r^{-1} \tag{2.2}$$

  where r^{-1} is the inverse relation of r;

- *antisymmetric* iff:

$$r \cap r^{-1} \subseteq \mathbb{1}_{S \times S} \tag{2.3}$$

  where $r \cap r'$ is the intersection of r and r'; *viz.*, the relation on S defined as: $r \cap r' \stackrel{\text{def}}{=} \{ \langle x, y \rangle \mid \langle x, y \rangle \in r \text{ and } \langle x, y \rangle \in r' \}$;

[1] One will note that a relation r_{AB} in $A \times B$ is also a function from A to $\mathcal{P}(B)$ [11].
[2] Or, equivalently, *iff* $f(x) = f(y) \Rightarrow x = y$.

- *transitive* iff:

$$r \circ r \subseteq r \tag{2.4}$$

where $r \circ r'$ is the composition of r and r'.

DEFINITION 2.1 (PREORDER) *A relation r on a set S is a preorder on S iff it is reflexive and transitive; i.e., iff r satisfies conditions* (2.1) *and* (2.4).

DEFINITION 2.2 (EQUIVALENCE) *A symmetric preorder r on a set S is called an equivalence on S; that is, r satisfies conditions* (2.1), (2.2), *and* (2.4).

An equivalence relation $\equiv$ on a set S defines a partition of this set; namely, a collection of non-empty subsets S_i, $1 \leq i \leq \mathbf{I}_{\equiv} \in \mathbb{N}$, of S (the equivalence classes) such that:

$$1 \leq i \neq j \leq \mathbf{I}_{\equiv} \Rightarrow S_i \cap S_j = \emptyset \tag{2.5}$$

and:

$$S = \bigcup_{i \leq \mathbf{I}_{\equiv}} S_i \tag{2.6}$$

where $\mathbf{I}_{\equiv}$, the *index* of $\equiv$, is the number of equivalence classes of $\equiv$ forming the partition of S. The equivalence class of an element of $x \in S$ is denoted $[x]^{\equiv}$ and is defined as:

$$[x]^{\equiv} \stackrel{\text{def}}{=} \{\, y \in S \mid x \equiv y \,\}. \tag{2.7}$$

DEFINITION 2.3 (PARTIAL ORDER) *A relation r on a set S is a partial order on S iff it is an antisymmetric preorder on S; i.e., iff r satisfies conditions* (2.1), (2.3), *and* (2.4).

A *partially-ordered set* (or *poset*) is a pair $S, \leq$ where S is a set, and $\leq$ is a partial order on S.

2.1.3 Homomorphisms

A *homomorphism* is a function between two sets possessing an algebraic structure that preserve this algebraic structure. See Appendix Section A.1 for a detailed ontology of some algebraic structures relevant to the material in this book.

Given any algebraic structure $\mathfrak{A}$, whether its concerns relations or operations, and given two sets A and B that are both $\mathfrak{A}$ structures, a $\mathfrak{A}$-homomorphism is a function $f : A \to B$ such that $f(x \star y) = f(x) f(\star) f(y)$ for any operation $\star$ on A corresponding to operation $f(\star)$ on B, or xRy implies $f(x) f(R) f(y)$ for any relation R on A corresponding to a relation $f(R)$ on B.

An common example of relation homomorphism is an *order-homomorphism* whenever an order relation $\leq_A$ on A is mapped by f to an order relation $\leq_B$ on B such that the following is satisfied: for all $x \in A$ and $y \in B$, $x \leq_A y$ implies $f(x) \leq_B f(y)$.

An common example of operation homomorphism is a *monoid-homomorphism* whenever a monoid operation $\star_A$ on A is mapped by f to a monoid operation $\star_B$ on B such that the following is satisfied: for all $x \in A$ and $y \in B$, $f(x \star_A y) = f(x) \star_B f(y)$.

A homomorphism $f : A \to B$ is:

- a *monomorphism iff* f is injective;
- an *epimorphism iff* f is surjective;
- an *isomorphism iff* f is bijective (*i.e.*, it is both a monomorphism and an epimorphism);
- an *endomorphism iff* $A = B$;
- an *automorphism iff* f is a bijective endomorphism.

2.1.4 Elementary lattice theory

Lattice Theory is the formal study of properties of specific partially-ordered sets that are closed under some operations. It is one of the major mathematical formalisms (along with Algebra and Logic) used in the formal semantics of computation. We review here basic background from Lattice Theory that we rely on in this work: Section 2.1.4 gives the essentials and Section 2.1.4 explains modularity and distributivity.

We restrict ourselves to notions that are relevant to our presentation with the aim to make it as self-contained as possible. For a comprehensive treatise on the subject, the definitive reference is Birkhoff's book [46].

Essentials

Let $L, \leq$ be a poset.

If all pairs of elements x and y in L admit a unique greatest lower bound (**glb**) in L, noted $x \wedge y$, then $L, \leq, \wedge$ is called a *(lower) semi-lattice*; by definition, $\wedge$ is necessarily commutative. Dually, if all pairs of elements x and y in L admit a unique least upper bound (**lub**) in L, noted $x \vee y$, then $L, \leq, \vee$ is called an *(upper) semi-lattice*; by definition, $\vee$ is necessarily commutative.

DEFINITION 2.4 (LATTICE) *A lattice $L, \leq, \wedge, \vee$ is a poset such that $L, \leq, \wedge$ is a lower semi-lattice and $L, \leq, \vee$ is an upper semi-lattice.*

A lower semi-lattice $L, \leq, \wedge$ is called *complete* if *all* (*i.e.*, even non-finite) subsets $X \subseteq L$ admit a **glb** in L. Dually, an upper semi-lattice $L, \leq, \vee$ is called a *complete*, if *all* subsets $X \subseteq L$ admit a **lub** in L.

A lattice $L, \leq, \wedge, \vee$ is *lower-complete* iff $L, \leq, \wedge$ is a complete lower semi-lattice. Dually, it is *upper-complete* iff $L, \leq, \vee$ is a complete upper semi-lattice. A lattice is *complete* iff it is both lower-complete and upper-complete.

When L has a greatest element, we call it "*top*" and write it "$\top$." Note that $\top \stackrel{\text{def}}{=} \bigvee L$ by definition. Dually, when L has a least element, we call it "*bottom*" and write it "$\bot$." Also note that $\bot \stackrel{\text{def}}{=} \bigwedge L$ by definition.

PROPOSITION 2.1 (MODULAR INEQUALITY) *In any lattice $L, \leq, \wedge, \vee$, the following holds:*

$$\text{if } x \leq y \text{ then } x \vee (z \wedge y) \;\leq\; (x \vee z) \wedge y \tag{2.8}$$

for all x, y, z in L.

PROPOSITION 2.2 (DISTRIBUTIVE INEQUALITIES) *In any lattice $L, \leq, \wedge, \vee$, the following two inequalities hold:*

$$x \wedge (y \vee z) \;\geq\; (x \wedge y) \vee (x \wedge y), \tag{2.9}$$

$$x \vee (y \wedge z) \;\leq\; (x \vee y) \wedge (x \vee y) \tag{2.10}$$

for all x, y in L.

DEFINITION 2.5 (COMPLEMENTED LATTICE) *A complemented lattice is a lattice $L, \leq, \wedge, \vee$ with top $\top$ and bottom $\bot$ in which for any element $x \in L$, there is a unique complement $\overline{x} \in L$ satisfying:*

$$x \vee \overline{x} \;=\; \top \tag{2.11}$$

and:

$$x \wedge \overline{x} \;=\; \bot \tag{2.12}$$

DEFINITION 2.6 (BOOLEAN LATTICE) *A Boolean lattice is a lattice that is also a boolean ring; i.e., it is a distributive complemented lattice.*

Modularity and distributivity

Intuitively, when thinking of an ordering as comparing information contents, submodularity — *i.e.*, Inequality (2.8) — and subdistributivity — *i.e.*, Inequality (2.9) or Inequality (2.10) — express the fact that there may be non-uniform distribution of information either horizontally (modularity) or vertically (distributivity). When equalities rather than inequalities are required to hold always, this restricts the class of lattices to fully modular and fully distributive lattices.

DEFINITION 2.7 (MODULAR LATTICE) *A modular lattice $L, \leq, \wedge, \vee$ is a lattice in which the modular inequality (2.8) becomes an equality everywhere; viz.,*

$$\textit{if } x \leq y \textit{ then } x \vee (z \wedge y) \;=\; (x \vee z) \wedge y \tag{2.13}$$

for all x, y, z in L.

A useful way to visualize what makes a lattice be modular is that, for all pair of elements x and y, the interval between x and $x \vee y$ ($= \mathbf{lub}(x, y)$) is order-isomorphic with the interval between $x \wedge y$ ($= \mathbf{glb}(x, y)$) and y. In other words, in a modular lattice, opposite edges of diamond-shaped order diagrams are isomorphic (*i.e.*, bijective and order-homomorphic). This is why modularity is often referred to as the "diamond isomorphism" property.[3] Formally, this means that for any pair x and y, the two functions $u \to \big(u \vee x\big)$ and $v \to \big(v \wedge y\big)$ are mutually inverse order isomorphisms.

The following result provides a simple test of modularity.

THEOREM 2.1 (MODULARITY CONDITION) *Any lattice that admits a sublattice isomorphic to the 5-element lattice on the left side of Figure 2.1 is not modular.*

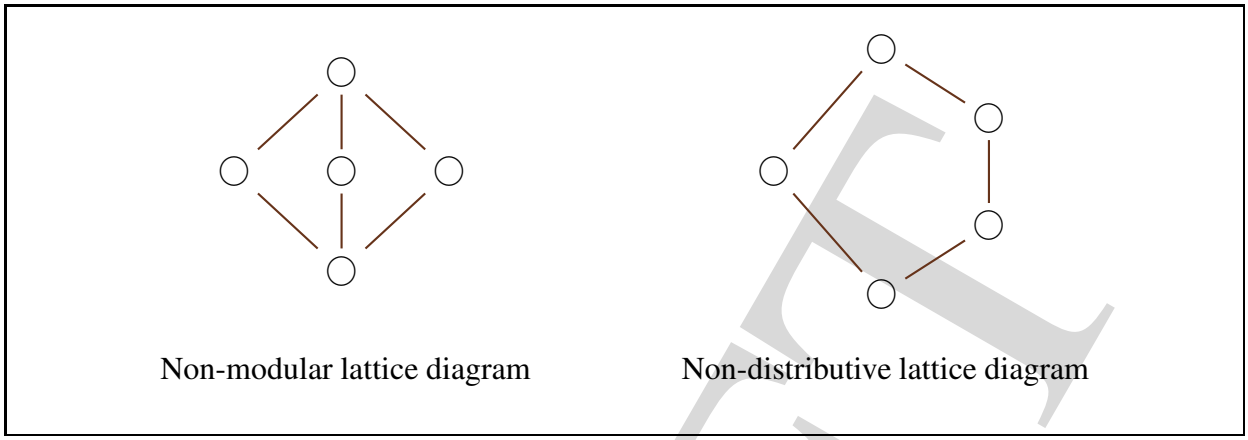


Figure 2.1: Non-modular and non-distributive lattice diagrams

DEFINITION 2.8 (DISTRIBUTIVE LATTICE) *A distributive lattice $L, \leq, \wedge, \vee$ is a lattice in which the inequality in condition (2.9) — or, equivalently the inequality in condition (2.10) — is strengthened into an equality everywhere.*

The following result provides a simple test of distributivity.

THEOREM 2.2 (DISTRIBUTIVITY CONDITION) *Any lattice that admits a sublattice isomorphic to the 5-element lattice on the right side of Figure 2.1 is not distributive.*

Therefore, a way to visualize whether a lattice is distributive is when all paths between two related elements have equal lengths; namely, information contents varies uniformly vertically. A useful property is that uniform vertical distribution of information entails necessarily uniform horizontal distibution of information. Indeed, it is not difficult to derive the following important fact.

THEOREM 2.3 *Every distributive lattice is also modular.*

2.2 Fuzzy Set Algebra

In this section, we review essential terminology and notation on Fuzzy Set algebra used in this book. Section 2.2.1 covers fuzzy relations; Section 2.2.2 covers fuzzy equivalence relations (called similarities); Section 2.2.3 covers fuzzy partial orders; Section 2.2.4 covers fuzzy lattices; and, Section 2.2.5 covers hiher-order fuzzy sets.

The fuzzy operator symbol on $[0.0, 1.0] \times [0.0, 1.0]$ we shall use for fuzzy conjunction, also called *Triangular norm* or T-norm,[4] is $\wedge$ (resp., $\vee$ for its fuzzy dual operation). This is generally interpreted as $\min$ (resp., $\max$); *e.g.*, as in Zadeh's seminal paper [176]. But other fuzzy operation interpretations can be considered depending on the desired effect.

[3] https://en.wikipedia.org/wiki/Modular_lattice#Diamond_isomorphism_theorem
[4] See https://www.encyclopediaofmath.org/index.php/Triangular_norm.

Thus, all the issues considered in this book are generic in the choice of fuzzy operators. Indeed, in fuzzifying terms, or anything for that matter, it is important to realize that many kinds of fuzziness may be obtained depending on the choice of these operators. In this section, we make some general points regarding the interpretations of fuzzy operators over the continuous interval $[0.0, 1.0]$ other than the classical $\mathfrak{min}$ and $\mathfrak{max}$.

Given to sets A and B, we use the notation B^A to denote the set of functions from A to B.

DEFINITION 2.9 (FUZZY SET) *A fuzzy set on a universe $\mathcal{U}$ is a function in* $[0.0, 1.0]^{\mathcal{U}}$.

When ambiguity may arise, a conventional set will be explicitly qualified as a "*crisp*" set.[5]

In Set Theory, a set S of elements of a universe $\mathcal{U}$ is formally identified with its Boolean-valued characteristic function $\mathbf{1}_S : \mathcal{U} \to \{\mathfrak{false}, \mathfrak{true}\}$ so that for all x in $\mathcal{U}$, $x \in S$ iff $\mathbf{1}_S(x) = \mathfrak{true}$. This defines a Boolean algebra isomorphism between the set of subsets of elements of $\mathcal{U}$ and $\{\mathfrak{false}, \mathfrak{true}\}$-valued functions on $\mathcal{U}$. When identifying a set with its Boolean characteristic function, set operations become logical operations: intersection becomes conjunction $[\mathbf{1}_{S\cap S'}(x) \stackrel{\text{def}}{=} \mathbf{1}_S(x)\ \mathfrak{and}\ \mathbf{1}_{S'}(x)]$, union becomes disjunction $[\mathbf{1}_{S\cup S'}(x) \stackrel{\text{def}}{=} \mathbf{1}_S(x)\ \mathfrak{or}\ \mathbf{1}_{S'}(x)]$, and complementation becomes negation $[\mathbf{1}_{\overline{S}}(x) \stackrel{\text{def}}{=} \mathfrak{not}\ \mathbf{1}_S(x)]$. Another equivalent Boolean algebra isomorphism is the one identifying the logical constants $\mathfrak{false}$ and $\mathfrak{true}$ with the numerical values 0 and 1, respectively, and the logical operations $\mathfrak{and}$, $\mathfrak{or}$, and $\mathfrak{not}$, with the numerical operations $\mathfrak{min}$, $\mathfrak{max}$, and $x \to (1.0 - x)$, respectively.[6] This is because these numerical operations on the numbers 0 and 1 stay isomorphically internal to $\{0, 1\} \subset [0.0, 1.0]$ with $0 < 1$. Formally, this amounts to plunging the discrete 2-valued poset $(\{0, 1\}, \leq)$ homomorphically into the continuous unit interval $([0.0, 1.0], \leq)$ with the identical numerical operations ($\mathfrak{min}$, $\mathfrak{max}$, and $x \to (1.0-x)$). Hence, when seen as $[0.0, 1.0]$-valued functions, these latter operations are a homomorphic extension of conventional Boolean algebra. This was the interpretation proposed originally by Zadeh in his seminal article on Fuzzy Sets [176].

However, there are many other ways in which conventional two-valued Boolean $\{0, 1\}$-Logic may be fuzzified by extending it to a multiple-valued logic depending on whether it allows multiple discrete or continuous similarity degrees in $[0.0, 1.0]$ (or any suitable "L-*structure*" [83, 74]).

In essence, *any* Boolean Algebra can be fuzzified by extending its basic Boolean connectives $\mathfrak{and}$, $\mathfrak{or}$, $\mathfrak{not}$, on the $\{0, 1\}$ similarity degrees of characteristic function (and thus the set operations intersection, union, and complementation), respectively into generic Boolean *Lattice* operations $\wedge$, $\vee$, $x \to \overline{x}$ on $[0.0, 1.0]$-valued functions whereby:

1. $\mathbf{1}_{S\cap S'}(x) \stackrel{\text{def}}{=} \mathbf{1}_S(x) \wedge \mathbf{1}_{S'}(x)$,
2. $\mathbf{1}_{S\cup S'}(x) \stackrel{\text{def}}{=} \mathbf{1}_S(x) \vee \mathbf{1}_{S'}(x)$,
3. $\mathbf{1}_{\overline{S}}(x) \stackrel{\text{def}}{=} \overline{\mathbf{1}_S(x)}$.

This may look like an innocuous property, but it turns out that, as thoroughly explained in Dubois and Prade's comprehensive treatise on Fuzzy Sets and Systems [74],[7] there may be many other

[5] This is the commonly used qualifier, in contrast to "*fuzzy*."

[6] We use the notation "$x \to e$" to denote the nameless function associating the expression e to the formal argument x; it is expressed as $\lambda x.e$ in functional programming.

[7] See also the more recent [180].

possible choices for the lattice operations on $[0.0, 1.0]$ for $\wedge$, $\vee$, and $x \to \overline{x}$ besides $\min$, $\max$, and $x \to (1.0 - x)$. It is for this reason, and with no loss of generality, that we use the former three generic operations rather than the latter specific more intuitive (and more familiar) ones in this book since our results hold in all general fuzzy algebras.

All we need are lattice operations $\wedge$ and $\vee$ (and $x \to \overline{x}$) on $[0.0, 1.0]$, which can then be made to apply to fuzzy sets defined as functions in $[0.0, 1.0]^{\mathcal{U}}$ by pointwise extension. Namely:

$$\left.\begin{array}{rcl} \wedge & : & [0.0,1.0]^{\mathcal{U}} \times [0.0,1.0]^{\mathcal{U}} \to [0.0,1.0]^{\mathcal{U}} \\ \vee & : & [0.0,1.0]^{\mathcal{U}} \times [0.0,1.0]^{\mathcal{U}} \to [0.0,1.0]^{\mathcal{U}} \\ x \to \overline{x} & \stackrel{\text{def}}{=} & 1.0 - x \ : \ [0.0,1.0]^{\mathcal{U}} \to [0.0,1.0]^{\mathcal{U}} \\ \top & \stackrel{\text{def}}{=} & x \to 1.0 \ : \ [0.0,1.0]^{\mathcal{U}} \\ \bot & \stackrel{\text{def}}{=} & x \to 0.0 \ : \ [0.0,1.0]^{\mathcal{U}} \end{array}\right\} \tag{2.14}$$

so that:

- $[0.0, 1.0]^{\mathcal{U}}, \wedge, \top$ is a commutative monoid,
- $[0.0, 1.0]^{\mathcal{U}}, \vee, \bot$ is a commutative monoid,
- $\wedge$, $\vee$ are mutually distributive,
- $\overline{\bot} = \top$,
- $\overline{\top} = \bot$,

and, for all fuzzy sets ϕ, ϕ_1, ϕ_2:

- $\phi \wedge \bot = \bot$,
- $\phi \vee \top = \top$,
- $\overline{\phi_1 \wedge \phi_2} = \overline{\phi_1} \vee \overline{\phi_2}$,
- $\overline{\phi_1 \vee \phi_2} = \overline{\phi_1} \wedge \overline{\phi_2}$,
- $\overline{\overline{\phi}} = \phi$.

From this, a plethora of familiar algebraic and order-theoretic properties ensue [46]. In the literature, the "$\wedge$" operator is sometimes called "*T-norm*" (for "*triangular* norm"), while the "$\vee$" operator is sometimes called "*T-conorm*" or "*S-norm*." Thanks to their algebraic properties, they can be derived from one another by duality using:

$$\begin{array}{rcl} \phi_1 \wedge \phi_2 & = & \overline{\big(\overline{\phi_1} \vee \overline{\phi_2}\big)} \\ \phi_1 \vee \phi_2 & = & \overline{\big(\overline{\phi_1} \wedge \overline{\phi_2}\big)}. \end{array} \tag{2.15}$$

In particular, since $\overline{\phi} \stackrel{\text{def}}{=} 1.0 - \phi$:

$$\begin{array}{rcl} \phi_1 \wedge \phi_2 & = & 1.0 - \big((1.0 - \phi_1) \vee (1.0 - \phi_2)\big) \\ \phi_1 \vee \phi_2 & = & 1.0 - \big((1.0 - \phi_1) \wedge (1.0 - \phi_2)\big). \end{array} \tag{2.16}$$

For example, here are three popular such $\wedge$ and $\vee$ operations on $[0.0, 1.0]$ used in practice [74], [102]:

- "Gödel" fuzzy operators:

$$\begin{cases} \alpha_1 \wedge_G \alpha_2 \stackrel{\text{def}}{=} \min(\alpha_1, \alpha_2) \\ \alpha_1 \vee_G \alpha_2 \stackrel{\text{def}}{=} \max(\alpha_1, \alpha_2) \end{cases} \tag{2.17}$$

- "Product" (or "probabilistic") fuzzy operators:

$$\begin{cases} \alpha_1 \wedge_P \alpha_2 \stackrel{\text{def}}{=} \alpha_1 \alpha_2 \\ \alpha_1 \vee_P \alpha_2 \stackrel{\text{def}}{=} \alpha_1 + \alpha_2 - \alpha_1 \alpha_2 \end{cases} \tag{2.18}$$

- "Łukasiewicz" fuzzy operators:

$$\begin{cases} \alpha_1 \wedge_L \alpha_2 \stackrel{\text{def}}{=} \max(0.0, \alpha_1 + \alpha_2 - 1.0) \\ \alpha_1 \vee_L \alpha_2 \stackrel{\text{def}}{=} \min(\alpha_1 + \alpha_2, 1.0) \end{cases} \tag{2.19}$$

Choosing any of these, or others, will determine how fuzzy inference is affected by each argument. For example, contrary to the "Gödel" fuzzy conjunction $\wedge_G$ that imposes the value of one over the other of two truth values, the "Product" version $\wedge_P$ is less "drastic" and will take a more balanced consideration of the values of both arguments. There are a few other fuzzy operators that have been given specific denominations that correspond to particular situations.[8] But one may design their own adequate $\wedge$ operator (or $\vee$ operator since one can be derived from the other by duality).[9]

Be that as it may, in all the actual numerical examples provided in this book for illustration, we use $\min$ (resp., $\max$), for simplicity reasons.

2.2.1 Fuzzy relation

Let us now fuzzify the conventional set theoretic definitions reviewed in Section 2.1.2. It is a straightforward homomorphic extension of the conventional view of (crisp) sets as $\{0, 1\}$-valued functions to $[0.0, 1.0]$-valued functions. Indeed, the former are just a particular case of the more general (fuzzy) sets seen as $[0.0, 1.0]$-valued characteristic functions.[10] That is, all the fuzzy notions are obtained as straightforward extensions of their crisp counterparts through a Boolean lattice homomorphism. The advantage of the fuzzy extension over conventional sets is that, being structurally richer, it is more expressive. It is a homomorphic extension insofar as all the formal algebraic properties of fuzzy sets and fuzzy-set connectives reduce to their conventional crisp versions when unfuzzifying the truth value $\phi(x)$ of every fuzzy element $\phi(x)/x$ of a fuzzy set ϕ into a crisp value in $\{0, 1\}$ for truth values which, when compared to a given value α in $[0.0, 1.0]$, are either strictly less (assimilated to 0), or greater or equal (assimilated to 1). Informally, this is the

[8] See, *e.g.*: http://www.nicodubois.com/bois5.2.htm.

[9] Designing specific fuzzy norms can be done visually in 3D using publicly available tools such as, *e.g.*, http://www.math.uri.edu/~bkaskosz/flashmo/graph3d2/.

[10] Such a fuzzy characteristic function is called a "*membership function*" in the literature following Zadeh's original terminology [176].

crisp set of elements with "*at least*" α as fuzzy truth value. This is called a fuzzy set's "α-*cut*" ϕ_α such that $\phi_\alpha(x) \stackrel{\text{def}}{=} 0$ whenever $\phi(x) < \alpha$ and $\phi_\alpha(x) \stackrel{\text{def}}{=} 1$ whenever $\phi(x) \geq \alpha$, for any given *approximation degree* α in $[0.0, 1.0]$.

We shall identify a fuzzy set ϕ on a set S with a function $(\phi : S \to [0.0, 1.0]) \in [0.0, 1.0]^S$.

DEFINITION 2.10 (FUZZY RELATION) *A fuzzy relation on a set S is a fuzzy set on $S \times S$.*

The following properties generalize those of crisp binary relations seen in Section 2.1.2. Like in the crisp case, we will look closer at essentially two kinds of fuzzy binary relations: fuzzy orders and fuzzy equivalences.[11]

Let $\rho : S \times S \to [0.0, 1.0]$ be a fuzzy relation on S. We say that ρ is:

- *reflexive* iff:

$$\mathbb{1}_{S\times S} \leq \rho \tag{2.20}$$

  where $\mathbb{1}_{S\times S}$ is the *fuzzy identity* relation on S defined as: $\mathbb{1}_{S\times S}(x, y) = 1$ if $x = y$ and 0 if $x \neq y$, for all x and y in S; and $\leq$ is *fuzzy set inclusion* defined as: $\rho \leq \rho'$ iff $\rho(x, y) \leq \rho'(x, y)$, for all x and y in S;

- *symmetric* iff:

$$\rho = \rho^{-1} \tag{2.21}$$

  where ρ^{-1} is the *fuzzy inverse* of ρ; *viz.*, the fuzzy relation on S defined as: $\rho^{-1}(x, y) \stackrel{\text{def}}{=} \rho(y, x)$, for all x and y in S;

- *antisymmetric* iff:

$$\rho \wedge \rho^{-1} \leq \mathbb{1}_{S\times S} \tag{2.22}$$

  where the *fuzzy meet* $\rho \wedge \rho'$ is the fuzzy relation on S defined as: $\big(\rho \wedge \rho'\big)(x, y) \stackrel{\text{def}}{=} \rho(x, y) \wedge \rho'(x, y)$, for all x and y in S;

- *transitive* iff:

$$\rho \circ \rho \leq \rho \tag{2.23}$$

  where the *fuzzy composition* $\rho \circ \rho'$ is the fuzzy relation on S defined as: $\big(\rho \circ \rho'\big)(x, y) \stackrel{\text{def}}{=} \bigvee_{z\in S}\big(\rho(x, z) \wedge \rho'(z, y)\big)$, for all x and y in S.

DEFINITION 2.11 (FUZZY PREORDER) *A fuzzy relation ρ on a set S is a fuzzy preorder on S iff it is reflexive and transitive; i.e., iff r satisfies conditions (2.20) and (2.23).*

[11] See [174] for even finer and more expressive kinds of useful fuzzy relations that can be defined algebraically.

2.2.2 Similarity

DEFINITION 2.12 (FUZZY EQUIVALENCE) *A fuzzy equivalence ρ on a set S is a fuzzy relation on S which is a symmetric fuzzy preorder on S — that is, ρ satisfies conditions* (2.20)*,* (2.21)*, and* (2.23)*.*

A fuzzy equivalence relation is also called "*similarity*" relation in the literature [102]. For this reason, we speak of "*similarity degree*" to denote the truth value of a pair so related.

A similarity relation $\sim$ on a set S is a fuzzy equivalence relation on S; *i.e.*, a fuzzy set of pairs of $S \times S$. When S is a finite discrete set, say indexed over $\{1, \ldots, n\}$, since a similarity relation $\sim$ on S is a fuzzy subset of $S \times S$, the three conditions of an equivalence can be visualized on a square $n \times n$ matrix $\sim \in \{1, \ldots, n\}^2 \to [0.0, 1.0]$ as follows. For all $i, j, k = 1, \ldots, n$:

- *reflexivity*: $i \sim i = 1.0$ (*i.e.*, entries on the diagonal are equal to 1.0);
- *symmetry*: $i \sim j = j \sim i$ (*i.e.*, all symmetric entries on either side of the diagonal are equal);
- *transitivity*: $i \sim k \wedge k \sim j \leq i \sim j$, for any $k \in \{1, \ldots, n\}$ (*i.e.*, going via an intermediate element will always result in a smaller or equal similarity degree than going directly).[12]

Given a similarity relation $\sim$ on a set S, the subset of $[0.0, 1.0]$ denoted $\textbf{DEGREES}^{\sim}$ and defined as $\textbf{DEGREES}^{\sim} \stackrel{\text{def}}{=} \{\, \alpha \in [0.0, 1.0] \mid x \sim_\alpha y, \text{ for some } x, y \in S \,\}$ is called the "*similarity degree set*" of $\sim$. A similarity degree $\alpha \in \textbf{DEGREES}^{\sim}$ can thus be used as an approximation-degree condition, and a similarity can be rendered a crisp equivalence on S by keeping only pairs in $\sim$ with similarity degree greater than or equal to α (*i.e.*, the α-cut of the similarity).

The similarity class $[x]^{\sim}_{\alpha}$ of an element $x \in S$ at an approximation degree α in $[0.0, 1.0]$ given a similarity $\sim$ on S is defined as:

$$[x]^{\sim}_{\alpha} \stackrel{\text{def}}{=} \{\, y \in S \mid x \sim_\beta y, \text{ for some } \beta \in [\alpha, 1.0] \,\}. \tag{2.24}$$

Thus, as the similarity degree α decreases from 1.0 down to 0.0, more similarities appear in $\sim_\alpha$ between pairs of distinct elements of S that were not related in α-cuts of $\sim$ at greater approximation degrees. In other words, as α decreases, the equivalence classes of $\sim_\alpha$ grow larger by coalescing classes of lesser similarity degrees; that is, for any $x \in S$, if $\alpha \leq \beta$, then $[x]^{\sim}_{\beta} \subseteq [x]^{\sim}_{\alpha}$. In particular, it is always the case that $[x]^{\sim}_{0.0} = S$; indeed, then, all elements are indistinguishable. Note finally that, for any pair $\langle x, y \rangle$ in $S \times S$ and similarity $\sim$: $S \times S \to [0.0, 1.0]$, if $x \sim_\alpha y$ for some α in $[0.0, 1.0]$, then $x \sim_\beta y$ for all $\beta \in [0, \alpha]$. For this reason, unless otherwise specified, *whenever we use an approximation degree as a subscript, we mean the* ***greatest*** *such degree.*

2.2.3 Fuzzy partial order

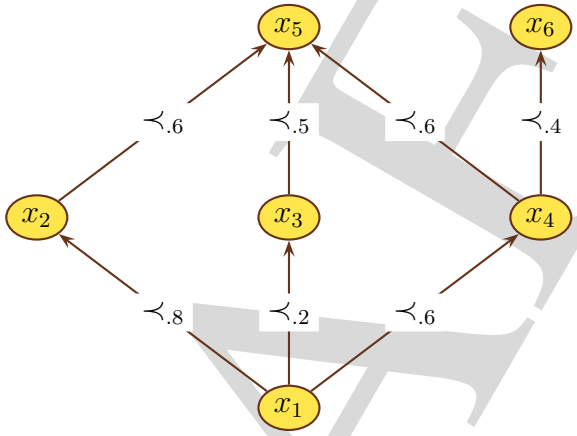
DEFINITION 2.13 (FUZZY PARTIAL ORDER) *A fuzzy relation ρ on a set S is a fuzzy partial order on S iff it is an antisymmetric fuzzy preorder; i.e., iff ρ satisfies conditions* (2.20)*,* (2.22)*, and* (2.23)*.*

[12] Here and elsewhere in this book, we shall use $\wedge/\vee$ for fuzzy conjunction/disjunction in generic formulas. We prefer using these more general symbols in our formalization since which specific fuzzy operations are used is irrelevant, as justified in Section 2.2. However, we shall use $\min/\max$ in all the illustrative examples we give that use actual numbers.

For a fuzzy partial order, as in the case of a fuzzy equivalence relation, when S is a finite discrete set $\{x_1, \ldots, x_n\}$, the three conditions of the above definition can be visualized on a square $n \times n$ matrix $\preceq \ in \ [0.0, 1.0]^2$ as follows:

- reflexivity and transitivity (just as for a similarity matrix);
- antisymmetry: the matrix must be triangular (up to reordering of columns and lines); this is because $\preceq_{ij} > 0$ implies $\preceq_{ji} = 0$, for all $i, j = 1, \ldots, n$ (*i.e.*, all symmetric entries on either side of the diagonal may not be both non-zero).

For example, the fuzzy binary relation $\preceq$ on the 6-element set $\{x_1, \ldots, x_6\}$ defined as the fuzzy min/max reflexive-transitive closure of the following weighted acyclic graph:[13]



corresponds to the following fuzzy matrix:

$$\preceq \ \stackrel{\text{def}}{=} \ \begin{bmatrix} 1.0 & 0.8 & 0.2 & 0.6 & 0.6 & 0.4 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.6 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.6 & 0.4 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{2.25}$$

upon which these conditions can be verified — which means that the fuzzy relation $\preceq$ so defined is a fuzzy partial order on the set $\{x_1, x_2, x_3, x_4, x_5, x_6\}$.

Note that, just as in the crisp case, any fuzzy preorder $\preceq$ on a set S (*i.e.*, a fuzzy relation on S that is reflexive and transitive) always implicitly defines the following fuzzy relations:

- a similarity $\sim$ on S defined, for any $\alpha \in [0.0, 1.0]$, as:

  $$\sim_\alpha \ \stackrel{\text{def}}{=} \ \preceq_\alpha \ \wedge \ \succeq_\alpha \tag{2.26}$$

  where $\succeq_\alpha$ is the fuzzy relation defined as: $\succeq_\alpha \ \stackrel{\text{def}}{=} \ \preceq_\alpha^{-1}$;

- a fuzzy partial order $\preceq$, a fuzzy set of partial orders $\preceq_\alpha$ on each partition $\Pi_\beta^\sim$ of S generated by $\sim$ in the fuzzy partition $\Pi^\sim \ \stackrel{\text{def}}{=} \ \{ \Pi_\beta^\sim \mid \beta \in \textbf{DEGREES}^\sim \}$, such that:

  $$[x]_\beta^\sim \ \preceq_\alpha \ [y]_\beta^\sim \ \text{iff} \ x' \ \preceq_\alpha \ y' \tag{2.27}$$

  for some $x' \in [x]_\beta^\sim$ and some $y' \in [y]_\beta^\sim$.

[13] This example is from [177].

2.2.4 Fuzzy lattice

DEFINITION 2.14 (FUZZY LATTICE) *A fuzzy lattice is a family of lattices* $\langle L, \leq_\alpha, \vee_\alpha, \wedge_\alpha \rangle$ *on partitions* Π_α *of similarity classes at each approximation degree* $\alpha \in$ **DEGREES**$^\sim$ *where* $\sim$ *is defined as* (2.26).

> ***Authors' comment:*** *Summarize relevant material from [29], [121], [60], using our notation to be consistent with the rest.*
>
> *[To be completed later. . .]*

2.2.5 Higher-order fuzzy sets

A "*Type-2 Fuzzy Set*" is a fuzzy set where the approximation degree is itself fuzzy. This was introduced by Zadeh [179] in 1975 in response to criticisms regarding the fact that the approximation degrees used in Fuzzy Logic do not represent uncertainty as probabilities do. A probability entails possible fluctuations of likelihood. This is not so for fuzzy truth values. There are no moments (mean, variance, higher-order moments), nor even (first, second, higher) derivatives. So in order to introduce a less commited notion of fuzziness, Zadeh introduced fuzzy values of higher types.

The idea is to fuzzify the membership value: instead of a value in $[0.0, 1.0]$ (called a "*Type-1*" membership value), a Type-2 membership value is a non-empty closed-interval $[\alpha, \beta] \subseteq [0.0, 1.0]$ (*i.e.*, $0.0 \leq \alpha < \beta \leq 1.0$). Writing $\wedge_1$ and $\leq_1$ the Type-1 fuzzy conjunction and ordering on $[0.0, 1.0]$, a $\wedge_2$ operation on Type-2 values is defined as:[14]

$$[\alpha, \beta] \wedge_2 [\alpha', \beta'] \stackrel{\text{def}}{=} [\alpha \wedge_1 \alpha', \beta \wedge_1 \beta'], \tag{2.28}$$

from which the corresponding ordering of the fuzzy Type-2 values is derived as:

$$[\alpha, \beta] \leq_2 [\alpha, \beta'] \text{ iff } \alpha \leq_1 \alpha' \text{ and } \beta \leq_1 \beta'. \tag{2.29}$$

The process can be repeated in this way for fuzzy sets of higher types. If we define a *Type-0* fuzzy set as a crisp set (*i.e.*, where membership values are in $\{0, 1\}$), and a *Type-n* fuzzy set for any $n > 0$ as a fuzzy set whose membership values are Type-$(n-1)$ values (*i.e.*, intervals of fuzzy Type-$(n-1)$ values), for any $n > 0$, then Equation (2.28) and Equation (2.29) can be generalized to define fuzzy conjunction and ordering of Type-n for any $n > 0$. That is,

$$[\alpha, \beta] \wedge_n [\alpha', \beta'] \stackrel{\text{def}}{=} [\alpha \wedge_{n-1} \alpha', \beta \wedge_{n-1} \beta'], \tag{2.30}$$

and:

$$[\alpha, \beta] \leq_n [\alpha, \beta'] \text{ iff } \alpha \leq_{n-1} \alpha' \text{ and } \beta \leq_{n-1} \beta.' \tag{2.31}$$

An essential result in [179] is that (2.30) and (2.31) confer a fuzzy lattice structure to any Type-n fuzzy set, for any $n \geq 0$.

[14] *Op. cit.*, pp. 241ff.

One way to view this generalization of "traditional" Type-1 fuzzy sets into Type-2 fuzzy sets is as adding a fuzzy dimension: instead of 2D, where a membership is caracterized as pair $\langle x, \alpha \rangle$ giving the $[0.0, 1.0]$ fuzzy membership of x as α, a Type-2 fuzzy set is a 3D triple $\langle x, \alpha, \beta \rangle$ to express that $\langle x, \alpha \rangle$ has β as its Type-1 fuzzy truth value: $\langle \langle x, \alpha \rangle, \beta \rangle$; *i.e.*, that the fuzzy membership value α for x has itself the fuzzy truth value β.

Similarly, higher-type fuzzy sets correspond to higher-dimension fuzzy sets (*i.e.*, fuzzy set of fuzzy sets). Mathematically, for any $n > 0$, Type-n fuzzy membership functions take values in n-dimensional subsets of $[0.0, 1.0]^n$, and Type-n fuzzy elements are $(n{+}1)$-tuples in $S{\times}[0.0, 1.0]^n$.

Technically, Zadeh's definition requires all fuzzy sets to be *convex* fuzzy sets [176].[15] Informally, a convex fuzzy set on a compact continuous set of elements is one such that element membership values never zig-zag up, then down, then up again, for increasing elements. A non-convex fuzzy set can always be "convexified" into its convex hull; *i.e.*, into the least convex membership function containing it (for the fuzzy subset ordering).[16] This is the assumption made by Zadeh in [179].

[15] *Op. cit.*, pp. 346*ff.*

[16] Geometrically, this is done by "shrink-wrapping" a non-convex set using any of the many existing convex-hull algorithms known in Computational Geometry.

DRAFT—PLEASE DO NOT DISTRIBUTE

Chapter 3

Version of April 10, 2020

First-Order Terms

3.1 Introduction

We are motivated by the versatile use of the First-Order Term ($\mathcal{FOT}$) as a *data structure* as done in Logic Programming thanks to the unification operation ([95] and [120]) and Inductive Logic Programming thanks to the generalization (or "anti-unification") operation ([156] and [94]). We extend the formal characterization of the set of $\mathcal{FOT}$s modulo variable renaming as a lattice due to Reynolds ([141]) and Plotkin ([135]) to such an algebraic structure where similarities among distinct constructors may exist that tolerate fuzzy $\mathcal{FOT}$ approximation. We study how these notions may be formalized while abiding by a fully declarative approach based on constraint processing in the same way as crisp unification is presented in [95] and [120], as opposed to a procedural control-conscious algorithm such as Robinson's [142].

Origins of unification and generalization

Unification The earliest printed account of the $\mathcal{FOT}$ unification operation, although not under this name, appears in 1930 in Jacques Herbrand's PhD thesis [95].[1] Later, in 1960, Dag Prawitz uses this as part of his Natural Deduction proof procedure for First-Order Logic [137]. Chap. 5 of Herbrand's thesis is first translated into English in 1967 by Jean van Heijenoort [166], and his full thesis is translated into English in 1971 by Warren Goldfarb [96].[2] In his thesis, although he does not call it "unification," Herbrand describes a declarative specification for $\mathcal{FOT}$ equation normalization more than 30 years before J. Alan Robinson actually gives the familiar name to an equivalent procedural method and dubbing it "unification" in order to extend his resolution principle from Proposition Logic to First-Order Logic [142]. This fact is already explicitly pointed out in 1976 by Gérard Huet in his French *thèse d'état* [97]. These rules are later used explicitly by Martelli and Montanari in 1982 (20 years after Robinson's paper, and 55 years after Herbrand's 1930 PhD thesis) in their method seeking to optimize Robinsons's algorithm [120]. They do not cite Herbrand's thesis, although it is explicitly cited in Huet's 1976 thesis which they cite. Probably because its name first appears in his paper on proof by resolution in First-Order Logic

[1] *Op. cit.*, Chap. 5, Sec. 2.4, pp. 95–97, where it corresponds to expanding an equation into normal form that verifies what he calls "*Property A*."

[2] Chap. 5 is on pp. 148*ff*.

in [142] and this name has been used in Logic Programming, most current venues attribute the paternity of $\mathcal{FOT}$ unification to Robinson.[3] While the name was indeed his coinage, the operation however was not new.

Generalization In 1970, John Reynolds and Gordon Plotkin publish each an article, in the same volume, each giving a different but equivalent procedure for the generalization of two $\mathcal{FOT}$s. The former calls it "*anti-unification*" ([141], page 138), and the latter calls it "*least generalization*" ([135], page 155). Each describes a method for computing the most specific $\mathcal{FOT}$ subsuming two given $\mathcal{FOT}$s in finitely many steps. The method consists in scanning them simultaneously from left to right as long as they agree, and where they disagree generating a pair of minimal generalizing substitutions by introducing a fresh variable, each time replacing the disagreeing terms with this new variable wherever they occur again simultaneously in each term.

Interestingly, in their 1982 ACM ToPLaS article on unification [120], Martelli and Montanari use a method that computes generalization of two terms implicitly (so-called "*common parts*") in preprocessing equations into congruence classes of terms (called "*multi-equations*"). This is in order to make unification more efficient by solving not just one equation but many at a time. However, they do not point out that this common-part form derived from two terms, by keeping only what is common in both, is in fact the dual of their unified form, which brings into one term what is in either.

3.2 Formalizing First-Order Terms

The $\mathcal{FOT}$ was introduced as a data structure in software programming by the Prolog language.[4] Thus, the $\mathcal{FOT}$ is Prolog's universal data structure in exactly the same way as the S-expression is that of LISP.[5] In this chapter, we formalize the $\mathcal{FOT}$ data structure as it is used in Logic Programming, then we expose in this formal setting its lattice-theoretic characterization before studying fuzzy extensions thereof.[6]

Using formal algebra notation, we write $\mathcal{T}_{\Sigma,\mathcal{V}}$ for the set of $\mathcal{FOT}$s on an operator signature $\Sigma \stackrel{\text{def}}{=} \biguplus_{n\geq 0} \Sigma_n$ where Σ_n is a set of n-ary operator symbols. The set $\mathcal{V}$ is a countably infinite set of variables. Also following Prolog's tradition, we shall designate an element f in Σ as a *functor*, with $\mathbf{arity}(f)$ denoting its number of arguments.[7] This set $\mathcal{T}_{\Sigma,\mathcal{V}}$ can then be defined inductively as:

$$\mathcal{T}_{\Sigma,\mathcal{V}} \stackrel{\text{def}}{=} \mathcal{V} \cup \{\, f(t_1,\ldots,t_n) \mid f \in \Sigma_n,\ n \geq 0,\ \text{ and } \ t_i \in \mathcal{T}_{\Sigma,\mathcal{V}},\ 1 \leq i \leq n \,\}.$$

We write c instead of $c()$ for a constant $c \in \Sigma_0$. Also, when the set Σ of functor symbols and the set $\mathcal{V}$ of variables are implicit from the context, we simply write $\mathcal{T}$ instead of $\mathcal{T}_{\Sigma,\mathcal{V}}$.

[3] For example, https://en.wikipedia.org/wiki/Unification_(computer_science).
[4] https://en.wikipedia.org/wiki/Prolog
[5] https://en.wikipedia.org/wiki/Lisp_(programming_language)
[6] Parts of this chapter have appeared in [20] and [22]. For presentation slides, see [21].
[7] When $\mathbf{arity}(f) = n$, this is sometimes denoted by writing f/n.

The set $\mathbf{var}(t)$ of variables occurring in a $\mathcal{FOT}$ $t \in \mathcal{T}$ is defined as:[8]

$$\mathbf{var}(t) \stackrel{\text{def}}{=} \begin{cases} \{ X \} & \text{if } t = X \in \mathcal{V} \\ \bigcup_{i=1}^{n} \mathbf{var}(t_i) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

A term t such that $\mathbf{var}(t) = \emptyset$ is called a *ground term*. We call $\mathcal{T}_\emptyset$ the subset of $\mathcal{T}$ of ground terms. The *depth* of a $\mathcal{FOT}$ t is a value in $\mathbb{N}$ defined inductively as:

$$\mathbf{depth}(t) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t \in \mathcal{V} \cup \Sigma_0; \\ 1 + \max_{i=1}^{n} \mathbf{depth}(t_i) & \text{if } t = f(t_1, \ldots, t_n) \text{ with } n > 0. \end{cases}$$

The **var** and **depth** notation is extended to a set of terms $T \subset \mathcal{T}$ as $\mathbf{var}(T) \stackrel{\text{def}}{=} \bigcup_{t \in T} \mathbf{var}(t)$ and $\mathbf{depth}(T) \stackrel{\text{def}}{=} \max_{t \in T} \mathbf{depth}(t)$.

3.3 Substitution

In this section, we give a brief essential account of $\mathcal{FOT}$ substitutions. For additional terminology and proofs of formal properties, please refer to A.2. Further formal order-theoretic properties of $\mathcal{FOT}$ substitution and unification are also studied in [75].

In order to express the notion of instance of a term, the concept of variable substitution σ is formalized as a functional mapping $\sigma : \mathcal{V} \to \mathcal{T}_{\Sigma,\mathcal{V}}$ that is the identity function everywhere on $\mathcal{V}$ except on a finite set of n variables, $n \in \mathbb{N}$, written $\mathbf{dom}(\sigma) \stackrel{\text{def}}{=} \{ X_k \mid X_k \neq \sigma(X_k) \}_{k=1}^{k=n}$, and called the domain of σ. The range of a substitution σ is the set of terms in $\mathcal{T}$ defined as $\mathbf{ran}(\sigma) \stackrel{\text{def}}{=} \{ t \in \mathcal{T} \mid \exists X \in \mathbf{dom}(\sigma) \text{ s.t. } \sigma(X) = t \}$.

Such a mapping σ from $\mathcal{V}$ to $\mathcal{T}$ is then extended homomorphically to a mapping $\bar{\sigma}$ from $\mathcal{T}$ to $\mathcal{T}$ as follows:

$$\bar{\sigma}(t) \stackrel{\text{def}}{=} \begin{cases} \sigma(X) & \text{if } t = X \in \mathcal{V} \\ f(\bar{\sigma}(t_1), \ldots, \bar{\sigma}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases} \tag{3.1}$$

which, because it coincides with σ on $\mathcal{V}$, will be written simply σ rather than $\bar{\sigma}$, even when applied to non-variable terms. In a similar fashion, substitutions may be applied to equations, as well as to sets of terms or set of equations in the obvious manner.

We shall denote as $\textsc{Subst}_{\mathcal{T}}$ the set of functions in $\mathcal{V} \to \mathcal{T}$ that are substitutions. Because it is non-identical only on a finite number of variables, we can express a substitution σ in $\textsc{Subst}_{\mathcal{T}}$ as a finite set of "*term/variable*" pairs of the form $\{ \sigma(X_k)/X_k \mid X_k \neq \sigma(X_k),\ k = 1, \ldots, n \}$ associating each of a finite set of n variables with a term not equal to it. When the number n of variables is equal to 0, this set is empty, giving the identity on $\mathcal{V}$, which we shall call the empty substitution. Each pair t/X in a substitution's set notation is read "*term t is substituted for all occurrences of variable X.*"

By tradition, rather than the prefix parenthesized notation usually used for functional application, substitution application to a term is written in postfix notation; *viz.*, $t\sigma$ instead of $\sigma(t)$. Thus,

[8] We shall use Prolog's convention of writing variables with capitalized symbols.

as defined by Expression (3.1), a substitution σ is a function in $\mathcal{T} \to \mathcal{T}$ mapping a term t into another one noted $t\sigma$, called its (σ-)*instance*, obtained after replacing all occurrences in t (if any) of variables in $\mathbf{dom}(\sigma)$, the domain of the substitution, by the term associated with this variable by σ. If $\mathbf{var}(\mathbf{ran}(\sigma)) = \emptyset$, σ is called a *ground substitution*, and for any term t in $\mathcal{T}$, $t\sigma \in \mathcal{T}_\emptyset$ and is called a *ground instance* of t.

We define the composition of two substitutions $\sigma \in \textsc{Subst}_\mathcal{T}$ and $\theta \in \textsc{Subst}_\mathcal{T}$ seen as finite sets of non-identical term/variable pairs as the set of pairs written as $\sigma\theta$ and defined in terms of σ and θ as:

$$\begin{aligned} \sigma\theta \quad \stackrel{\text{def}}{=} \quad & \big(\{\, t\theta/X \mid t/X \in \sigma \,\} \setminus \{\, X/X \mid X \in \mathbf{dom}(\sigma) \,\}\big) \\ & \cup \\ & \big(\theta \setminus \{\, u/Y \mid Y \in \mathbf{dom}(\sigma) \,\}\big). \end{aligned} \tag{3.2}$$

3.4 $\mathcal{FOT}$ Subsumption Lattice

The lattice-theoretic properties of $\mathcal{FOT}$s as data structures were initially and independently studied by Reynolds (in [141]) and Plotkin (in [134] and [135]). They both noted that the set $\mathcal{T}$ is preordered by term subsumption (denoted as '$\preceq$'); *viz.*, $t \preceq t'$ (and we say: "t' *subsumes* t") iff there exists a variable substitution $\sigma \in \textsc{Subst}_\mathcal{T}$ such that $t'\sigma = t$. Two $\mathcal{FOT}$s t and t' are considered "*equal up to variable renaming*" (denoted as $t \simeq t'$) whenever both $t \preceq t'$ and $t' \preceq t$. Then, the quotient set of first-order terms modulo variable renaming augmented with a bottom element $\mathcal{T}_{/\simeq} \cup \{\bot_\mathcal{T}\}$ has a lattice structure for subsumption. It has a least element $\bot_\mathcal{T}$ that corresponds to no term in $\mathcal{T}$, since there exists no term that is an instance of all terms. It has a top element which is the set of all variables $\mathcal{V}$, since $\mathcal{V}$ is the class of any variable modulo renaming.

Unification corresponds to the greatest lower bound (**glb**) operation. This is the case also for failure of unification as in this case the **glb** operation results in $\bot_\mathcal{T}$. Given two $\mathcal{FOT}$s t_1 and t_2, unifying them is seeking to compute a substitution of their variables σ such that: $t_1\sigma = t_2\sigma$. Such a substitution, when one exists, is not unique since any less general substitution satisfies the equation; indeed, then $t_1\sigma\theta = t_2\sigma\theta$ for any $\theta \in \textsc{Subst}_\mathcal{T}$. We want only the most general such substitution. That is, for any other substitution $\theta \in \textsc{Subst}_\mathcal{T}$ such that $t_1\theta = t_2\theta$, then necessarily $\theta \preceq \sigma$. This is why it is called the Most General Unifier (**mgu**) of t_1 and t_2 [142]. If no such substitution exists, unification fails and returns $\bot_\mathcal{T}$ as the **glb** of t_1 and t_2, and no substitution. Formally, this is equivalent to instantiating the two terms with a bottom substitution $\bot_{\textsc{Subst}_\mathcal{T}}$ that is added to $\textsc{Subst}_\mathcal{T}$. This new substitution is a zero element in the quotient monoid of substitutions with composition. Namely, for all $\sigma \in \textsc{Subst}_\mathcal{T}$, $\sigma\bot_{\textsc{Subst}_\mathcal{T}} = \bot_{\textsc{Subst}_\mathcal{T}}\sigma = \bot_{\textsc{Subst}_\mathcal{T}}$; which implies that $\bot_{\textsc{Subst}_\mathcal{T}} \preceq \sigma$ for all $\sigma \in \textsc{Subst}_\mathcal{T}$. From this, it follows necessarily that, for all $t \in \mathcal{T}$, $t\bot_{\textsc{Subst}_\mathcal{T}} = \bot_\mathcal{T}$. Thus, when t_1 and t_2 are not unifiable, $\mathbf{mgu}(t_1, t_2) = \bot_{\textsc{Subst}_\mathcal{T}}$.

The dual operation, generalization of two terms, yields a term that is their least upper bound (**lub**) for subsumption. That is, it finds the most specific term t, and two most general substitutions σ_1 and σ_2 such that $t_i = t\sigma_i$ for $i = 1, 2$. Importantly, unlike unification, generalization cannot fail. This is because two term structures having different functors, or two unequal terms one of which is a variable, are always generalizable into a new variable (which may be construed as "*anything*"). Also, generalization yields *two* substitutions rather than just one like for unification.

This is because a variable in the generalizing term t may correspond to two different instantiations in t_1 and t_2. Unification, on the other hand, seeks the *same* instantiation for all the variables in t_1 and t_2 to compute their most general common instance.

This can be summarized as the lattice diagram shown in Figure 3.1. In this diagram, given a pair of terms $\langle t_1, t_2 \rangle$, the pair of substitutions $\langle \sigma_1, \sigma_2 \rangle$ are their respective most general generalizers, and the substitution σ is the pair's most general unifier (**mgu**).

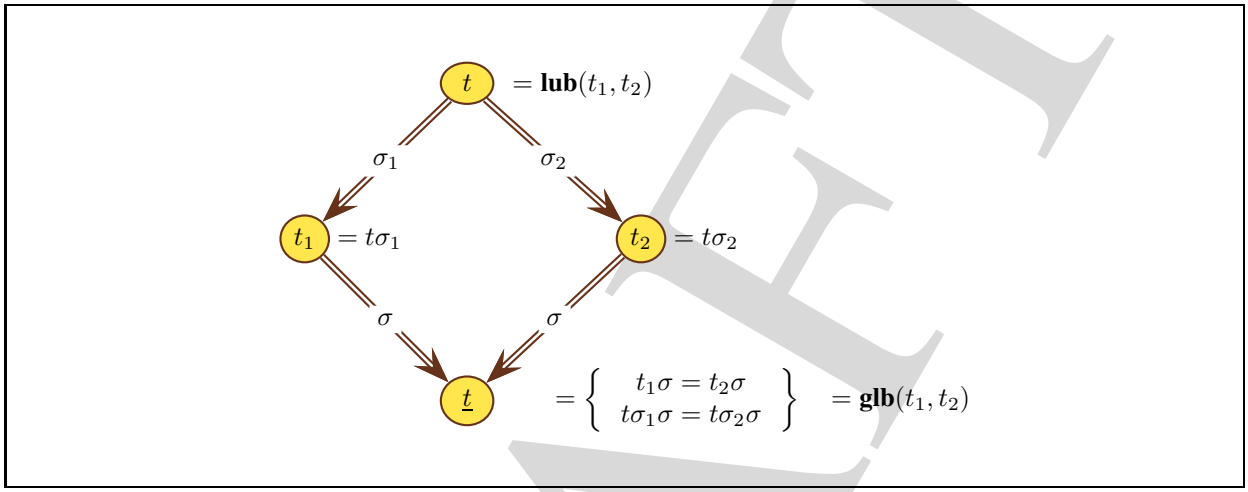


Figure 3.1: Subsumption lattice operations

Example 3.1 $\mathcal{FOT}$ **lattice operations** — Consider the terms t_1 and t_2 defined as:

$$t_1 \stackrel{\text{def}}{=} f(a, g(X_1, b), Y_1, g(a, Y_1)),$$
$$t_2 \stackrel{\text{def}}{=} f(X_2, Y_2, g(X_2, g(X_2, b)), g(X_2, g(a, Z_2))).$$

Their most general unifier $\mathbf{mgu}(t_1, t_2)$ is the substitution σ given by:

$$\sigma = \{\, a/X_2, g(X_1, b)/Y_2, g(a, g(a, b))/Y_1, g(a, b)/Z_2 \,\}$$

and so their greatest lower bound $\mathbf{glb}(t_1, t_2) = \underline{t}$ is given by:

$$\underline{t} = t_1\sigma = t_2\sigma = f(a, g(X_1, b), g(a, g(a, b)), g(a, g(a, g(a, b)))).$$

Dually, their least upper bound $\mathbf{lub}(t_1, t_2) = t$ is given by $t = f(X, Y, Z, g(U, V))$, with their most general generalizers $\langle \sigma_1, \sigma_2 \rangle$ such that:

$$t_1 = t\sigma_1 \text{ with } \sigma_1 = \{\, a/X, g(X_1, b)/Y, Y_1/Z, a/U, Y_1/V \,\}$$
$$t_2 = t\sigma_2 \text{ with } \sigma_2 = \{\, X_2/X, Y_2/Y, g(X_2, g(X_2, b))/Z, X_2/U, g(a, Z_2)/V \,\}.$$

Next, we formalize these lattice operations on $\mathcal{FOT}$s by specifiying them as declarative constraint normalization.

3.5 $\mathcal{FOT}$ Unification Rules

Figure 3.2 illustrates $\mathcal{FOT}$ unification as a commutative diagram constraint. Solving such a con-

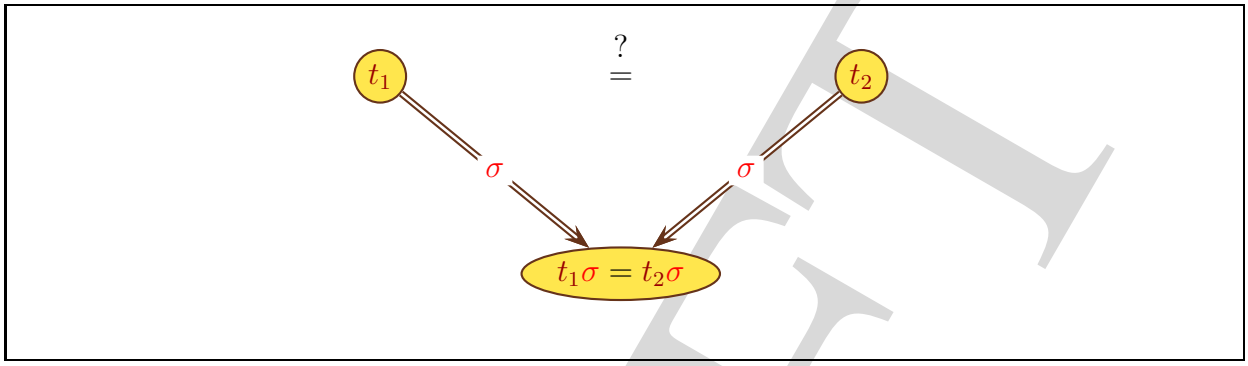


Figure 3.2: $\mathcal{FOT}$ unification as a constraint

straint is done by a system of equation-normalization rules that we shall call Herbrand-Martelli-Montanari [95], [120]. These rules are given in Figure 3.3. Each rule can be proven correct as a solution-preserving transformation of a set of equations. In Rule VARIABLE ELIMINATION, the notation $E[t/X]$ denotes the set of equations E in which the term t is substituted for all occurrences of the variable X.

Thus, we can use these rules to unify two $\mathcal{FOT}$s t_1 and t_2, starting with the singleton set of equations $E \stackrel{\text{def}}{=} \{ t_1 \doteq t_2 \}$.[9] Then, we transform this set of equations using any applicable rule in any order until none applies. This always terminates into a finite set of equations E'. If all the equations in E' are of the form $X \doteq t$ with X occurring nowhere else in E', then this is a most general unifying substitution (up to consistent variable renaming) $\sigma \stackrel{\text{def}}{=} \{ t/X \mid X \doteq t \in E' \}$ solving the original equation (*i.e.*, $t_1\sigma = t_2\sigma$); otherwise, there is no solution.

In the rules of Figure 3.3, Rule VARIABLE ELIMINATION has the side condition $X \notin \mathbf{var}(t)$ to prevent cyclic terms (such as, *e.g.*, $X = f(X)$) whose presence indicates no $\mathcal{FOT}$ solutions. This condition could be omitted if wished, thus extending the set of $\mathcal{FOT}$s and solutions of equations to rational $\mathcal{FOT}$s — also called "*infinite trees*" (see, *e.g.*, [165], [103], [62]).

Example 3.2 $\mathcal{FOT}$ **unification** — Consider the equation set $\{ t_1 \doteq t_2 \}$ for the terms t_1 and t_2 of Example 3.1:

$$\{ f(a, g(X_1, b), Y_1, g(a, Y_1)) \doteq f(X_2, Y_2, g(X_2, g(X_2, b)), g(X_2, g(a, Z_2))) \}$$

and let us apply the rules of Figure 3.3:

- Rule TERM DECOMPOSITION:

$$\{ a \doteq X_2, g(X_1, b) \doteq Y_2, Y_1 \doteq g(X_2, g(X_2, b)), g(a, Y_1) \doteq g(X_2, g(a, Z_2)) \};$$

[9] In such equations, we use the notation $t_1 \doteq t_2$ not to confuse it with the equality symbol "=" (at the meta-level).

TERM DECOMPOSITION

$[n \geq 0]$

$$\frac{E \cup \{ f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n) \}}{E \cup \{ s_1 \doteq t_1, \ldots, s_n \doteq t_n \}}$$

VARIABLE ERASURE

$$\frac{E \cup \{ X \doteq X \}}{E}$$

VARIABLE ELIMINATION

$[X \notin \mathbf{var}(t);\ X \text{ occurs in } E]$

$$\frac{E \cup \{ X \doteq t \}}{E[t/X] \cup \{ X \doteq t \}}$$

EQUATION ORIENTATION

$[t \notin \mathcal{V}]$

$$\frac{E \cup \{ t \doteq X \}}{E \cup \{ X \doteq t \}}$$

Figure 3.3: Herbrand-Martelli-Montanari unification rules

- Rule **EQUATION ORIENTATION** to $a \doteq X_2$:

  $$\{ X_2 \doteq a, g(X_1, b) \doteq Y_2, Y_1 \doteq g(X_2, g(X_2, b)), g(a, Y_1) \doteq g(X_2, g(a, Z_2)) \};$$

- Rule **VARIABLE ELIMINATION** to $X_2 \doteq a$:

  $$\{ X_2 \doteq a, g(X_1, b) \doteq Y_2, Y_1 \doteq g(a, g(a, b)), g(a, Y_1) \doteq g(a, g(a, Z_2)) \};$$

- Rule **EQUATION ORIENTATION** to $g(X_1, b) \doteq Y_2$:

  $$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), g(a, Y_1) \doteq g(a, g(a, Z_2)) \};$$

- Rule **VARIABLE ELIMINATION** to $Y_1 \doteq g(a, g(a, b))$:

  $$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), g(a, g(a, g(a, b))) \doteq g(a, g(a, Z_2)) \};$$

- Rule **TERM DECOMPOSITION** to $g(a, g(a, g(a, b))) \doteq g(a, g(a, Z_2))$:

  $$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), a \doteq a, g(a, g(a, b)) \doteq g(a, Z_2) \};$$

- Rule **TERM DECOMPOSITION** to $a \doteq a$:

  $$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), g(a, g(a, b)) \doteq g(a, Z_2) \};$$

- Rule **TERM DECOMPOSITION** to $g(a, g(a, b)) \doteq g(a, Z_2)$:

  $$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), a \doteq a, g(a, b) \doteq Z_2 \};$$

- Rule **TERM DECOMPOSITION** to $a \doteq a$:

$$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), g(a, b) \doteq Z_2 \};$$

- Rule **EQUATION ORIENTATION** to $g(a, b) \doteq Z_2$:

$$\{ X_2 \doteq a, Y_2 \doteq g(X_1, b), Y_1 \doteq g(a, g(a, b)), Z_2 \doteq g(a, b) \}.$$

This last equation set is in normal form defining the substitution

$$\sigma = \{ a/X_2, g(X_1, b)/Y_2, g(a, g(a, b))/Y_1, g(a, b)/Z_2 \}.$$

So the greatest lower bound $\underline{t} \stackrel{\text{def}}{=} \mathbf{glb}(t_1, t_2)$ of:

$$t_1 \stackrel{\text{def}}{=} f(a, g(X_1, b), Y_1, g(a, Y_1))$$

and:

$$t_2 \stackrel{\text{def}}{=} f(X_2, Y_2, g(X_2, g(X_2, b)), g(X_2, g(a, Z_2)))$$

is given by:

$$\underline{t} = t_1\sigma = t_2\sigma = f(a, g(X_1, b), g(a, g(a, b)), g(a, g(a, g(a, b)))).$$

3.6 $\mathcal{FOT}$ Generalization Rules

Next, we present a set of constraint normalization rules for $\mathcal{FOT}$ generalization which are equivalent to the procedural method of Reynolds and Plotkin. The advantage of specifying this operation in this manner rather than procedurally as done originally by Reynolds and Plotkin is that each rule or axiom relates a pair of prior substitutions to a pair of posterior substitutions based only on local syntactic-pattern properties of the terms to generalize, and this without resorting to side-effects on global structures. In this way, the terms and substitutions involved are derived as solutions of logical syntactic constraints. In addition, correctness of the so-specified operation is made much easier to establish since we only need to prove each rule's correctness independently of that of the others. Finally, the rules also provide an effective means for the derivation of an operational semantics for the so-specified operation by constraint solving, without need for control specification as any applicable rule may be invoked in any order.[10]

DEFINITION 3.1 (GENERALIZATION JUDGMENT) *A generalization judgment is an expression of the form:*

$$\begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} \vdash \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} t \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \tag{3.3}$$

where $\sigma_i \in$ **SUBST** $_{\mathcal{T}}$, $\theta_i \in$ **SUBST** $_{\mathcal{T}}$, $t_i \in \mathcal{T}$ $(i = 1, 2)$, *and* $t \in \mathcal{T}$.

[10] Such as the Herbrand-Martelli-Montanari rules *w.r.t.* to Robinson's procedural unification algorithm.

Informally, it reads: "*given two prior substitutions σ_1 and σ_2, the term t is the least generalization of terms $t_1\sigma_1$ and $t_2\sigma_2$ with posterior substitutions θ_1 and θ_2.*" How all the constituents of such a generalization judgment must be related to constitute what we shall consider a valid judgment, is defined next.

DEFINITION 3.2 (GENERALIZATION JUDGMENT VALIDITY) *A $\mathcal{FOT}$ generalization judgment such as (3.3) is said to be valid whenever, for $i = 1, 2$:*

1. $t_i\sigma_i = t\theta_i$*; and,*
2. $\exists\, \delta_i \in$ **SUBST**$_{\mathcal{T}}$ *s.t.* $t_i = t\delta_i$ *and* $\theta_i = \delta_i\sigma_i$ *(i.e.,* $t_i \preceq t$ *and* $\theta_i \preceq \sigma_i$*).*

Figure 3.4 illustrates the validity of a $\mathcal{FOT}$ generalization judgment as a commutative diagram constraint.

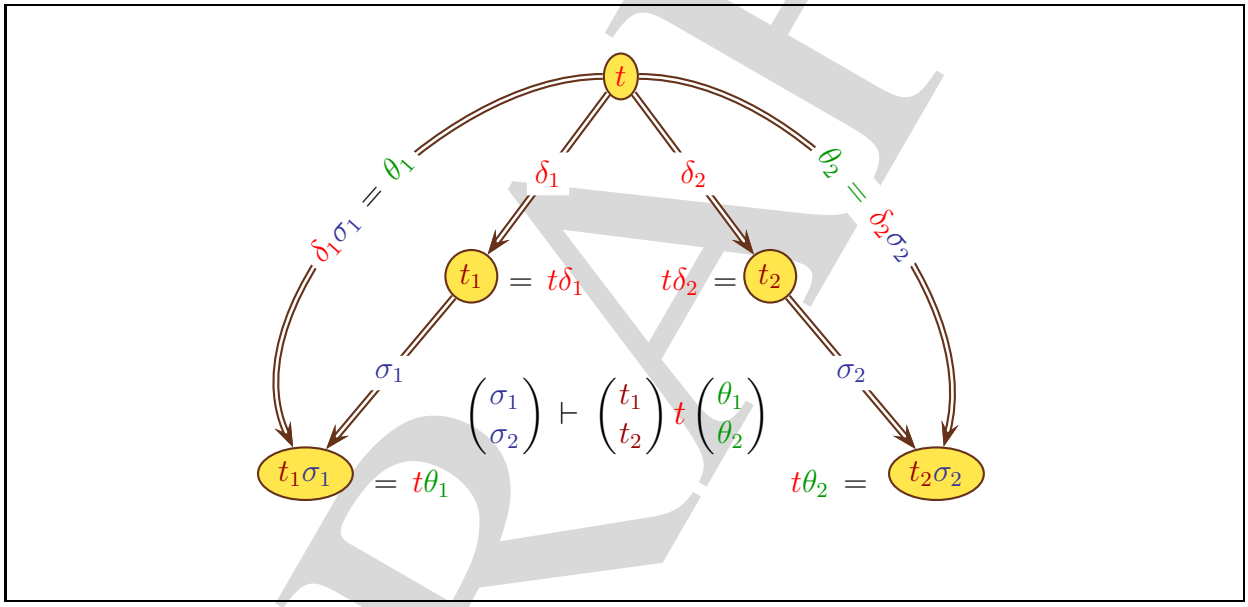


Figure 3.4: $\mathcal{FOT}$ generalization judgment validity as a constraint

DEFINITION 3.3 (TRIVIAL $\mathcal{FOT}$ GENERALIZATION JUDGMENT) *The $\mathcal{FOT}$ generalization judgment:*

$$\texttt{true} \stackrel{def}{=} \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}t\\ t\end{pmatrix} t \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \tag{3.4}$$

where t is an arbitrary term in $\mathcal{T}$ is called a "trivial $\mathcal{FOT}$ generalization judgment."

LEMMA 3.1 (TRIVIAL $\mathcal{FOT}$ GENERALIZATION JUDGMENT VALIDITY) *The trivial $\mathcal{FOT}$ generalization judgment* true *is always valid.*

PROOF This follows from Definition 3.2 since in this particular case the equations of the first condition of Definition (3.2) becomes $t = t$, which is trivially true for any term $t \in \mathcal{T}$. □

Contrary to unification normalization rules which are expressed as conditional rewrite rules whereby a prior form (the "numerator") is related to a posterior form (the "denominator"), these normalization rules are more naturally rendered as (conditional) Horn clauses of judgments (*i.e.*, Prolog clauses of judgments). This is as convenient as rewrite rules since a Prolog-like operational semantics can then readily provide an effective interpretation.[11] Thus, a generalization rule is of the form:

$$\frac{[\phi] \quad J_1 \quad \ldots \quad J_n}{J} \tag{3.5}$$

where ϕ is an optional side meta-condition, and $J, J_1, \ldots, J_n$ are judgments, and it reads, "*whenever the side condition ϕ holds, if all the n antecedent judgments $J_1, \ldots, J_n$ are valid, then the consequent judgment J is also valid.*" Such a generalization rule without a specified antecedent (a "numerator") is called a "*generalization axiom.*" Such an axiom is said to be valid iff its consequent (the "denominator") is valid whenever its optional side condition holds. It is equivalent to a rule where the only antecedent is the trivial generalization judgment $\mathtt{true}$.

DEFINITION 3.4 (GENERALIZATION RULE CORRECTNESS) *A generalization rule such as Rule (3.5) is correct iff J_k is a valid judgment for all $k = 1, \ldots, n$ implies that J is a valid judgment, whenever the side condition ϕ holds.*

Given t_1 and t_2 two $\mathcal{FOT}$s, in order to find the most specific term t and most general substitutions σ_i, $i = 1, 2$, such that $t\sigma_i = t_i$, $i = 1, 2$, one needs to establish the generalization judgment:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \vdash \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} t \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix}. \tag{3.6}$$

In other words, this expresses the upper half of Figure 3.1 whereby $t = \mathbf{lub}(t_1, t_2)$, with most general substitutions σ_1 and σ_2. We give a complete set of normalization axioms and rule for generalization for all syntactic patterns in Figure 3.5.

Rule "**EQUAL FUNCTORS**" specifies a sequence of judgments constrained as a sequence. It does so exactly as a so-called "Definite Clause Grammar" (or DCG) rule does with a Prolog clause.[12] This rule uses an "*unapply*" operation ('↑') on a pair of terms (t_1, t_2) given a pair

[11] This operational semantics is also efficient because it does not need backtracking as long as the complete set of conditions of a ruleset covers all but mutually exclusive syntactic patterns.

[12] A DCG rule (see https://www.metalevel.at/prolog/dcg) is a Horn rule expressing constraints on a sequence of words constituting a sentence. The judgment sequencing in the rules we define uses exactly the same kind of constraint: the posterior pair of substitutions of a judgment must match the prior pair of substitutions of the judgment following it. However, contrary to a DCG rule that constrains an *ordered* sequence of constituents, the order of constraints on the antecedent judgments on the arguments is arbitrary. We choose the same order as that of the arguments as it is the most natural, but it could be any of its permutations as long as the sequence's posterior/prior constraints are consistent with the chosen argument ordering.

EQUAL VARIABLES

$$\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \vdash \begin{pmatrix}X\\ X\end{pmatrix} X \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}$$

VARIABLE-TERM

$[t_1 \in \mathcal{V} \text{ or } t_2 \in \mathcal{V};\ \ t_1 \neq t_2;\ \ X \text{ is new}]$

$$\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \vdash \begin{pmatrix}t_1\\ t_2\end{pmatrix} X \begin{pmatrix}\sigma_1\{\, t_1/X \,\}\\ \sigma_2\{\, t_2/X \,\}\end{pmatrix}$$

UNEQUAL FUNCTORS

$[m \geq 0, n \geq 0;\ \ m \neq n \text{ or } f \neq g;\ \ X \text{ is new}]$

$$\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \vdash \begin{pmatrix}f(s_1,\ldots,s_m)\\ g(t_1,\ldots,t_n)\end{pmatrix} X \begin{pmatrix}\sigma_1\{\, f(s_1,\ldots,s_m)/X \,\}\\ \sigma_2\{\, g(t_1,\ldots,t_n)/X \,\}\end{pmatrix}$$

EQUAL FUNCTORS

$[n \geq 0]$

$$\frac{\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \vdash \begin{pmatrix}s_1\\ t_1\end{pmatrix} \!\uparrow\! \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} u_1 \begin{pmatrix}\sigma_1^1\\ \sigma_2^1\end{pmatrix} \quad \cdots \quad \begin{pmatrix}\sigma_1^{n-1}\\ \sigma_2^{n-1}\end{pmatrix} \vdash \begin{pmatrix}s_n\\ t_n\end{pmatrix} \!\uparrow\! \begin{pmatrix}\sigma_1^{n-1}\\ \sigma_2^{n-1}\end{pmatrix} u_n \begin{pmatrix}\sigma_1^n\\ \sigma_2^n\end{pmatrix}}{\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \vdash \begin{pmatrix}f(s_1,\ldots,s_n)\\ f(t_1,\ldots,t_n)\end{pmatrix} f(u_1,\ldots,u_n) \begin{pmatrix}\sigma_1^n\\ \sigma_2^n\end{pmatrix}}$$

Figure 3.5: Generalization axioms and rule

of substitutions (σ_1, σ_2). It may be conceived as (and in fact is) the result of simultaneously "*unapplying*" σ_i from t_i into a common variable X only if such X is bound to t_i by σ_i, for $i = 1, 2$. If there is no such a variable, it is the identity. This operation avoids the introduction of a new variable when generalizing two already generalized terms. Formally, this is defined as:

$$\begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \uparrow \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} \stackrel{\text{def}}{=} \begin{cases} \begin{pmatrix} X \\ X \end{pmatrix} & \text{if } \exists\, X \in \mathcal{V},\ t_i = X\sigma_i,\ \text{for } i = 1, 2; \\ \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} & \text{otherwise.} \end{cases} \tag{3.7}$$

Note also that Rule "**Equal Functors**" is defined for $n \geq 0$. For $n = 0$, it becomes the following axiom for any constant c and any two substitutions σ_i, $i = 1, 2$:

$$\begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} \vdash \begin{pmatrix} c \\ c \end{pmatrix} c \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix}. \tag{3.8}$$

Referring to the axioms (seen as rules with no antecedent) and the rule of Figure 3.5, we first establish the following fact.

Lemma 3.2 *In Rule* **Equal Functors** *of Figure 3.5, taking* $\sigma_i^0 \stackrel{def}{=} \sigma_i$, *for* $i = 1, 2$, *the substitutions* $\sigma_i^0, \ldots, \sigma_i^n$ *are such that, for all* k, $1 \leq k \leq n$, $\sigma_i^k \preceq \sigma_i^{k-1}$, *for* $i = 1, 2$.

Proof We proceed by induction on the depth d of the terms; *i.e.*, we consider only terms of depth less than or equal to d.

1. $d = 0$: This limits terms to constants and variables. The inequality between prior and posterior substitutions is satisfied for the first three axioms: the posterior substitutions are all either equal to the corresponding prior substitutions or of the form $\theta = \sigma\{t/X\}$ where X is a new variable and σ is the corresponding prior substitution; that is, $\theta \preceq \sigma$. As well, when limited to terms of 0 depth, Rule **Equal Functors** becomes the single judgment Axiom (3.8), which preserves the substitutions.
2. $d > 0$: Let us assume that this is true for all terms of depth strictly less than d. We now consider two terms at least one of which is of depth d. For Axiom **Equal Variables**, the same argument given above for the case $d = 0$ justifies concluding that $\theta \preceq \sigma$, since then $\theta = \sigma$. For Axiom **Variable-Term** and Axiom **Unequal Functors**, this true for terms t_1 and t_2 of any depth since the posterior substitutions are both less general then the corresponding prior substitutions. As for Rule **Equal Functors**, there are two possible cases for the generalized terms in its consequent (the "denominator"):
   (a) $n = 0$: then, the conclusion follows true by Axiom (3.8).
   (b) $n \geq 0$: since the unapply operation (3.7) yields either a pair of terms having the same depth as the corresponding terms it is applied to, or 0 (because it can only be a new variable), we can say that all the terms of unapplied pairs of arguments in the judgments of the rule's antecedent (the "numerator") are of depth at most

$d-1$. Therefore, all the terms in the n antecedent judgments satisfy our inductive hypothesis; namely: $\sigma_i^k \preceq \sigma_i^{k-1}$, for all $k = 1, \ldots, n$. Then, by transitivity of the "more general" ordering on substitutions, the conclusion follows.

Hence, this establishes that, for both $i = 1, 2$, σ_i^k is monotonically refined from more general to less as k increases from 1 to n. □

From this lemma, the following corollary follows by transitivity of the $\preceq$ preorder on substitutions.

COROLLARY 3.1 *In Rule* **EQUAL FUNCTORS**, *the substitutions σ_i^k are such that, for all k, $1 \leq k \leq n$, $\sigma_i^n \preceq \sigma_i^{n-1} \preceq \ldots \preceq \sigma_i^1 \preceq \sigma_i^0$, for $i = 1, 2$.*

It is also verified in the proof of the following theorem.

THEOREM 3.1 *The axioms and the rule of Figure 3.5 are correct.*

PROOF We must show that they satisfy the conditions of Definition 3.4. For each of the three axioms of Figure 3.5 this means that they must be always valid as judgments, satisfying the conditions of Definition 3.2, which are:

- *Condition 1*: $t_i\sigma_i = t\theta_i$,
- *Condition 2*: $t_i \preceq t$ and $\theta_i \preceq \sigma_i$

for $i = 1, 2$, for a generalization judgment such as (3.3) in Definition 3.1. These conditions for the axioms and the rule of Figure 3.5 translate as the following.

Condition 1.

- **EQUAL VARIABLES**: it amounts to the two identities $X\sigma_i = X\sigma_i$, $i = 1, 2$;
- **VARIABLE-TERM**: it amounts to the two identities $t_i\sigma_i = t_i\sigma_i$, $i = 1, 2$;
- **UNEQUAL FUNCTORS**: it amounts to the two equations:

$$\begin{aligned} f(s_1, \ldots, s_n)\sigma_1 &= X\sigma_1\{\, f(s_1, \ldots, s_n)/X \,\}, \\ g(t_1, \ldots, t_n)\sigma_2 &= X\sigma_2\{\, g(t_1, \ldots, t_n)/X \,\}, \end{aligned}$$

which, because X is a new variable that does not occurs in either σ_1 or σ_2, can be simplified to the identities:

$$\begin{aligned} f(s_1, \ldots, s_n) &= f(s_1, \ldots, s_n), \\ g(t_1, \ldots, t_n) &= g(t_1, \ldots, t_n). \end{aligned}$$

Condition 2. All threes cases are tautologies:

- **EQUAL VARIABLES**: $X \preceq X$ and $\sigma_i \preceq \sigma_i$, $i = 1, 2$;
- **VARIABLE-TERM**: $t_i \preceq X$ and $\sigma_i\{t_i/X\} \preceq \sigma_i$, $i = 1, 2$;

– **UNEQUAL FUNCTORS**:

$$f(s_1,\ldots,s_n) \preceq X \text{ and } \sigma_1\{\, f(s_1,\ldots,s_n)/X \,\} \preceq \sigma_1,$$
$$g(s_1,\ldots,s_n) \preceq X \text{ and } \sigma_2\{\, g(s_1,\ldots,s_n)/X \,\} \preceq \sigma_2.$$

As for Rule **EQUAL FUNCTORS**, as required by Definition 3.4, we must show that if all the judgments in the numerator are valid, then the judgment in the denominator must be valid too. Let us proceed by induction on the argument-position number k, for $k = 1,\ldots,n$.

For $n = 0$, this rule becomes Axiom (3.8), a judgment that is trivially valid since the conditions of Definition 3.2 become the identity $c = c$, the term inequality $c \preceq c$, and the substitution inequalities $\sigma_i \preceq \sigma_i$, for $i = 1, 2$.

For $n > 0$, a fuzzy judgment in the rule's antecedent, for each argument-position $k = 1,\ldots,n$, is of the form:

$$\begin{pmatrix}\sigma_1^{k-1}\\ \sigma_2^{k-1}\end{pmatrix} \vdash \begin{pmatrix}s_k\\ t_k\end{pmatrix} \uparrow \begin{pmatrix}\sigma_1^{k-1}\\ \sigma_2^{k-1}\end{pmatrix} u_k \begin{pmatrix}\sigma_1^{k}\\ \sigma_2^{k}\end{pmatrix}$$

that is, the form given by Definition 3.1, whose formal validity conditions are given by Definition 3.2, which in the above case is equivalent to:

$$\begin{pmatrix}v_1^{k}\\ v_2^{k}\end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix}s_k\\ t_k\end{pmatrix} \uparrow \begin{pmatrix}\sigma_1^{k-1}\\ \sigma_2^{k-1}\end{pmatrix} \text{ and } \begin{pmatrix}\sigma_1^{k-1}\\ \sigma_2^{k-1}\end{pmatrix} \vdash \begin{pmatrix}v_1^{k}\\ v_2^{k}\end{pmatrix} u_k \begin{pmatrix}\sigma_1^{k}\\ \sigma_2^{k}\end{pmatrix}.$$

Let us now assume that all the judgments in the rule's antecedent are valid. That is, for $k = 1,\ldots,n$, for $i = 1, 2$ (taking $\sigma_i^0 \stackrel{\text{def}}{=} \sigma_i$):

– Condition 1 of Definition 3.2 holds:

$$u_k\sigma_i^k = v_i^k\sigma_i^{k-1}; \tag{3.9}$$

– Condition 2 of Definition 3.2 holds:

$$v_i^k \preceq u_k \text{ and } \sigma_i^k \preceq \sigma_i^{k-1}. \tag{3.10}$$

Condition 1. By Equation (3.7), this means that for all $k = 1,\ldots,n$:

$$\begin{pmatrix}v_1^{k}\\ v_2^{k}\end{pmatrix} = \begin{cases} \begin{pmatrix}X\\ X\end{pmatrix} & \text{if } s_k = X\sigma_1^{k-1} \text{ and } t_k = X\sigma_2^{k-1} \text{ for some variable} X; \\ \begin{pmatrix}s_k\\ t_k\end{pmatrix} & \text{otherwise.}\end{cases}$$

In other words, for each $k = 1,\ldots,n$, there are two cases:

1. $s_k = X\sigma_1^{k-1}$ and $t_k = X\sigma_2^{k-1}$ for some variable X; then, by Axiom **EQUAL VARIABLES**, we must have $u_k = X$, and $\sigma_i^k = \sigma_i^{k-1}$ for $i = 1, 2$; and therefore Equations (3.9) entail:

$$s_k\sigma_1^{k-1} = X\sigma_1^{k-1}\sigma_1^{k-1} = X\sigma_1^{k-1} = X\sigma_1^{k} = u_k\sigma_1^k$$
$$t_k\sigma_2^{k-1} = X\sigma_2^{k-1}\sigma_2^{k-1} = X\sigma_2^{k-1} = X\sigma_2^{k} = u_k\sigma_2^k.$$

2. There is no such variable X; in which case, Equations (3.9) also become:

$$s_k\sigma_1^{k-1} = u_k\sigma_1^k$$
$$t_k\sigma_2^{k-1} = u_k\sigma_2^k.$$

Thus, by the only non-identical transformation relating prior and posteriors substitutions in the axioms, for any argument position k, $1 \leq k \leq n$, we have:

$$\sigma_i^k = \sigma_i^0\{\,\tau_1/X_1\,\}\ldots\{\,\tau_\ell/X_\ell\,\}$$

where each of the variables $X_1 \ldots X_\ell$, with $0 \leq \ell$, is a variable possibly introduced in the validity of the judgment corresponding to some argument preceding position k. Therefore, for any argument position k, $1 \leq k \leq n$:

$$s_k\sigma_1^0 = s_k\sigma_1^1 = \ldots = s_k\sigma_1^{k-1}$$
$$t_k\sigma_2^0 = t_k\sigma_2^1 = \ldots = t_k\sigma_2^{k-1}$$

as well as:

$$u_k\sigma_1^k = u_k\sigma_1^{k+1} = \ldots = u_k\sigma_1^n$$
$$u_k\sigma_2^k = u_k\sigma_2^{k+1} = \ldots = u_k\sigma_2^n$$

because σ_i^k affects only new variables introduced in some axioms satisfying the validity of a subterm of argument at position k; and because the same variable in u_k is always instantiated by the same term, and thus as well all at higher argument positions.

This means that in both cases we have, for all $k = 1, \ldots, n$:

$$s_k\sigma_1^0 = u_k\sigma_1^n,$$
$$t_k\sigma_2^0 = u_k\sigma_2^n.$$

Therefore, for $k = n$:

$$f(s_1, \ldots, s_n)\sigma_1^0 = f(u_1, \ldots, u_n)\sigma_1^n,$$
$$f(t_1, \ldots, t_n)\sigma_2^0 = f(u_1, \ldots, u_n)\sigma_2^n.$$

This proves Condition 1.

Condition 2. By transitivity of the $\leq$ ordering on approximation degrees and that of the $\preceq$ preorder on $\textsc{Subst}_{\mathcal{T}}$, both parts of Condition 2 of our induction hypothesis (3.10) implies that:

$$f(s_1, \ldots, s_n) \preceq f(u_1, \ldots, u_n),$$
$$f(t_1, \ldots, t_n) \preceq f(u_1, \ldots, u_n),$$

and:

$$\sigma_i \preceq \sigma_i^n \text{ for } i = 1, 2;$$

which completes the proof. □

In particular, with empty prior substitutions, we obtain the following corollary.

COROLLARY 3.2 ($\mathcal{FOT}$ GENERALIZATION) *Whenever the judgment*

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}t_1\\ t_2\end{pmatrix} t \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}$$

is valid, then $t\sigma_i = t_i$, *for* $i = 1, 2$.

Example 3.3 $\mathcal{FOT}$ **generalization** — Consider the terms $f(a, a, a)$ and $f(b, c, c)$.

- Let us find term t and substitutions σ_1 and σ_2 such that $t\sigma_1 = f(a, a, a)$ and $t\sigma_2 = f(b, c, c)$; that is, let us try to solve the following constraint problem:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}f(a,a,a)\\ f(b,c,c)\end{pmatrix} t \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}$$

- By Rule **EQUAL FUNCTORS**, we must have $t = f(u_1, u_2, u_3)$ since:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}f(a,a,a)\\ f(b,c,c)\end{pmatrix} f(u_1,u_2,u_3) \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}$$

  where:

  - u_1 is the generalization of $\begin{pmatrix}a\\ b\end{pmatrix} \uparrow \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}$; that is, of a and b; and by Rule **UNEQUAL FUNCTORS**:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}a\\ b\end{pmatrix} X \begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix}$$

    therefore $u_1 = X$;

  - u_2 is the generalization of $\begin{pmatrix}a\\ c\end{pmatrix} \uparrow \begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix}$; that is, of a and c; and by Rule **UNEQUAL FUNCTORS**:

$$\begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix} \vdash \begin{pmatrix}a\\ c\end{pmatrix} Y \begin{pmatrix}\{\,a/X, a/Y\,\}\\ \{\,b/X, c/Y\,\}\end{pmatrix}$$

    so $u_2 = Y$;

  - u_3 is the generalization of $\begin{pmatrix}a\\ c\end{pmatrix} \uparrow \begin{pmatrix}\{\,a/X, a/Y\,\}\\ \{\,b/X, c/Y\,\}\end{pmatrix}$; that is, of Y and Y; and by Rule **EQUAL VARIABLES**:

$$\begin{pmatrix}\{\,a/X, a/Y\,\}\\ \{\,b/X, c/Y\,\}\end{pmatrix} \vdash \begin{pmatrix}Y\\ Y\end{pmatrix} Y \begin{pmatrix}\{\,a/X, a/Y\,\}\\ \{\,b/X, c/Y\,\}\end{pmatrix}$$

    so $u_3 = Y$;

- therefore, this yields:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}f(a,a,a)\\ f(b,c,c)\end{pmatrix} f(X,Y,Y) \begin{pmatrix}\{\,a/X, a/Y\,\}\\ \{\,b/X, c/Y\,\}\end{pmatrix}$$

that is $t = f(X,Y,Y)$ with $\sigma_1 = \{\,a/X, a/Y\,\}$ such that $t\sigma_1 = f(a,a,a)$, and $\sigma_2 = \{\,b/X, c/Y\,\}$ such that $t\sigma_2 = f(b,c,c)$.

Example 3.4 Generalization of ground $\mathcal{FOT}$s — Let us now consider the terms $f(a, g(b,a), b))$ and $f(b, g(a,b), a))$;

- let us find term t and substitutions σ_1 and σ_2 s.t. $t\sigma_1 = f(a,\, g(b,a),\, b))$ and $t\sigma_2 = f(b,\, g(a,b),\, a)$; *i.e.*, let us try to solve the following constraint problem:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}f(a,g(b,a),b)\\ f(b,g(a,b),a)\end{pmatrix} t \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}$$

- By Rule **EQUAL FUNCTORS**, we must have $t = f(u_1,u_2,u_3)$ since:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}f(a,g(b,a),b)\\ f(b,g(a,b),a)\end{pmatrix} f(u_1,u_2,u_3) \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \text{ where:}$$

  - u_1 is the generalization of $\begin{pmatrix}a\\ b\end{pmatrix} \uparrow \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}$; that is of a and b; and by Rule **UNEQUAL FUNCTORS**:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}a\\ b\end{pmatrix} X \begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix} \text{ and therefore } u_1 = X;$$

  - $u_2 = g(v_1,v_2)$ is the generalization of $\begin{pmatrix}g(b,a)\\ g(a,b)\end{pmatrix} \uparrow \begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix}$; that is, of $g(b,X)$ and $g(a,X)$; and by Rule **EQUAL FUNCTORS**:
    - v_1 is the generalization of $\begin{pmatrix}b\\ a\end{pmatrix} \uparrow \begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix}$; that is, of b and a; and by Rule **UNEQUAL FUNCTORS**:

$$\begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix} \vdash \begin{pmatrix}b\\ a\end{pmatrix} Y \begin{pmatrix}\{\,a/X, b/Y\,\}\\ \{\,b/X, a/Y\,\}\end{pmatrix} \text{ so } v_1 = Y;$$

    - v_2 is the generalization of $\begin{pmatrix}a\\ b\end{pmatrix} \uparrow \begin{pmatrix}\{\,a/X, b/Y\,\}\\ \{\,b/X, a/Y\,\}\end{pmatrix}$; that is, of X and X; and by Rule **EQUAL VARIABLES**:

$$\begin{pmatrix}\{\,a/X, b/Y\,\}\\ \{\,b/X, a/Y\,\}\end{pmatrix} \vdash \begin{pmatrix}X\\ X\end{pmatrix} X \begin{pmatrix}\{\,a/X, b/Y\,\}\\ \{\,b/X, a/Y\,\}\end{pmatrix} \text{ so } v_2 = X;$$

  therefore:

$$\begin{pmatrix}\{\,a/X\,\}\\ \{\,b/X\,\}\end{pmatrix} \vdash \begin{pmatrix}g(b,X)\\ g(a,X)\end{pmatrix} g(Y,X) \begin{pmatrix}\{\,a/X, b/Y\,\}\\ \{\,b/X, a/Y\,\}\end{pmatrix} \text{ so } u_2 = g(Y,X);$$

– u_3 is the generalization of $\begin{pmatrix} b \\ a \end{pmatrix} \uparrow \begin{pmatrix} \{\, a/X, b/Y \,\} \\ \{\, b/X, a/Y \,\} \end{pmatrix}$; that is, of Y and Y; and by Rule **EQUAL VARIABLES**:

$$\begin{pmatrix} \{\, a/X, b/Y \,\} \\ \{\, b/X, a/Y \,\} \end{pmatrix} \vdash \begin{pmatrix} Y \\ Y \end{pmatrix} Y \begin{pmatrix} \{\, a/X, b/Y \,\} \\ \{\, b/X, a/Y \,\} \end{pmatrix} \quad \text{so} \quad u_3 = Y;$$

- therefore, this yields:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \vdash \begin{pmatrix} f(a, g(b,a), b) \\ f(b, g(a,b), a) \end{pmatrix} f(X, g(Y,X), Y) \begin{pmatrix} \{\, a/X, b/Y \,\} \\ \{\, b/X, a/Y \,\} \end{pmatrix}$$

that is $t = f(X, g(Y,X), Y)$ with $\sigma_1 = \{\, a/X, b/Y \,\}$ such that $t\sigma_1 = f(a, g(b,a), b)$, and $\sigma_2 = \{\, b/X, a/Y \,\}$ such that $t\sigma_2 = f(b, g(a,b), a)$.

Example 3.5 Generalization of non-ground $\mathcal{FOT}$s — Let us apply the $\mathcal{FOT}$ generalization axioms and rules of Figure 3.5 to the following $\mathcal{FOT}$s:

$$t_1 \stackrel{\text{def}}{=} h(f(a, X_1), g(X_1, b), f(Y_1, Y_1)), \quad \text{and} \quad t_2 \stackrel{\text{def}}{=} h(X_2, X_2, g(c,d)).$$

- Let us find term t and substitutions σ_1 and σ_2 such that $t\sigma_1 = h(f(a, X_1), g(X_1, b), f(Y_1, Y_1))$ and $t\sigma_2 = h(X_2, X_2, g(c,d))$; that is, let us try to solve the constraint problem:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \vdash \begin{pmatrix} h(f(a, X_1), g(X_1, b), f(Y_1, Y_1)) \\ h(X_2, X_2, g(c,d)) \end{pmatrix} t \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix}.$$

- By Rule **EQUAL FUNCTORS**, we must have $t = h(u_1, u_2, u_3)$ since:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \vdash \begin{pmatrix} h(f(a, X_1), g(X_1, b), f(Y_1, Y_1)) \\ h(X_2, X_2, g(c,d)) \end{pmatrix} h(u_1, u_2, u_3) \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix} \quad \text{where:}$$

– u_1 is the generalization of $\begin{pmatrix} f(a, X_1) \\ X_2 \end{pmatrix} \uparrow \begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix}$; that is of $f(a, X_1)$ and X_2; and by Rule **VARIABLE-TERM**:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \vdash \begin{pmatrix} f(a, X_1) \\ X_2 \end{pmatrix} X \begin{pmatrix} \{\, f(a, X_1)/X \,\} \\ \{\, X_2/X \,\} \end{pmatrix} \quad \text{so} \quad u_1 = X;$$

– u_2 is the generalization of $\begin{pmatrix} g(X_1, b) \\ X_2 \end{pmatrix} \uparrow \begin{pmatrix} \{\, f(a, X_1)/X \,\} \\ \{\, X_2/X \,\} \end{pmatrix}$; that is, of $g(X_1, b)$ and X_2; and by Rule **VARIABLE-TERM**:

$$\begin{pmatrix} \{\, f(a, X_1)/X \,\} \\ \{\, X_2/X \,\} \end{pmatrix} \vdash \begin{pmatrix} g(X_1, b) \\ X_2 \end{pmatrix} Y \begin{pmatrix} \{\ldots, g(X_1, b)/Y \,\} \\ \{\ldots, X_2/Y \,\} \end{pmatrix} \quad \text{so} \quad u_2 = Y;$$

– u_3 is the generalization of $\begin{pmatrix} f(Y_1, Y_1) \\ g(c,d) \end{pmatrix} \uparrow \begin{pmatrix} \{\, f(a, X_1)/X, g(X_1, b)/Y \,\} \\ \{\, X_2/X, X_2/Y \,\} \end{pmatrix}$; that is, of $f(Y_1, Y_1)$ and $g(c,d)$; and by Rule **UNEQUAL FUNCTORS**:

$$\begin{pmatrix} \{\, f(a, X_1)/X, g(X_1, b)/Y \,\} \\ \{\, X_2/X, X_2/Y \,\} \end{pmatrix} \vdash \begin{pmatrix} f(Y_1, Y_1) \\ g(c,d) \end{pmatrix} Z \begin{pmatrix} \{\, \ldots, f(Y_1, Y_1)/Z \,\} \\ \{\, \ldots, g(c,d)/Z \,\} \end{pmatrix}$$

and so $u_3 = Z$;

- therefore, this yields:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix} \vdash \begin{pmatrix} h(f(a, X_1), g(X_1, b), f(Y_1, Y_1)) \\ h(X_2, X_2, g(c,d)) \end{pmatrix} h(X, Y, Z) \begin{pmatrix} \{\, \ldots, f(Y_1, Y_1)/Z \,\} \\ \{\, \ldots, g(c,d)/Z \,\} \end{pmatrix}$$

that is, $t = h(X, Y, Z)$ with:

$$\sigma_1 = \{\, f(a, X_1)/X, g(X_1, b)/Y, f(Y_1, Y_1)/Z \,\}$$

s.t. $t\sigma_1 = h(f(a, X_1), g(X_1, b), f(Y_1, Y_1))$; and,

$$\sigma_2 = \{\, X_2/X, X_2/Y, g(c,d)/Z \,\}$$

s.t. $t\sigma_2 = h(X_2, X_2, g(c,d))$.

3.7 Fuzzy Lattice Operations on $\mathcal{FOT}$s

For the formal Fuzzy Algebra notation and terminology that we use in the remainder of this work, see Chapter 2, Section 2.2.

3.7.1 Fuzzy $\mathcal{FOT}$ unification

Sessa's weak unification

A fuzzy unification operation on $\mathcal{FOT}$s, dubbed "*weak unification*," was proposed by Maria Sessa in [157] which consists in normalizing fuzzy equations between conventional $\mathcal{FOT}$s modulo a similarity relation $\sim$ over functor symbols [74]. This similarity relation is then homomorphically extended to one over all $\mathcal{FOT}$s.

Example 3.6 Functor similarity matrix — Given a similarity (*i.e.*, a fuzzy equivalence) relation $\sim$ on a finite signature $\Sigma = \cup_n \Sigma_n$, as explained in [74], this information is represented as a matrix in $\Sigma \times \Sigma \to [0.0, 1.0]$. For example, if the signature Σ is the union of $\Sigma_0 = \{\, a, b, c, d \,\}$, $\Sigma_2 = \{\, f, g \,\}$, $\Sigma_3 = \{\, h \,\}$, and $\Sigma_n = \emptyset$ otherwise ($n = 1$ or $n \geq 4$), and with a similarity that is the reflexive, symmetric,

and transitive closure of the pairs $a \sim_{0.7} b$, $c \sim_{0.6} d$, and $f \sim_{0.9} g$. This corresponds to the similarity matrix whose rows and columns are indexed by elements of Σ:

$\sim \stackrel{\text{def}}{=}$

	a	b	c	d	f	g	h
a	1	0.7	0	0	0	0	0
b	0.7	1	0	0	0	0	0
c	0	0	1	0.6	0	0	0
d	0	0	0.6	1	0	0	0
f	0	0	0	0	1	0.9	0
g	0	0	0	0	0.9	1	0
h	0	0	0	0	0	0	1

Following Maria Sessa's formal setting [157], we assume given such a similarity relation between functors of equal arity (*i.e.*, which admit the same number of arguments). Upon this basis, this similarity can be extended homomorphically from functors to $\mathcal{FOT}$s as follows. Let $\sim$ be a similarity on functors of equal arity in a signature Σ.

DEFINITION 3.5 (SESSA'S $\mathcal{FOT}$ SIMILARITY) *The fuzzy relation $\sim^{\mathcal{T}}$ on $\mathcal{T}_{\Sigma,\mathcal{V}}$ is defined inductively as follows:*

1. $\forall X \in \mathcal{V}, \quad X \sim_1^{\mathcal{T}} X$;
2. $\forall X \in \mathcal{V}, \forall t \in \mathcal{T}$ *such that* $X \neq t$, $X \sim_0^{\mathcal{T}} t$ *and* $t \sim_0^{\mathcal{T}} X$;
3. *if* $f \in \Sigma_n$ *and* $g \in \Sigma_n$ *with* $f \sim_\alpha g$, *and if* $s_i \in \mathcal{T}$ *and* $t_i \in \mathcal{T}$ *are such that* $s_i \sim_{\alpha_i}^{\mathcal{T}} t_i$ *for* $i = 1, \ldots, n$, *then:*

$$f(s_1, \ldots, s_n) \sim^{\mathcal{T}}_{\alpha \wedge \bigwedge_{i=1}^n \alpha_i} g(t_1, \ldots, t_n). \tag{3.11}$$

THEOREM 3.2 *The relation $\sim^{\mathcal{T}}$ defined by Definition 3.5 is a similarity relation on the set of $\mathcal{FOT}$s $\mathcal{T}_{\Sigma,\mathcal{V}}$.*

> **PROOF** See proof of more general Theorem 3.3 below, as this is a particular case of that theorem where every similar pairs of functors have same arity and every argument position mapping is the identity. □

Since from the above definition of similarity $\sim^{\mathcal{T}}$ extends homomorphically a similarity $\sim$ on the functors to all $\mathcal{FOT}$s in $\mathcal{T}$, we shall also assimilate $\sim^{\mathcal{T}}$ to $\sim$. This allows to define formally fuzzy subsumption among $\mathcal{FOT}$s as the fuzzy relation $\preceq$ on $\mathcal{T}$ that can be verified to be a preorder (modulo variable renaming) as a corollary of Theorem 3.2.

DEFINITION 3.6 (FUZZY $\mathcal{FOT}$ SUBSUMPTION) *For all terms $t_1 \in \mathcal{T}$ and $t_2 \in \mathcal{T}$, t_1 is said to be subsumed by t_2 for some α in $[0.0, 1.0]$ (and this is written $t_1 \preceq_\alpha t_2$) if and only if there exists a substitution $\sigma \in$ **SUBST**$_{\mathcal{T}}$ such that $t_1 \sim_\alpha t_2\sigma$.*

Note that, for the identity similarity on the signature and $\alpha = 1$, this reduces to the classical definition of term subsumption, as expected.

In Definition 3.6, the more specific term t_1 is then called a fuzzy instance of term t_2 realized with subsititution σ at approximation degree α. It comes also that the "more general" relation on $\mathcal{FOT}$ substitutions extends to a "fuzzy more general" fuzzy relation on substitutions, which can also readily be verified to be a fuzzy preorder on $\mathbf{SUBST}_{\mathcal{T}}$ as a corollary of Theorem 3.2. It is formally equivalent to the relation defined in [157].[13]

DEFINITION 3.7 (FUZZY "MORE GENERAL" ORDERING ON $\mathcal{FOT}$ SUBSTITUTIONS) *If σ_1 and σ_2 are two substitutions in $\mathbf{SUBST}_{\mathcal{T}}$ and α in $[0.0, 1.0]$, we say that σ_1 is less general than σ_2 at approximation degree α (and this is written $\sigma_1 \preceq_\alpha \sigma_2$), if and only if for any term $t \in \mathcal{T}$, it is true that $t\sigma_1 \preceq_\alpha t\sigma_2$ as terms.*

Also as expected, note that for the identity similarity on the signature and $\alpha = 1$, this reduces to the classical "more general than" ordering on substitutions.

The following fuzzy relation defined on $\mathbf{SUBST}_{\mathcal{T}}$ can also be verified to be a similarity as a corollary of Theorem 3.2.[14]

DEFINITION 3.8 ($\mathcal{FOT}$ SUBSTITUTION SIMILARITY) *Given an approximation degree α in $[0.0, 1.0]$, two substitutions σ and θ in $\mathbf{SUBST}_{\mathcal{T}}$ are said to be α-similar (written $\sigma \sim_\alpha \theta$) iff $t\sigma \sim_\alpha t\theta$ for all $\mathcal{FOT}$ t in $\mathcal{T}$.*

Therefore, referring to Definition 3.6 of fuzzy $\mathcal{FOT}$ subsumption, it comes as a fact that:

LEMMA 3.3 *For any two substitutions σ and θ in $\mathbf{SUBST}_{\mathcal{T}}$ and approximation degree α in $[0.0, 1.0]$, $\sigma \preceq_\alpha \theta$ iff $\sigma \sim_\alpha \theta\delta$ for some substitution δ.*

> **PROOF** Stating that $\sigma \preceq_\alpha \theta$, by Definition 3.7, is equivalent to stating that $t\sigma \preceq_\alpha t\theta$, for any $t \in \mathcal{T}$. By Definition 3.6, this is equivalent to stating that for all term t, $t\sigma \sim_\alpha t\theta\delta$, for some substitution δ; namely, again by Definition 3.7, that $\sigma \sim_\alpha \theta\delta$. □

The following two facts regarding the fuzzy term subsumption relation on terms and the fuzzy "more general" relation on substitutions will be useful later in proof arguments.

LEMMA 3.4 *For any two approximation degrees α and β in $[0.0, 1.0]$, for any terms t_1, t_2, and t_3 in $\mathcal{T}$, if $t_1 \preceq_\alpha t_2$ and $t_2 \preceq_\beta t_3$, then $t_1 \preceq_{\alpha\wedge\beta} t_3$.*

> **PROOF** Let $t_1 \preceq_\alpha t_2$ and $t_2 \preceq_\beta t_3$; this is, by definition, equivalent to $t_1 \sim_\alpha t_2\sigma$, for some $\sigma \in \mathbf{SUBST}_{\mathcal{T}}$, and $t_2 \sim_\beta t_3\theta$, for some $\theta \in \mathbf{SUBST}_{\mathcal{T}}$. However, for any set S, any pair $\langle x, y\rangle$ in $S \times S$, and any similarity $\sim\colon S \times S \to [0.0, 1.0]$, if $x \sim_\alpha y$ for some α in$[0.0, 1.0]$, then $x \sim_\beta y$ for all $\beta \in [0, \alpha]$.[15] This, the fact that $\alpha \wedge \beta \leq \alpha$ and $\alpha \wedge \beta \leq \beta$, together with our assumption, entail then that $t_1 \sim_{\alpha\wedge\beta} t_2\sigma$ and $t_2 \sim_{\alpha\wedge\beta} t_3\sigma$; which, by transitivity of $\sim_{\alpha\wedge\beta}$, implies that $t_1 \sim_{\alpha\wedge\beta} t_3\sigma$; that is, $t_1 \preceq_{\alpha\wedge\beta} t_3$. □

[13] *Op. cit.*, Page 410, Definition 6.2

[14] An equivalent definition is given in [101].

[15] See 2.2.2.

COROLLARY 3.3 *For any two approximation degrees α and β in $[0.0, 1.0]$, for any substitutions σ_1, σ_2, and σ_3 in* **SUBST**$_{\mathcal{T}}$*, if $\sigma_1 \preceq_\alpha \sigma_2$ and $\sigma_2 \preceq_\beta \sigma_3$, then $\sigma_1 \preceq_{\alpha\wedge\beta} \sigma_3$.*

PROOF Let $\sigma_1 \preceq_\alpha \sigma_2$ and $\sigma_2 \preceq_\beta \sigma_3$; this is, by definition, equivalent to stating that for any term $t \in \mathcal{T}$, $t\sigma_1 \preceq_\alpha t\sigma_2$ and $t\sigma_2 \preceq_\beta t\sigma_3$. By Lemma 3.4, it follows that for any term $t \in \mathcal{T}$, $t\sigma_1 \preceq_{\alpha\wedge\beta} t\sigma_3$; that is, $\sigma_1 \preceq_{\alpha\wedge\beta} \sigma_3$. □

Using the definition of similarity between terms in $\mathcal{T}$ extending one on functors of equal arity, Sessa proposes to extend the $\mathcal{FOT}$ unification problem to the following fuzzy unification problem: given two $\mathcal{FOT}$s t_1 and t_2 in $\mathcal{T}$, find the most general substitution $\sigma \in$ **SUBST**$_{\mathcal{T}}$ and maximum approximation degree α in $[0.0, 1.0]$ such that $t_1\sigma \sim_\alpha t_2\sigma$. Figure 3.6 expresses fuzzy unification as a commutative diagram constraint.

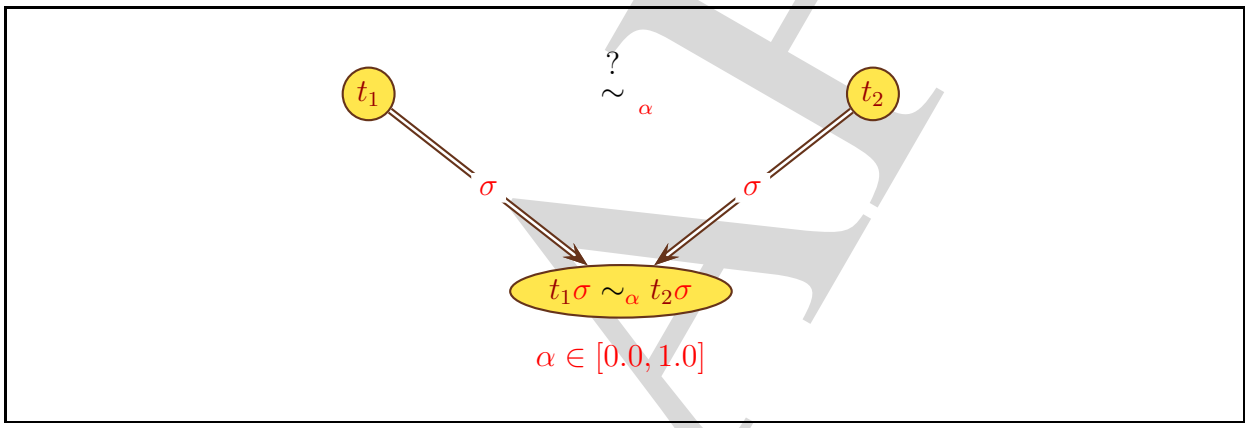


Figure 3.6: Fuzzy unification as a constraint

In Figure 3.7, we provide a set of declarative rewrite rules for fuzzy unfication equivalent to Sessa's case-based "weak unification algorithm" [157]. To simplify the presentation of these rules while remaining faithful to Sessa's weak unification algorithm, it is assumed for now that functor symbols f/m and g/n of different arities $m \neq n$ are never similar. This follows Sessa's assumption for weak unification, which fails on term structures of different arities. (See Case (2) of the weak unification algorithm given in [157], Page 413.) Later, we will relax this and allow functors of different arities to be similar.

The rules of Figure 3.7 transform E_α, a finite conjunctive set E of equations among $\mathcal{FOT}$s along with an associated approximation degree α in $[0.0, 1.0]$, into $E'_{\alpha'}$, another set of equations E' at approximation degree α' in $[0, \alpha]$. Given to solve a fuzzy unification equation $s \doteq t$ between two $\mathcal{FOT}$s s and t, we start by forming the set $\{\, s \doteq t \,\}_1$ (*i.e.*, a singleton equation set at approximation degree 1), then transform it using any applicable rules in Figure 3.7 until either the approximation degree of the transformed set of equations is 0 (in which case there is no solution to the original equation, not even a fuzzy one), or the final resulting set E_α is a solution at approximation degree α in the form of a variable substitution $\sigma \stackrel{\text{def}}{=} \{\, t/X \mid X \doteq t \in E \,\}$ such that $s\sigma \sim_\alpha t\sigma$.

In [157],[16] a transformation rule of a set of equation at approximation degree is considered to

[16] *Op. cit.*, Page 410.

WEAK TERM DECOMPOSITION

$[f \sim_\beta g;\ n \geq 0]$

$$\frac{(E \cup \{\, f(s_1, \ldots, s_n) \doteq g(t_1, \ldots, t_n) \,\})_\alpha}{(E \cup \{\, s_1 \doteq t_1, \ldots, s_n \doteq t_n \,\})_{\alpha \wedge \beta}}$$

VARIABLE ERASURE

$$\frac{(E \cup \{\, X \doteq X \,\})_\alpha}{E_\alpha}$$

VARIABLE ELIMINATION

$[X \notin \mathbf{var}(t);\ X \text{ occurs in } E]$

$$\frac{(E \cup \{\, X \doteq t \,\})_\alpha}{(E[t/X] \cup \{\, X \doteq t \,\})_\alpha}$$

EQUATION ORIENTATION

$[t \notin \mathcal{V}]$

$$\frac{(E \cup \{\, t \doteq X \,\})_\alpha}{(E \cup \{\, X \doteq t \,\})_\alpha}$$

Figure 3.7: Normalization rules corresponding to Maria Sessa's "weak unification"

be correct when all the solutions of the posterior set are also solutions of the anterior set but with a possibly lesser similarity degree, which is also our Definition 3.10.[17]

Example 3.7 $\mathcal{FOT}$ **fuzzy unification** — Taking the functor signature of Example 3.6, let us consider the fuzzy equation set:

$$\{\, h(f(a, X_1), g(X_1, b), f(Y_1, Y_1)) \doteq h(X_2, X_2, g(c, d)) \,\}_1 \tag{3.12}$$

and let us apply the rules of Figure 3.7:

- Rule **WEAK TERM DECOMPOSITION** with $\alpha = 1$ and $\beta = 1$:

  $$\{\, f(a, X_1) \doteq X_2, g(X_1, b) \doteq X_2, f(Y_1, Y_1) \doteq g(c, d) \,\}_1 ;$$

- Rule **EQUATION ORIENTATION** to $f(a, X_1) \doteq X_2$ with $\alpha = 1$:

  $$\{\, X_2 \doteq f(a, X_1), g(X_1, b) \doteq X_2, f(Y_1, Y_1) \doteq g(c, d) \,\}_1 ;$$

- Rule **VARIABLE ELIMINATION** to $X_2 \doteq f(a, X_1)$ with $\alpha = 1$:

  $$\{\, X_2 \doteq f(a, X_1), g(X_1, b) \doteq f(a, X_1), f(Y_1, Y_1) \doteq g(c, d) \,\}_1 ;$$

- Rule **WEAK TERM DECOMPOSITION** to $g(X_1, b) \doteq f(a, X_1)$ with $\alpha = 1$ and $\beta = .9$:

  $$\{\, X_2 \doteq f(a, X_1), X_1 \doteq a, b \doteq X_1, f(Y_1, Y_1) \doteq g(c, d) \,\}_{.9} ;$$

[17] Note that in [157], no explicit proof for of formal correctness of "weak unification algorithm" is given: it is just mentioned that "it can be proven following the same line of the proof" for crisp unification in classible Logic Programming in [34].

- Rule **VARIABLE ELIMINATION** to $X_1 \doteq a$ with $\alpha = .9$:

$$\{ X_2 \doteq f(a,a), X_1 \doteq a, b \doteq a, f(Y_1,Y_1) \doteq g(c,d) \}_{.9};$$

- Rule **WEAK TERM DECOMPOSITION** to $b \doteq a$ with $\alpha = .9$ and $\beta = .7$:

$$\{ X_2 \doteq f(a,a), X_1 \doteq a, f(Y_1,Y_1) \doteq g(c,d) \}_{.7};$$

- Rule **WEAK TERM DECOMPOSITION** to $f(Y_1,Y_1) \doteq g(c,d)$ with $\alpha = .7$ and $\beta = .9$:

$$\{ X_2 \doteq f(a,a), X_1 \doteq a, Y_1 \doteq c, Y_1 \doteq d \}_{.7};$$

- Rule **VARIABLE ELIMINATION** to $Y_1 \doteq c$ with $\alpha = .7$:

$$\{ X_2 \doteq f(a,a), X_1 \doteq a, Y_1 \doteq c, c \doteq d \}_{.7};$$

- Rule **WEAK TERM DECOMPOSITION** to $c \doteq d$ with $\alpha = .7$ and $\beta = .6$:

$$\{ X_2 \doteq f(a,a), X_1 \doteq a, Y_1 \doteq c \}_{.6}.$$

This last equation set is in normal form with similarity degree .6 and defines the substitution σ given by:

$$\sigma = \{ a/X_1, c/Y_1, f(a,a)/X_2 \} \tag{3.13}$$

so that:

$$t_1\sigma = h(f(a,a), g(a,b), f(c,c)) \sim_{.6} h(f(a,a), f(a,a), g(c,d)) = t_2\sigma. \tag{3.14}$$

Generic fuzzy $\mathcal{FOT}$ unification

From our perspective, a fuzzy unification operation ought to be able to fuzzify *full* $\mathcal{FOT}$ unification: whether (1) functor symbol mistmatch, and/or (2) arity mismatch, and/or (3) in which order subterms correspond. Sessa's fuzzification of unification as weak unification misses on the last two items. This is unfortunate as this can turn out to be quite useful. In real life, there is indeed no such garantee that argument positions of different functors match similar information in data and knowledge bases, hence the need for alignment [114].

Still, Sessa's approach has several qualities:

- *It is simple* — specified as a straightforward extension of crisp unification: only one rule (Rule "**FUZZY TERM DECOMPOSITION**") may alter the fuzziness of an equation set by tolerating similar functors.

- *It is conservative* — neither $\mathcal{FOT}$s nor $\mathcal{FOT}$ substitutions *per se* need be fuzzified; so conventional crisp representations and operations can be used; if restricted to only 0 or 1 similarity degrees, it is equivalent to crisp $\mathcal{FOT}$ unification.

We now give an extension of Sessa's weak unification which can tolerate such similarity among functors of different arities. We are given a similarity relation $\approx : \Sigma \times \Sigma \to [0.0, 1.0]$ on a ranked signature $\Sigma \stackrel{\text{def}}{=} \biguplus_{n \geq 0} \Sigma_n$. Unlike M. Sessa's equal-arity condition, we now allow similar symbols with distinct arities, or equal arities but different argument orders.

Example 3.8 Similar functors with different arities — Consider `person`/3, a functor of arity 3, and `individual`/4, a functor of arity 4 with:

- `person`/3 $\approx_{.9}$ `individual`/4; and,
- one-to-one position mapping $p : \{1, 2, 3\} \to \{1, 2, 3, 4\}$:

  from `person`/3 to `individual`/4 with $p : \{1 \to 1, 2 \to 3, 3 \to 4\}$

  so that:

  $$\texttt{person}(\textit{Name}, \textit{SSN}, \textit{Address}) \approx^{p}_{.9} \texttt{individual}(\textit{Name}, \textit{DoB}, \textit{SSN}, \textit{Address})$$

where we write $f \approx^{p}_{\alpha} g$ to denote a pair in the similarity relation $\approx$ consisting of a functor f and a functor g, with similarity degree α and f-to-g argument-position mapping p; in our example, this is rendered as: `person` $\approx^{\{1\to1,2\to3,3\to4\}}_{.9}$ `individual`.

With this kind of specification, we can tolerate not only fuzzy mismatching of terms with distinct functors `person` and `individual` up to a realigning correspondence of argument positions from `person` to `individual` specified as p, all with a similarity degree of .9.

We formalize this by requiring that the fuzzy equivalence relation $\approx$ on Σ be such that:

- for each pair of functors $\langle f, g \rangle \in \Sigma_m \times \Sigma_n$ where $0 \leq m \leq n$ and $f \approx g$, and any approximation degree α, there exists a one-to-one (*i.e.*, injective) mapping $\mu^{\alpha}_{fg} : \{1, \ldots, m\} \to \{1, \ldots, n\}$ associating each of the m argument positions of f with a unique position among the n arguments of g, which we shall express as $f \approx^{\mu^{\alpha}_{fg}} g$;
- argument-position alignment mappings between similar functors must be *consistent* at any approximation level; namely, they must satisfy the following four conditions:
  - *approximation consistency*: for any functors $f \in \Sigma$ and $g \in \Sigma$, and any approximation degrees $\alpha \in [0.0, 1.0]$ and $\beta \in [0.0, 1.0]$:

    $$\alpha \leq \beta \Rightarrow \mu^{\alpha}_{fg} \subseteq \mu^{\beta}_{fg} \quad \text{(as sets of pairs)}; \tag{3.15}$$

  - *reflexive consistency*: for any functor f/n and any degree $\alpha \in [0.0, 1.0]$:

    $$\mu^{\alpha}_{ff} = \mathbb{1}_{\{1,\ldots,n\}}; \tag{3.16}$$

  - *symmetric consistency*: for any two equal-arity functors f/n and g/n and any degree $\alpha \in [0.0, 1.0]$:

    $$\mu^{\alpha}_{fg} \circ \mu^{\alpha}_{gf} = \mathbb{1}_{\{1,\ldots,n\}}; \tag{3.17}$$

– *transitive consistency*: for any three functors f/m, g/n, h/ℓ s.t. $0 \leq m \leq n \leq \ell$ and any degree $\alpha \in [0.0, 1.0]$:

$$\mu^{\alpha}_{fh} \;=\; \mu^{\alpha}_{gh} \circ \mu^{\alpha}_{fg}. \tag{3.18}$$

Figure 3.8 illustrates Condition (3.16), Figure 3.9 illustrates Condition (3.17), and Figure 3.10 illustrates Condition (3.18). Note that Condition (3.18) applies when $0 \leq m \leq n \leq \ell$; so the

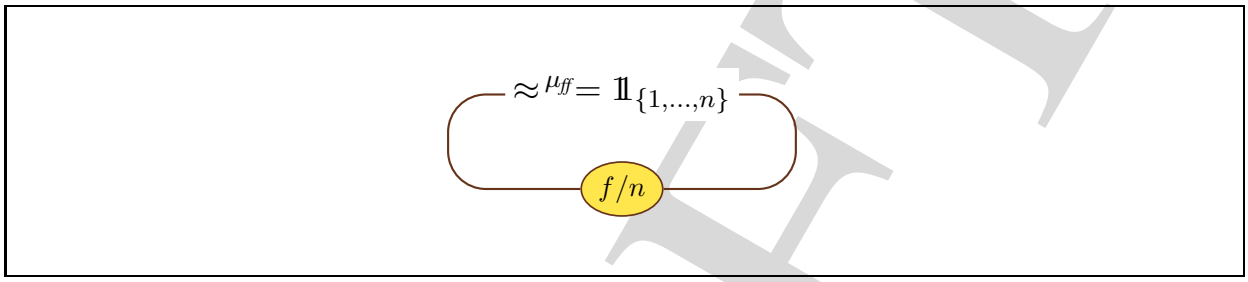


Figure 3.8: Identity consistency for $\mathcal{FOT}$ argument mapping

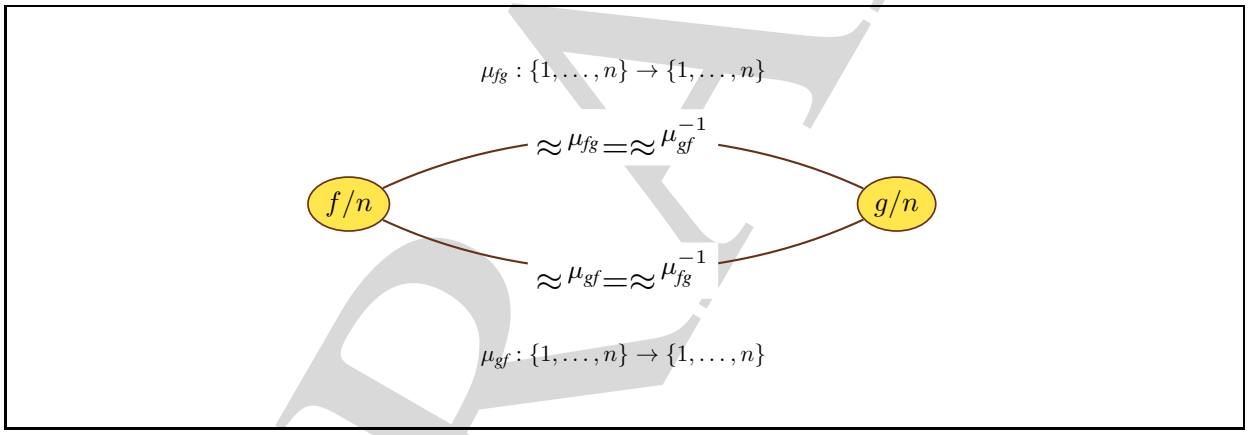


Figure 3.9: Invertibility consistency for equal-arity $\mathcal{FOT}$ argument mapping

one-to-one argument-position mappings always go from a smaller set to a larger set. There is no loss of generality with this assumption as this will be taken into account in the definition of non-aligned $\mathcal{FOT}$ similarity,[18] and in the normalization rules.[19] This amounts to systematically taking a $\mathcal{FOT}$ with functor of least arity as similarity class representative. Finally, note also that such a class representative is not unique because for similar functors of equal arity, it can be either terms due to Condition (3.17). Indeed, then the set of positions are equal and there are two injections from this set to itself in each direction which are mutually inverse bijections; *i.e.*, inverse permutations in the order of arguments realigning one's with the other's in either direction. The similarity degrees in both directions are always equal due to symmetry of similarity.

[18] *Cf.*, Definition 3.9 below.
[19] *Cf.*, Figure 3.11 below, Rule **FUZZY EQUATION ORIENTATION**.

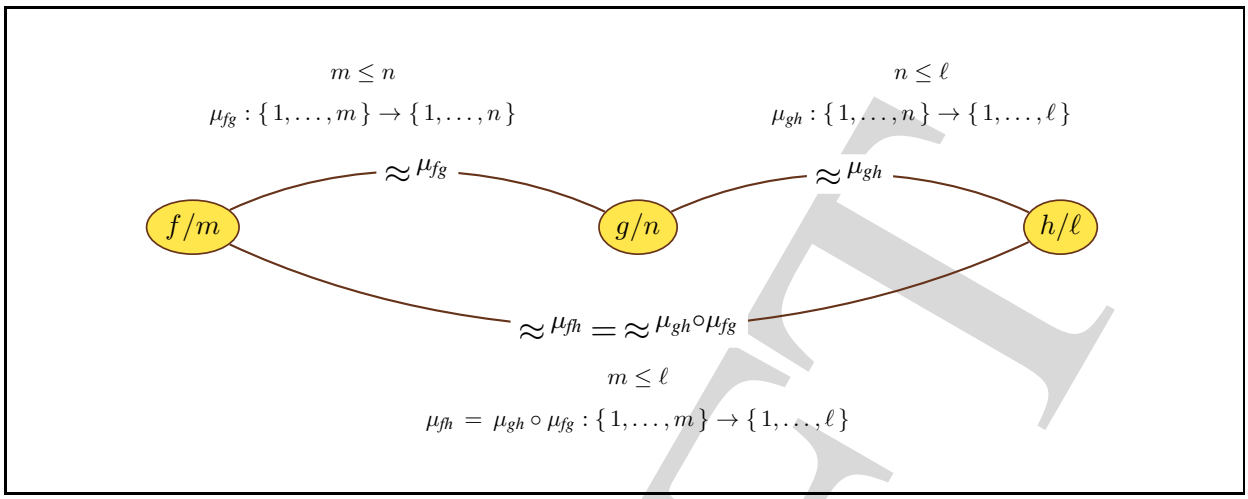


Figure 3.10: Compositional consistency for non-aligned $\mathcal{FOT}$ argument mapping

Fuzzy unification with similar functors and arity mismatch

As in the case of similarity restricted to functors of equal arities only, the similarity with argument position alignment mapping on functors can be extended homomorphically to a similarity on $\mathcal{FOT}$s. Let $\approx$ be a similarity on functors of any arity in a signature Σ. To lighten notation, rather than writing systematically $f \approx^{\mu_{fg}} g$ for two functors f and g such that **arity**$(f) \leq$ **arity**(g), we shall sometimes simply write $f \approx^{p}_{\alpha} g$, with p standing for the injective argument realignment mapping μ_{fg}.

DEFINITION 3.9 *The fuzzy relation $\approx^{\mathcal{T}}$ on $\mathcal{T}_{\Sigma,\mathcal{V}}$ is defined inductively as:*

1. $\forall X \in \mathcal{V}, \;\; X \approx^{\mathcal{T}}_{1} X$;
2. $\forall X \in \mathcal{V}, \forall t \in \mathcal{T}$ *such that* $X \neq t$, $X \sim^{\mathcal{T}}_{0} t$ *and* $t \sim^{\mathcal{T}}_{0} X$;
3. *if* $s = f(s_1, \ldots, s_m)$ *and* $t = g(t_1, \ldots, t_n)$ *with* $n < m$, *then* $s \approx^{\mathcal{T}} t = t \approx^{\mathcal{T}} s$;
4. *if* $f \in \Sigma_m$ *and* $g \in \Sigma_n$ *with* $m \leq n$ *and* $f \approx^{p}_{\alpha} g$, *and if* $s_i \in \mathcal{T}$, $i = 1, \ldots, m$, *and* $t_j \in \mathcal{T}$, $j = 1, \ldots, n$, *are such that* $s_i \approx^{\mathcal{T}}_{\alpha_i} t_{p(i)}$ *for all* $i \in \{1, \ldots, m\}$, *then:*

$$f(s_1, \ldots, s_m) \approx^{\mathcal{T}}_{\alpha \wedge \bigwedge_{i=1}^{m} \alpha_i} g(t_1, \ldots, t_n). \tag{3.19}$$

THEOREM 3.3 (NON-ALIGNED $\mathcal{FOT}$ SIMILARITY) *The fuzzy relation $\approx^{\mathcal{T}}$ on the set $\mathcal{T}$ of $\mathcal{FOT}$s specified in Definition 3.9 is a similarity.*

PROOF We must establish that $\approx^{\mathcal{T}}$ is reflexive, symmetric, and transitive.

Reflexivity: we must show that $t \approx^{\mathcal{T}}_{1} t$, for all $t \in \mathcal{T}$. We proceed by induction on the depth of the term. *Base case:* either $t = X \in \mathcal{V}$, in which case, by the first condition of Definition 3.9, $X \approx^{\mathcal{T}}_{1} X$; or, $t = c \in \Sigma_0$, in which case the fourth condition of Definition 3.9

and the fact that $c \approx_1 c$ implies that $c \approx_1^{\mathcal{T}} c$, for all $c \in \Sigma_0$. *Inductive case:* let us assume that $\approx^{\mathcal{T}}$ is reflexive for all terms of depth less than or equal to d, and consider the term $t = f(t_1, \ldots, t_n)$ of depth $d+1$; then, the fourth condition of Definition 3.9 implies also that $t \approx^{\mathcal{T}} t$ since, by Condition (3.16) and the fact that $\approx$ is a similarity, $f \approx_1^{\mathbb{1}_{\{1,\ldots,n\}}} f$ for all $f \in \Sigma_n$, for any arity $n > 0$.

Symmetry: we must show that $s \approx^{\mathcal{T}} t = t \approx^{\mathcal{T}} s$ for all s and t in $\mathcal{T}$. When either of the terms is a variable, this is so by the two first cases of Definition 3.9. When $s = f(s_1, \ldots, s_m)$ and $t = g(t_1, \ldots, t_n)$, it is is always the case that $\approx^{\mathcal{T}}$ is symmetric on such pairs since the third condition of Definition 3.9, states precisely that in this case $\approx^{\mathcal{T}}$ is symmetric.

Transitivity: we must show that $(s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u) \leq s \approx^{\mathcal{T}} u$ for all terms s, t, u. There are eight possibilities:

(1) $s \in \mathcal{V}$ and $t \in \mathcal{V}$ and $u \in \mathcal{V}$;
(2) $s \in \mathcal{V}$ and $t \in \mathcal{V}$ and $u \notin \mathcal{V}$;
(3) $s \in \mathcal{V}$ and $t \notin \mathcal{V}$ and $u \in \mathcal{V}$;
(4) $s \in \mathcal{V}$ and $t \notin \mathcal{V}$ and $u \notin \mathcal{V}$;
(5) $s \notin \mathcal{V}$ and $t \in \mathcal{V}$ and $u \in \mathcal{V}$;
(6) $s \notin \mathcal{V}$ and $t \in \mathcal{V}$ and $u \notin \mathcal{V}$;
(7) $s \notin \mathcal{V}$ and $t \notin \mathcal{V}$ and $u \in \mathcal{V}$;
(8) $s \notin \mathcal{V}$ and $t \notin \mathcal{V}$ and $u \notin \mathcal{V}$.

– Case (1): $s \in \mathcal{V}, t \in \mathcal{V}, u \in \mathcal{V}$. In this case, there are five possiblities. Using different variable names to denote different variables, the corresponding similarity degrees for $s \approx^{\mathcal{T}} t$, $t \approx^{\mathcal{T}} u$, and $s \approx^{\mathcal{T}} u$, for each possibility do indeed satisfy the inequality. Namely:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$
X	Y	Z	$0 \wedge 0 \leq 0$
X	Y	Y	$0 \wedge 1 \leq 1$
X	Y	X	$0 \wedge 0 \leq 1$
X	X	Y	$1 \wedge 0 \leq 0$
X	X	X	$1 \wedge 1 \leq 1$

– Case (2): $s \in \mathcal{V}, t \in \mathcal{V}, u \notin \mathcal{V}$. There are two possibilities, each satisfying the inequality:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$
X	X	u	$1 \wedge 0 \leq 0$
X	Y	u	$0 \wedge 0 \leq 0$

– Case (3): $s \in \mathcal{V}, t \notin \mathcal{V}, u \in \mathcal{V}$. There are two possibilities, and each satisfies the inequality:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$
X	t	X	$0 \wedge 0 \leq 1$
X	t	Y	$0 \wedge 0 \leq 0$

– Case (4): $s \in \mathcal{V}, t \notin \mathcal{V}, u \notin \mathcal{V}$. There is only one possibility, for any $\alpha \in [0.0, 1.0]$, which satisfies the inequality:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$
X	t	u	$0 \wedge \alpha \leq 0$

– Case (5): $s \notin \mathcal{V}, t \in \mathcal{V}, u \in \mathcal{V}$. There are two possibilities, and each satisfies the inequality:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$				
s	X	X	0	$\wedge$	1	$\leq$	0
s	X	Y	0	$\wedge$	0	$\leq$	0

– Case (6): $s \notin \mathcal{V}, t \in \mathcal{V}, u \notin \mathcal{V}$. There is only one possibility, for any $\alpha \in [0.0, 1.0]$, which satisfies the inequality:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$				
s	X	u	0	$\wedge$	0	$\leq$	α

– Case (7): $s \notin \mathcal{V}, t \notin \mathcal{V}, u \in \mathcal{V}$. There is only one possibility, for any $\alpha \in [0.0, 1.0]$, which satisfies the inequality:

s	t	u	$s \approx^{\mathcal{T}} t \wedge t \approx^{\mathcal{T}} u \leq s \approx^{\mathcal{T}} u$				
s	t	X	α	$\wedge$	0	$\leq$	0

– Case (8): $s \notin \mathcal{V}, t \notin \mathcal{V}, u \notin \mathcal{V}$. In this case, we must have $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and $u = h(u_1, \ldots, u_\ell)$. We detail this case below.

We must then show that:

$$\big(f(s_1, \ldots, s_m) \approx^{\mathcal{T}} g(t_1, \ldots, t_n)\big) \wedge \big(g(t_1, \ldots, t_n) \approx^{\mathcal{T}} h(u_1, \ldots, u_\ell)\big) \leq f(s_1, \ldots, s_m) \approx^{\mathcal{T}} h(u_1, \ldots, u_\ell).$$

By symmetry of $\approx^{\mathcal{T}}$, all cases are equivalent to when $0 \leq m \leq n \leq \ell$, so we assume that this is so, with $f \approx_{\alpha}^{\mu_{fg}} g$ and $g \approx_{\beta}^{\mu_{gh}} h$. By the fourth condition of Definition 3.9, the above inequality is the same as the following one:

$$\big(f \approx g \wedge \textstyle\bigwedge_{i=1}^{m} s_i \approx^{\mathcal{T}} t_{\mu_{fg}(i)}\big) \wedge bigl(g \approx h \wedge \textstyle\bigwedge_{j=1}^{n} t_j \approx^{\mathcal{T}} u_{\mu_{gh}(j)} bigr) \leq f \approx h \wedge \textstyle\bigwedge_{i=1}^{m} s_i \approx^{\mathcal{T}} u_{\mu_{fh}(i)}.$$

Using commutativity of $\wedge$, let us rearrange the different factors of the conjunction in the lefthand-side of this inequality as:

$$\big(f \approx g \wedge g \approx hbigr\big) \wedge \big(\textstyle\bigwedge_{i=1}^{m} s_i \approx^{\mathcal{T}} t_{\mu_{fg}(i)} \wedge t_{\mu_{fg}(i)} \approx^{\mathcal{T}} u_{\mu_{gh}(\mu_{fg}(i))}\big) \wedge \Delta \leq f \approx h \wedge \textstyle\bigwedge_{i=1}^{m} s_i \approx^{\mathcal{T}} u_{\mu_{fh}(i)}$$

where Δ stands for the remaining conjunction $\bigwedge_{j \in \{1,\ldots,n\}}^{j \notin \mathbf{ran}(\mu_{fg})} t_j \approx^{\mathcal{T}} u_{\mu_{gh}(j)}$. Let us now proceed by induction on the depth of the terms to verify this inequality. For terms of depth 0, it is

satisfied since it reduces to the transitivity inequality of $\approx$ on Σ_0. Let us assume that it holds for terms of depth less than d, and that at least one of the terms s, t, or u, is of depth d. By transitivity of $\approx$ on Σ, we have $(f \approx g) \wedge (g \approx h) \leq f \approx h$. Also, by the inductive hypothesis, the transitivity inequality for $\approx^{\mathcal{T}}$ holds for all similar subterms of depth less than or equal to d. Therefore, this assumption entails that for all $i \in \{1, \ldots, m\}$:

$$s_i \approx^{\mathcal{T}} t_{\mu_{fg}(i)} \wedge t_{\mu_{fg}(i)} \approx^{\mathcal{T}} u_{\mu_{gh}(\mu_{fg}(i))} \leq s_i \approx^{\mathcal{T}} u_{\mu_{gh}(\mu_{fg}(i))};$$

thus, since, by Condition (3.18), it is required that the mappings be consistent and satisfy $\mu_{fh} = \mu_{gh} \circ \mu_{fg}$, and by isotonicity of $\wedge$ *w.r.t.* $\leq$, this is equivalent to:

$$\bigwedge_{i=1}^{m} (s_i \approx^{\mathcal{T}} t_{\mu_{fg}(i)}) \wedge (t_{\mu_{fg}(i)} \approx^{\mathcal{T}} u_{\mu_{fh}(i)}) \leq \bigwedge_{i=1}^{m} s_i \approx^{\mathcal{T}} u_{\mu_{fh}(i)}.$$

In summary, the inequality we seek to establish is of the form $A \wedge B \wedge \Delta \leq A' \wedge B'$, and we have shown that $A \leq A'$ and $B \leq B'$. From this, the inequality follows by isotonicity of $\wedge$ *w.r.t.* $\leq$. □

Since we have just formally defined a new notion of similarity $\approx^{\mathcal{T}}$ on $\mathcal{T}$ extending Sessa's similarity $\sim^{\mathcal{T}}$ to non-aligned functors, all the properties we covered for $\sim^{\mathcal{T}}$ carry over to corresponding extensions for terms with non-aligned functors. Namely, Definitions 3.6–3.8 and Lemmas 3.3–3.4, as well as Corollary 3.3, where the term similarity $\sim^{\mathcal{T}}$ is replaced with any similarity on $\mathcal{T}$ such as $\approx^{\mathcal{T}}$ (or $\approxeq^{\mathcal{T}}$ that we shall define later and prove also to be a similarity on $\mathcal{T}$ extending $\approx^{\mathcal{T}}$). Indeed, it is easy to see that all these notions are valid algebraically when parameterized with any relation on $\mathcal{FOT}$ proven to be a similarity on $\mathcal{T}$.

Weak unification with fuzzy functor/arity mismatch

Starting with the Herbrand-Martelli-Montanari ruleset of Figure 3.3, fuzziness is introduced in Sessa's weak unification by relaxing "**Term Decomposition**" to make it also tolerate possible arity or argument-order mismatch in two structures being unified. It is the only rule that does not preserve the equation set's similarity degree. In the same manner, Rule **Fuzzy Non-Aligned-Argument Term Decomposition** in Figure 3.11 is the only one that may possibly alter (decrease) the equation set's similarity degree. Also, the given functor similarity relation $\approx$ on Σ is adjoined a position mapping from argument positions of a functor f to those of a functor g when $f \approx_\alpha g$ with $f \neq g$, for some α in $(0.0, 1.0]$. This is then taken into account in tolerating a fuzzy mismatch between two term structures $s \stackrel{\text{def}}{=} f(s_1, \ldots, s_m)$ and $t \stackrel{\text{def}}{=} g(t_1, \ldots, t_n)$. This may involve a mismatch between the terms' functor symbols (f and g), their arities (m and n), subterm ordering, or a combination. We first reorient all such equations by flipping sides so that the left-hand side is the one wih lesser or equal arity. In this manner, assuming $f \approx^p_\beta g$ and $0 \leq \alpha, \beta \leq 1$, an equation set of the form: $\{ \ \ldots \ , f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n), \ \ldots \ \}_\alpha$ for $0 \leq m \leq n$ acquires its new similarity degree $\alpha \wedge \beta$ due to functor and arity mismatch when equated. Thus, a fully fuzzified term-decomposition rule should proceed by replacing a structure equation by the conjunction of equations between their respective subterms at corresponding indices given by the

FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION

$$\left[0 \leq m \leq n;\ f \approx^{p}_{\beta} g\right]$$

$$\frac{(E \cup \{\, f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n) \,\})_{\alpha}}{(E \cup \{\, s_1 \doteq t_{p(1)}, \ldots, s_m \doteq t_{p(m)} \,\})_{\alpha \wedge \beta}}$$

FUZZY EQUATION ORIENTATION

$$[0 \leq m < n]$$

$$\frac{(E \cup \{\, g(t_1, \ldots, t_n) \doteq f(s_1, \ldots, s_m) \,\})_{\alpha}}{(E \cup \{\, f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n) \,\})_{\alpha}}$$

Figure 3.11: Fuzzy $\mathcal{FOT}$ unification's non-aligned decomposition and orientation rules

one-to-one argument mapping $p : \{\, 1, \ldots, m \,\} \to \{\, 1, \ldots, n \,\}$, but (possibly) decreasing the original equation set similarity degree by conjoining it with that of the decomposed terms' functor pair; that is, $\{\, \ldots,\ s_1 \doteq t_{p(1)},\ \ldots,\ s_m \doteq t_{p(m)},\ \ldots \,\}_{\alpha \wedge \beta}$. Note that all the subterms in the right-hand side term that are arguments at indices which are not p-images are ignored as they have no counterparts in the left-hand side. These terms are simply dropped as part of the approximation. This generic rule is shown in Figure 3.11 along with another rule needed to make it fully effective: a rule reorienting a term equation into one with a lesser-arity term on the left.

DEFINITION 3.10 (FUZZY UNIFICATION RULE CORRECTNESS) *A fuzzy unification rule that transforms a pair E_{α} consisting of a set of equations E and a prior approximation degree α, into a pair E'_{β} consisting of a set of equations E' and a posterior approximation degree β, is said to be correct iff β is the largest degree such that $\beta \leq \alpha$ and all the solutions of E' are also solutions of E at approximation degree β.*

Note that this notion of correctness, contrary to that of crisp unification, does not require that all solutions of the posterior sets of equations be the same as those of of the prior set. It only states that this is so at a possibly lesser posterior approximation degree.

THEOREM 3.4 *The fuzzy unification rules of Figure 3.7 where Rule "***WEAK TERM DECOMPOSITION***" is replaced by the rules of Figure 3.11 are correct.*

PROOF Rules **VARIABLE ELIMINATION**, **VARIABLE ERASURE**, and **EQUATION ORIENTATION** are those, unchanged, of Maria Sessa's weak unification. Their correctness follows from those of the corresponding Herbrand-Martelli-Montanari rules since all three rules

keep their similarity degree α unchanged under the same side conditions as their crisp versions. As for Rule **FUZZY EQUATION ORIENTATION**, it is also correct as it simply uses the symmetry of equality or similarity denoted by the $\doteq$ relation and it leaves the similarity degree unchanged.

The correctness of Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** follows from the fact that it tolerates equations between two distinct but similar functors, f on the left and g on the right, by "paying a toll" as the most general way this can be true is by reducing the prior equation set's similarity degree α to $\alpha \wedge \beta$. It must do so whenever a prior equation set contains an equation between two terms whose respective head functors f and g are β-similar with f having at most as many arguments as g. It collects m corresponding subterm equations from the two terms' subterms using the specific one-to-one argument mapping p that associates with each position i among f's m a unique specific position $p(i)$ among g's n, $n \geq m$. Orienting all functorial term equations to have the lesser number of arguments on the left guarantees completeness over all such syntactic patterns. By structural induction, assuming that all $s_i \approx_{\alpha\wedge\beta} t_{p(i)}$ for all $i \in \{1, \ldots, m\}$, then whenever $f \approx^p_\beta g$, we must also have $f(s_1, \ldots, s_m) \approx_{\alpha\wedge\beta} g(t_1, \ldots, t_n)$ (by definition, since $\alpha \wedge \beta \leq \alpha$) for whatever arguments of g at indices missed by p and these two terms are in the same similarity class at approximation degree $\alpha \wedge \beta$ for arbitrary arguments in these positions (by definition of $f \approx^p_\beta g$ and since $\alpha \wedge \beta \leq \beta$). The rest of the equations in E that were true at approximation degree α must now be considered true only up to approximation degree $\alpha \wedge \beta$ in order to account for f and g being functors of possibly fuzzier similarity β. Hence, all solutions of the new set of equations are also solutions of the previous one, although only at the possibly lesser approximation degree $\alpha \wedge \beta$. This approximation degree is also the greatest such degree by virtue of the $\wedge$ operation yielding the infimum of its operands.

Finally, when $m = n$ this rule is correct in either direction since a consistent similarity on a signature requires by definition that equal-arity functors f and g have arguments in bijection (inverse permutations of the set $\{1, \ldots, n\}$): $f \approx^p_\alpha g$ and $g \approx^{p^{-1}}_\alpha f$. In this case, the set of solutions of the new equation set is also a solution of the previous one, with equal similarity degree.

As for termination, it follows (like that of the Herbrand-Martelli-Montanari rules) from (1) the finite width and depth of $\mathcal{FOT}$s, and (2) there being no rule that is indefinitely applicable. Regarding (1), term decomposition always replaces a term equation with finitely many shallower term equations, which is a well-known well-founded process guaranteed to terminate (multiset ordering [5]). Regarding (2), Rule **FUZZY EQUATION ORIENTATION** may not be reapplied to the same functors thanks to the side condition $m < n$.

In other words, applying this modified ruleset to $E_1 \stackrel{\text{def}}{=} \{s \doteq t\}_1$, an equation set of similarity degree 1 (in any order as long as a rule applies and its similarity degree is not zero) always terminates. And when the final equation set is a substitution σ at approximation degree α, σ is the most general substitution (up to a variable renaming) that is a solution at approximation degree α (*i.e.*, $s\sigma \approx_\alpha t\sigma$), and α is the greatest approximation degree for which this is true. □

Example 3.9 $\mathcal{FOT}$ **fuzzy unification with similar functors of different arities** — Take a functor signature such that: $\{a, b, c, d\} \subseteq \Sigma_0$, $\{f, g, \ell\} \subseteq \Sigma_2$, $\{h\} \subseteq \Sigma_3$; and let us further assume the functor similarity that is the reflexive symmetric transitive closure of:[20]

$$a \approx_{.7} b, \quad c \approx_{.6} d, \quad f \approx_{.9}^{\{1\to 2,2\to 1\}} g, \quad g \approx_{.9}^{\{1\to 2,2\to 1\}} f, \quad \text{and} \quad \ell \approx_{.8}^{\{1\to 2,2\to 3\}} h.$$

Let us consider the fuzzy equation set $\{t_1 \doteq t_2\}_1$:

$$\{h(X, g(Y, b), f(Y, c)) \doteq \ell(f(a, Z), g(d, c))\}_1 \tag{3.20}$$

and let us apply the rules of Figure 3.7 where rule **WEAK TERM DECOMPOSITION** has been replaced by the rules of Figure 3.11:

- Rule **FUZZY EQUATION ORIENTATION** with $\alpha = 1$ because **arity**(ℓ) < **arity**(h); new set: $\{\ell(f(a, Z), g(d, c)) \doteq h(X, g(Y, b), f(Y, c))\}_1$;
- Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** with $\alpha = 1$ and $\beta = .8$ since $\ell \approx_{.8}^{\{1\to 2,2\to 3\}} h$; new set: $\{f(a, Z) \doteq g(Y, b), g(d, c) \doteq f(Y, c)\}_{.8}$;
- Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** to $f(a, Z) \doteq g(Y, b)$ with $\alpha = .8$ and $\beta = .9$ since $f \approx_{.9}^{\{1\to 2,2\to 1\}} g$; new set: $\{a \doteq b, Z \doteq Y, g(d, c) \doteq f(Y, c)\}_{.8}$;
- Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** to $a \doteq b$ with $\alpha = .8$ and $\beta = .7$ since $a \approx_{.7} b$; new set: $\{Z \doteq Y, g(d, c) \doteq f(Y, c)\}_{.7}$;
- Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** to $g(d, c) \doteq f\ (Y, c)$ with $\alpha = .7$ and $\beta = .9$ since $f \approx_{.9}^{\{1\to 2,2\to 1\}} g$; new set: $\{Z \doteq Y, d \doteq c, c \doteq Y\}_{.7}$;
- Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** to $d \doteq c$ with $\alpha = .7$ and $\beta = .6$ since $d \approx_{.6} c$; new set: $\{Z \doteq Y, c \doteq Y\}_{.6}$;
- Rule **EQUATION ORIENTATION** to $c \doteq Y$ with $\alpha = .6$; new set:: $\{Z \doteq Y, Y \doteq c\}_{.6}$.
- Rule **VARIABLE ELIMINATION** to $Y \doteq c$ with $\alpha = .6$; new set: $\{Z \doteq c, Y \doteq c\}_{.6}$.

This last equation set at approximation degree .6 is in normal form and defines the substitution $\sigma = \{c/Z, c/Y\}$ so that: $t_1\sigma = h(X, g(Y, b), f(Y, c))\sigma \approx_{.6} \ell(f(a, Z), g(d, c))\sigma = t_2\sigma$; that is: $t_1\sigma = h(X, g(c, b), f(c, c)) \approx_{.6} \ell(f(a, c), g(d, c)) = t_2\sigma$.

Example 3.10 Example 3.9 with more expressive symbols — Example 3.9 uses abstract generic symbols such as f, g, a, b, ..., which have the advantage, being one-letter symbols, to make each normalization step more compact to write. But one may be more content with a more illustrative choice of identifiers as would be the case of a real-life data base.

So let us give such names to functors of Example 3.9 in the case of a gift-shop Prolog database which describes various configurations for multi-item gift boxes containing such items as flowers, sweets, *etc.*, which can be already joined as pairs or not joined as loose couples. In such a database are consigned facts over objects identified by the following functors (each corresponding to the one indicated from Example 3.9):

[20] The argument mapping is the identity by default.

- $a/0 \stackrel{\text{def}}{=} \mathit{violet}$,
- $b/0 \stackrel{\text{def}}{=} \mathit{lilac}$,
- $c/0 \stackrel{\text{def}}{=} \mathit{chocolate}$,
- $d/0 \stackrel{\text{def}}{=} \mathit{candy}$,
- $f/2 \stackrel{\text{def}}{=} \mathit{pair}$,
- $g/2 \stackrel{\text{def}}{=} \mathit{couple}$,
- $h/3 \stackrel{\text{def}}{=} \mathit{small\text{-}gift\text{-}box}$,

with the closure under reflexivity, symmetry, and transitivity of the following similar pairs:

- $\mathit{violet} \sim_{.7} \mathit{lilac}$,
- $\mathit{chocolate} \sim_{.6} \mathit{candy}$,
- $\mathit{pair} \sim_{.9} \mathit{couple}$.

With these functors and their similarity degrees, Equation (3.12) now reads:

$$
\begin{array}{ll}
 & \mathit{small\text{-}gift\text{-}box}\big(\ \mathit{pair}(\mathit{violet}, X_1) \\
(t_1) & \qquad\qquad\quad ,\ \mathit{couple}(X_1, \mathit{lilac}) \\
 & \qquad\qquad\quad ,\ \mathit{pair}(Y_1, Y_1) \\
 & \qquad\qquad\quad \big) \\
 & \doteq \\
 & \mathit{small\text{-}gift\text{-}box}\big(\ X_2 \\
(t_2) & \qquad\qquad\quad ,\ X_2 \\
 & \qquad\qquad\quad ,\ \mathit{couple}(\mathit{chocolate}, \mathit{candy}) \\
 & \qquad\qquad\quad \big)
\end{array}
$$

Substitution (3.13) obtained after normalization is now defined as follows, using the new functor symbols:

$$\sigma \stackrel{\text{def}}{=} \{X_1 = \mathit{violet}, Y_1 = \mathit{chocolate}, X_2 = \mathit{pair}(\mathit{violet}, \mathit{violet})\}$$

and yields the fuzzy solution:

$$
\begin{array}{ll}
 & \mathit{small\text{-}gift\text{-}box}(\ \mathit{pair}(\mathit{violet},\mathit{violet}) \\
(t_1\sigma) & \qquad\quad ,\ \mathit{couple}(\mathit{violet},\mathit{lilac}) \\
 & \qquad\quad ,\ \mathit{pair}(\mathit{chocolate},\mathit{chocolate}) \\
 & \qquad\quad)
\end{array}
$$

$$\sim_{.6}$$

$$
\begin{array}{ll}
 & \mathit{small\text{-}gift\text{-}box}(\ \mathit{pair}(\mathit{violet},\mathit{violet}) \\
(t_2\sigma) & \qquad\quad ,\ \mathit{pair}(\mathit{violet},\mathit{violet}) \\
 & \qquad\quad ,\ \mathit{couple}(\mathit{chocolate},\mathit{candy}) \\
 & \qquad\quad)
\end{array}
$$

with similarity degree .6 capturing the fuzzy degree to which σ solves the original equation.

Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** is a very general rule for normalizing fuzzy equations over $\mathcal{FOT}$ structures. It has the following convenient properties:

1. it accounts for fuzzy mismatches of similar functors of possibly different arity or order of arguments;
2. when restricted to tolerating only similar equal-arity functors with matching argument positions, it reduces to Sessa's weak unification's **WEAK TERM DECOMPOSITION** rule;
3. when similarity degrees are further restricted to be in $\{0, 1\}$, it is the Herbrand-Martelli-Montanari **TERM DECOMPOSITION** rule;
4. it requires no alteration of the standard notions of $\mathcal{FOT}$s and $\mathcal{FOT}$ substitutions: similarity among $\mathcal{FOT}$s is derived from that of signature symbols;
5. finally, and most importantly, it keeps fuzzy unification in the same complexity class as crisp unification: that of Union-Find [173].[21]

As a result, it is more general than all other extant approaches we know which propose a fuzzy $\mathcal{FOT}$ unification operation. The same will be established for the fuzzification of the dual operation: first a limited "*functor-weak*" $\mathcal{FOT}$ generalization corresponding to the dual operation of Sessa's "weak" unification, then to a more expressive "*functor/arity-weak*" $\mathcal{FOT}$ generalization corresponding to our extension of Sessa's unification to functor/arity weak unification.

3.7.2 Fuzzy $\mathcal{FOT}$ generalization

While there has been relatively intense interest in devising a fuzzy $\mathcal{FOT}$ unification operation, we know of ***no*** work regarding its dual operation, fuzzy $\mathcal{FOT}$ generalization. This comes as no surprise since even in the crisp case only marginal attention has been paid to generalization (*a.k.a.* anti-unification) as compared to unification.

[21] Quasi-linear; *i.e.*, linear with a $\log \ldots \log$ coefficient [2].

The Reynolds-Plotkin characterization of $\mathcal{FOT}$ subsumption as a lattice ordering relies on formalizing this ordering as $\mathcal{FOT}$ instantiation. Namely, $t_1 \preceq t_2$ iff there exists a variable substitution σ such that $t_1 = t_2\sigma$. Then, unification and generalization are respectively the **glb** and **lub** operations for this ordering and are specified in terms of variable substitutions.

It is clear however, as overviewed in the previous section, that there are several ways one can propose to fuzzify $\mathcal{FOT}$ unification. As a consequence of this, for each specific fuzzification of $\mathcal{FOT}$ unification, and therefore of associated specific fuzzy subsumption ordering on $\mathcal{FOT}$s, there should also correspond a dual operation of fuzzy generalization of $\mathcal{FOT}$s.

In what follows, we first elaborate some lattice-theoretic consequences for Maria Sessa's "weak unification" fuzzy operation on $\mathcal{FOT}$s presented in [157]. In particular, we derive its corresponding fuzzy dual lattice operation that we shall dub "weak $\mathcal{FOT}$ generalization." We then extend this lattice to signatures admitting similar functors with differing arity or argument order.

Fuzzy functor-weak generalization

Let t_1 and t_2 be two $\mathcal{FOT}$s in $\mathcal{T}$ to generalize. We shall use the following notation for a fuzzy generalization judgment:

$$\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha} \vdash \begin{pmatrix}t_1\\ t_2\end{pmatrix} t \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix}_{\beta} \tag{3.21}$$

given:

- $\sigma_i \in \textbf{SUBST}_{\mathcal{T}}$ $(i = 1, 2)$: two prior substitutions with prior similarity degree α,
- t_i $(i = 1, 2)$: two prior $\mathcal{FOT}$s,
- t: a posterior $\mathcal{FOT}$,
- $\theta_i \in \textbf{SUBST}_{\mathcal{T}}$ $(i = 1, 2)$: two posterior substitutions with similarity degree β.

DEFINITION 3.11 (FUZZY $\mathcal{FOT}$ GENERALIZATION JUDGMENT VALIDITY) *A fuzzy $\mathcal{FOT}$ generalization judgment such as* (3.21) *is valid whenever, for* $i = 1, 2$*:*

1. $\beta \in (0, \alpha]$*;*
2. $t_i\sigma_i \approx_{\beta} t\theta_i$*;*
3. $\exists\, \delta_i \in \textbf{SUBST}_{\mathcal{T}}$ *s.t.* $t_i \approx_{\alpha} t\delta_i$ *and* $\theta_i \approx_{\beta} \delta_i\sigma_i$ *(i.e.,* $t_i \preceq_{\alpha} t\sigma_i$ *and* $\theta_i \preceq_{\beta} \sigma_i$*).*

Figure 3.12 shows an illustration of a valid fuzzy generalization judgment constraint as a commutative diagram.

DEFINITION 3.12 (FUZZY GENERALIZATION RULE CORRECTNESS) *A fuzzy generalization rule is correct iff, whenever the side condition holds, if all the fuzzy generalization judgments making up its antecedent are valid, then necessarily the fuzzy generalization judgment in its consequent is valid.*

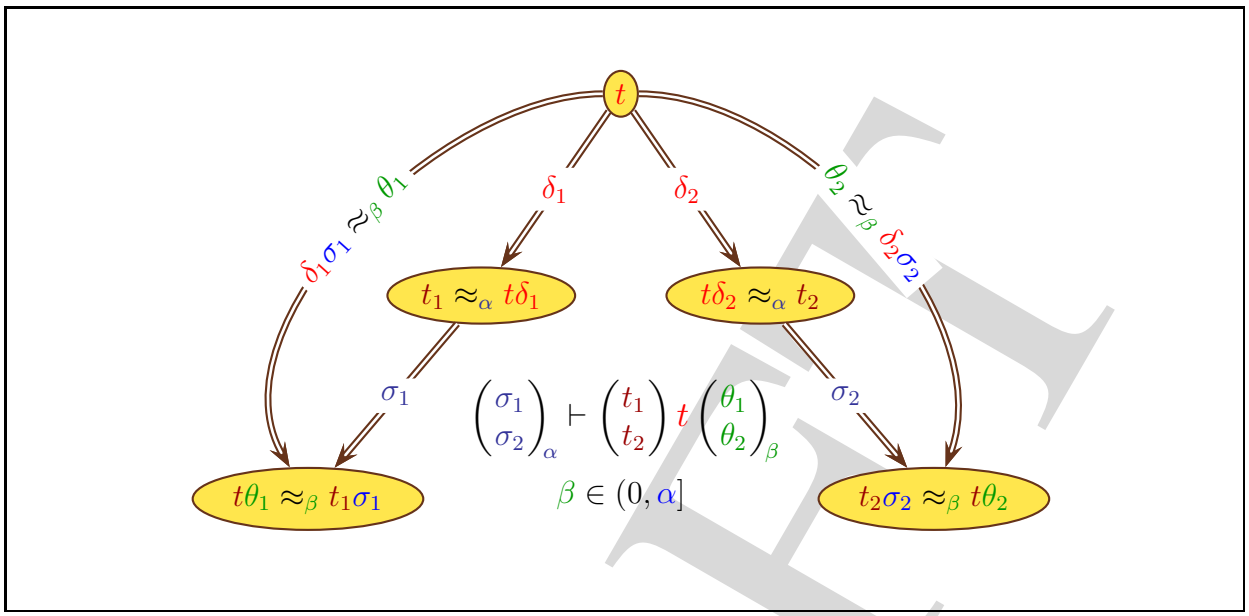


Figure 3.12: Fuzzy generalization judgment validity as a constraint

In Figure 3.13, we give a fuzzy version of the generalization rules of Figure 3.5. As was the case in Sessa's weak unification, we assume as well for now that we are given a similarity relation $\sim\colon \Sigma\times\Sigma \to [0.0, 1.0]$ on the signature $\Sigma = \cup_{n\geq 0}\Sigma_n$ such that for all $m \geq 0$ and $n \geq 0$, $m \neq n$ implies $f \not\sim g$. In other words, functors of different arities may not be similar.

Rule **SIMILAR FUNCTORS** uses a "*fuzzy unapply*" operation ('$\uparrow_\alpha$') on a pair of terms (t_1, t_2) given a pair of substitutions (σ_1, σ_2) and a similarity degree α. It is the result of "*unapplying*" σ_i from t_i, for $i = 1, 2$, into a common variable X, if any such exists such that the terms $X\sigma_i$ are respectively similar to t_i with similarity degrees α_i. It returns a fuzzy pair of terms and a similarity degree in $(0, \alpha]$ defined as:

$$\begin{pmatrix}t_1\\ t_2\end{pmatrix}\uparrow_\alpha\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix} \stackrel{\text{def}}{=} \begin{cases}\begin{pmatrix}X\\ X\end{pmatrix}_{\alpha\wedge\alpha_1\wedge\alpha_2} & \text{if } \exists X\in\mathcal{V},\ t_i \sim_{\alpha_i} X\sigma_i \\ & \text{for some } \alpha_i \in (0.0, 1.0],\ i = 1, 2; \\ \begin{pmatrix}t_1\\ t_2\end{pmatrix}_\alpha & \text{otherwise.}\end{cases} \tag{3.22}$$

The condition in Equation (3.22) is: "$\exists X \in \mathcal{V}$, $t_i \sim_{\alpha_i} X\sigma_i$, for some $\alpha_i \in (0.0, 1.0]$ $(i = 1, 2)$." But could there be two such variables? Namely, is it ever possible that:

$$\exists X \in \mathcal{V}, \exists Y \in \mathcal{V},\ X \neq Y,\ \text{s.t. } t_i \sim_{\alpha_i} X\sigma_i \text{ and } t_i \sim_{\beta_i} Y\sigma_i \tag{3.23}$$

for some $\alpha_i \in (0.0, 1.0]$ and $\beta_i \in (0.0, 1.0]$, $i = 1, 2$? Note that a new variable is introduced in the generalizing pair of substitutions only in Axiom **FUZZY VARIABLE-TERM** and Axiom **DISSIMILAR FUNCTORS**. Then, each axiom binds the new variable in the two substitutions to two terms that are dissimilar at any similarity degree (as required by their side conditions). However, by Lemma 3.4 on Page 39, Condition (3.23) would imply that:

$$t_i \sim_{\alpha_i\wedge\beta_i} X\sigma_i \sim_{\alpha_i\wedge\beta_i} Y\sigma_i$$

FUZZY EQUAL VARIABLES

$$\binom{\sigma_1}{\sigma_2}_\alpha \vdash \binom{X}{X}\, X \,\binom{\sigma_1}{\sigma_2}_\alpha$$

FUZZY VARIABLE-TERM

$[t_1 \in \mathcal{V} \text{ or } t_2 \in \mathcal{V};\ \ t_1 \neq t_2;\ \ X \text{ is new}]$

$$\binom{\sigma_1}{\sigma_2}_\alpha \vdash \binom{t_1}{t_2}\, X \,\binom{\sigma_1\{\, t_1/X\,\}}{\sigma_2\{\, t_2/X\,\}}_\alpha$$

DISSIMILAR FUNCTORS

$[f \not\sim g;\ m \geq 0, n \geq 0;\ X \text{ is new}]$

$$\binom{\sigma_1}{\sigma_2}_\alpha \vdash \binom{f(s_1,\ldots,s_m)}{g(t_1,\ldots,t_n)}\, X \,\binom{\sigma_1\{\, f(s_1,\ldots,s_m)/X\,\}}{\sigma_2\{\, g(t_1,\ldots,t_n)/X\,\}}_\alpha$$

SIMILAR FUNCTORS

$$\left[f \sim_\beta g;\ \ \beta > 0;\ \ n \geq 0;\ \ \alpha_0 \stackrel{\text{def}}{=} \alpha \wedge \beta\right]$$

$$\frac{\binom{\sigma_1}{\sigma_2}_{\alpha_0} \vdash \binom{s_1}{t_1} \uparrow_{\alpha_0} \binom{\sigma_1}{\sigma_2}\, u_1 \,\binom{\sigma_1^1}{\sigma_2^1}_{\alpha_1} \cdots \binom{\sigma_1^{n-1}}{\sigma_2^{n-1}}_{\alpha_{n-1}} \vdash \binom{s_n}{t_n} \uparrow_{\alpha_{n-1}} \binom{\sigma_1^{n-1}}{\sigma_2^{n-1}}\, u_n \,\binom{\sigma_1^n}{\sigma_2^n}_{\alpha_n}}{\binom{\sigma_1}{\sigma_2}_\alpha \vdash \binom{f(s_1,\ldots,s_n)}{g(t_1,\ldots,t_n)}\, f(u_1,\ldots,u_n) \,\binom{\sigma_1^n}{\sigma_2^n}_{\alpha_n}}$$

Figure 3.13: Functor-weak generalization axioms and rule

with $\alpha_i \wedge \beta_i \in (0.0, 1.0]$, for $i = 1, 2$. This would mean that X or Y would have been introduced while the side condition of neither Axiom **FUZZY VARIABLE-TERM** nor Axiom **DISSIMILAR FUNCTORS** was satisfied; which is impossible. Thus, there can be at most only one such variable.

As importantly, note also that fuzzy unapplication defined by Equation (3.22) returns a pair of terms and a possibly lesser or equal approximation degree, unlike crisp unapplication defined by Equation (3.7) which returns only a pair of terms. Because of this, when we write a fuzzy judgment such as:

$$\begin{pmatrix}\sigma\\ \sigma'\end{pmatrix}\vdash_{\alpha} \begin{pmatrix}t\\ t'\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\sigma\\ \sigma'\end{pmatrix} u \begin{pmatrix}\theta\\ \theta'\end{pmatrix}_{\beta} \tag{3.24}$$

as we do in the premiss of Rule **SIMILAR FUNCTORS**, this is shorthand to indicate that the posterior similarity degree β is *at most* the one returned by the fuzzy unapplication $\begin{pmatrix}t\\ t'\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\sigma\\ \sigma'\end{pmatrix}$. Formally, the notation of the fuzzy judgment (3.24) is equivalent to:

$$\begin{pmatrix}s\\ s'\end{pmatrix}_{\beta'} \stackrel{\text{def}}{=} \begin{pmatrix}t\\ t'\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\sigma\\ \sigma'\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}\sigma\\ \sigma'\end{pmatrix}_{\beta'} \vdash \begin{pmatrix}s\\ s'\end{pmatrix} u \begin{pmatrix}\theta\\ \theta'\end{pmatrix}_{\beta} \tag{3.25}$$

for some β' such that $\beta \leq \beta' \leq \alpha$. This is because a fuzzy unapplication invoked while proving the validity of a fuzzy judgment may require, by Expression (3.22), lowering the *prior* approximation degree of the judgment.

Finally, note that Rule "**SIMILAR FUNCTORS**" is defined for $n \geq 0$. For $n = 0$, it becomes the following fuzzy judgment:

$$\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha} \vdash \begin{pmatrix}c\\ c\end{pmatrix} c \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha} \tag{3.26}$$

which can be verified to be an axiom since it is valid at any approximation degree α in $[0.0, 1.0]$, for any constant c in Σ_0, and any substitutions σ_1 and σ_2 in $\textbf{SUBST}_{\mathcal{T}}$, thanks to the reflexivity of the similarity $\sim_{\alpha}$ on $\mathcal{T}$.

Referring to the axioms (seen as rules with no antecedent) and the rule of Figure 3.13, we establish the following fact corresponding to Lemma 3.2 on Page 30 (taking $\sigma_i^0 \stackrel{\text{def}}{=} \sigma_i$, for $i = 1, 2$), where the fuzzy ordering on substitutions is defined in Definition 3.7 on Page 39.

LEMMA 3.5 *In Rule* **SIMILAR FUNCTORS** *of Figure 3.13, taking* $\sigma_i^0 \stackrel{\text{def}}{=} \sigma_i$, *for* $i = 1, 2$, *the approximation degrees* $\alpha_i^0, \ldots, \alpha_i^n$ *are such that* $\alpha_i^k \leq \alpha_i^{k-1}$, *and the substitutions* $\sigma_i^0, \ldots, \sigma_i^n$ *are such that* $\sigma_i^k \preceq_{\alpha_i^k} \sigma_i^{k-1}$, *for all* k, $1 \leq k \leq n$ $(i = 1, 2)$.

PROOF We proceed by induction on the depth d of the terms; *i.e.*, we consider only terms of depth less than or equal to d.

1. $d = 0$: This limits terms to constants and variables. The inequality between prior and posterior substitutions is satisfied for the first three axioms of Figure 3.13: each

preserves the prior approximation degree and the posterior substitutions are all either equal to the corresponding prior substitutions or of the form $\theta = \sigma\{t/X\}$ where X is a new variable and σ is the corresponding prior substitution. As well, when limited to terms of 0 depth, Rule **SIMILAR FUNCTORS** becomes the Axiom (3.26), which preserves both the approximation degree and the substitutions.

2. $d > 0$: Let us assume now that this is true for all terms of depth less than d. That is, we consider two terms to generalize, at least one of which is of depth d. The same argument given above for when $d = 0$ for the first three axioms of Figure 3.5 justifies concluding that $\theta \preceq \sigma$, since this is true in these cases for terms of any depth. As for Rule **EQUAL FUNCTORS**, there are two possible cases for the two terms in its consequent (the "denominator"):

   (a) $n = 0$: then, the conclusion follows true by Axiom (3.26);

   (b) $n \geq 0$: since the fuzzy unapply operation (3.22) yields either a pair of terms having the same depth as the corresponding terms it is applied to, or 0 (because it can only be a new variable), we can say that all the terms resulting from fuzzy-unapplied pairs of arguments in the judgments of the rule's antecedent (the "numerator") are of depth at most $d - 1$. Therefore, this fact, together with our induction hypothesis being satisfied for depths less than d and the expression of a fuzzy judgment (3.25) involving only terms of such depths, we can conclude that all the judgments in the rule's antecedent can only reduce their prior approximation degree. Therefore, $\alpha_i^k \leq \alpha_i^{k-1}$ and $\sigma_i^k \preceq_{\alpha_i^k} \sigma_i^{k-1}$, for all $k = 1, \ldots, n$. Then, by Corollary 3.3 and transitivity of the "more general" ordering on substitutions $\preceq_\alpha$ at fixed α, the conclusion follows.

Hence, this establishes that, for both $i = 1, 2$, the approximation degree α_i^k is monotonically decreasing and the substitution σ_i^k is monotonically refined from more general to less, as k increases from 1 to n; which concludes our proof. □

And the corresponding corollary also follows.

COROLLARY 3.4 *In Rule* **SIMILAR FUNCTORS** *of Figure 3.13, for all k, $1 \leq k \leq n$:*

- *the approximation degrees α_i^k are such that $\alpha_i^n \leq \alpha_i^{n-1} \leq \ldots \alpha_i^1 \leq \alpha_i^0$, and*
- *the substitutions σ_i^k are such that $\sigma_i^n \preceq_{\alpha_i^n} \sigma_i^{n-1} \preceq_{\alpha_i^{n-1}} \ldots \sigma_i^1 \preceq_{\alpha_i^1} \sigma_i^0$,*

for $i = 1, 2$.

THEOREM 3.5 (FUNCTOR-WEAK GENERALIZATION CORRECTNESS) *The fuzzy generalization rules of Figure 3.13 are correct.*

PROOF We must show that they satisfy the conditions of Definition 3.12 on page 54. For each of the three axioms of Figure 3.13, this means that they must be valid as fuzzy judgments, satisfying the three conditions of Definition 3.11, which are:

- *Condition 1*: $\beta \in (0, \alpha]$,
- *Condition 2*: $t_i\sigma_i \sim_\beta t\theta_i$,
- *Condition 3*: $t_i \preceq_\alpha t$ and $\theta_i \preceq_\beta \sigma_i$,

for $i = 1, 2$, for a fuzzy $\mathcal{FOT}$ generalization judgment such as (3.21). These conditions for the axioms and the rule of Figure 3.13 translate as the following.

Condition 1. All three axioms satisfy this condition because they preserve the approximation degree.

Condition 2. This condition becomes the following for each of the three axioms (for $i = 1, 2$):

- **FUZZY EQUAL VARIABLES**: Condition 2 becomes the similarity $X\sigma_i \sim_\alpha X\sigma_i$, which is true by reflexivity of $\sim_\alpha$ for all X, σ_i, and α;
- **FUZZY VARIABLE-TERM**: it becomes the similarity $t_i\sigma_i \sim_\alpha t_i\sigma_i$, which is true also by reflexivity of $\sim_\alpha$, for all t_i, σ_i, and α;
- **DISSIMILAR FUNCTORS**: Condition 2 becomes:

$$\begin{aligned} f(s_1, \ldots, s_m)\sigma_1 &\sim_\alpha X\sigma_1\{\, f(s_1, \ldots, s_m)/X \,\} \\ g(t_1, \ldots, t_n)\sigma_2 &\sim_\alpha X\sigma_2\{\, g(t_1, \ldots, t_n)/X \,\} \end{aligned}$$

  which, because X is a new variable that does not occurs in either σ_1 or σ_2, simplify respectively to the similarities:

$$\begin{aligned} f(s_1, \ldots, s_m) &\sim_\alpha f(s_1, \ldots, s_m) \\ g(t_1, \ldots, t_n) &\sim_\alpha g(t_1, \ldots, t_n) \end{aligned}$$

  which hold by reflexivity of $\sim_\alpha$ at any approximation degree α.

Condition 3. The three axioms satisfy the following at all approximation degrees α and β (for $i = 1, 2$):

- **FUZZY EQUAL VARIABLES**: $X \preceq_\alpha X$ and $\sigma_i \preceq_\beta \sigma_i$;
- **FUZZY VARIABLE-TERM**: $t_i \preceq_\alpha X$ and $\sigma_i\{t_i/X\} \preceq_\beta \sigma_i$;
- **DISSIMILAR FUNCTORS**:

$$\begin{aligned} f(s_1, \ldots, s_m) &\preceq_\alpha X \quad \text{and} \quad \sigma_1\{\, f(s_1, \ldots, s_m)/X \,\} \preceq_\beta \sigma_1, \\ g(t_1, \ldots, t_n) &\preceq_\alpha X \quad \text{and} \quad \sigma_2\{\, g(t_1, \ldots, t_n)/X \,\} \preceq_\beta \sigma_2. \end{aligned}$$

As for Rule **SIMILAR FUNCTORS**, as required by Definition 3.12, we must show that if all the fuzzy judgments in the numerator are valid, then the fuzzy judgment in the denominator is valid too. For all three conditions, let us proceed by induction on the arity n:

Condition 1. For $n = 0$, the conclusion follows also because Axiom (3.26) applies and it also preserves the approximation degree; for $n > 0$, if we assume that $0 \leq \alpha_k \leq \alpha_{k-1} \leq 1$ for all $k = 1, \ldots, n$, by transitivity of $\leq$ on $[0.0, 1.0]$, it follows that $0 \leq \alpha_n \leq \alpha_0 \leq 1$, which satisfies the definition.

Condition 2. For $n = 0$, this rule becomes Axiom (3.26). Since it preserves the approximation degree, Condition 1 is satisfied. Also, this fuzzy judgment is trivially valid at all approximation degrees: the conditions of Definition 3.11 become the reflexive similarity $c \sim_\alpha c$, and the conjunction of reflexive fuzzy inequality $c \preceq_\alpha c$ and reflexive substitution fuzzy inequalities $\sigma_i \preceq_\alpha \sigma_i$, for $i = 1, 2$. Thus, this verifies both Condition 2 and Condition 3 for $n = 0$.

For $n > 0$, for each argument-position $k = 1, \ldots, n$, a fuzzy judgment in the rule's antecedent is of the form:

$$\begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix}_{\alpha_{k-1}} \vdash \begin{pmatrix} s_k \\ t_k \end{pmatrix} \uparrow_{\alpha_{k-1}} \begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix} u_k \begin{pmatrix} \sigma_1^{k} \\ \sigma_2^{k} \end{pmatrix}_{\alpha_k};$$

that is, the form of Expression (3.24), whose formal meaning is given as Expression (3.25), which in the above case is equivalent to:

$$\begin{pmatrix} v_1^k \\ v_2^k \end{pmatrix}_{\beta_k} \stackrel{\text{def}}{=} \begin{pmatrix} s_k \\ t_k \end{pmatrix} \uparrow_{\alpha_{k-1}} \begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix}_{\beta_k} \vdash \begin{pmatrix} v_1^k \\ v_2^k \end{pmatrix} u_k \begin{pmatrix} \sigma_1^{k} \\ \sigma_2^{k} \end{pmatrix}_{\alpha_k}$$

for some β_k s.t. $\alpha_{k-1} \leq \beta_k \leq \alpha_k$. Let us now assume that all the fuzzy judgment in the rule's antecedent are valid. That is, for $k = 1, \ldots, n$ (defining $\alpha_0 \stackrel{\text{def}}{=} \alpha \wedge \beta$), for $i = 1, 2$:

$$u_k \sigma_i^k \sim_{\alpha_k} v_i^k \sigma_i^{k-1} \tag{3.27}$$

and (defining $\sigma_i^0 \stackrel{\text{def}}{=} \sigma_i$):

$$v_i^k \preceq_\alpha u_k \text{ and } \sigma_i^k \preceq_\beta \sigma_i^{k-1}. \tag{3.28}$$

By Equation (3.22), this means:

$$\begin{pmatrix} v_1^k \\ v_2^k \end{pmatrix}_{\alpha_k} \stackrel{\text{def}}{=} \begin{cases} \begin{pmatrix} X \\ X \end{pmatrix}_{\alpha_{k-1} \wedge \beta_1^k \wedge \beta_2^k} & \text{if } \exists X \in \mathcal{V} \text{ s.t. } s_k \sim_{\beta_1^k} X\sigma_1^{k-1} \text{ and } t_k \sim_{\beta_2^k} X\sigma_2^{k-1}; \\ \begin{pmatrix} s_k \\ t_k \end{pmatrix}_{\alpha_{k-1}} & \text{otherwise.} \end{cases}$$

for some β_1^k and β_2^k in $(0.0, 1.0]$. In other words, for each $k = 1, \ldots, n$, there are two cases:

1. $s_k \sim_{\beta_1^k} X\sigma_1^{k-1}$ and $t_k \sim_{\beta_2^k} X\sigma_2^{k-1}$ for some variable X; then, by Axiom **FUZZY EQUAL VARIABLES**, we must have $\alpha_k = \alpha_{k-1} \wedge \beta_1^k \wedge \beta_2^k$, $u_k = X$, and $\sigma_i^k = \sigma_i^{k-1}$ for $i = 1, 2$; thus, $\alpha_k \leq \alpha_{k-1}$ and Similarity (3.27) becomes $u_k \sigma_i^k \sim_{\alpha_k} X\sigma_i^{k-1}$ So that:
$$s_k \sigma_1^{k-1} \sim_{\alpha_k} X\sigma_1^{k-1}\sigma_1^{k-1} = X\sigma_1^{k-1} = X\sigma_1^k \sim_{\alpha_k} u_k\sigma_1^k,$$
$$t_k \sigma_2^{k-1} \sim_{\alpha_k} X\sigma_2^{k-1}\sigma_2^{k-1} = X\sigma_2^{k-1} = X\sigma_2^k \sim_{\alpha_k} u_k\sigma_2^k.$$
2. There is no such variable X; in which case, $\alpha_k = \alpha_{k-1}$ and Similarity (3.27) becomes:
$$s_k \sigma_1^{k-1} \sim_{\alpha_k} u_k \sigma_1^k,$$
$$t_k \sigma_2^{k-1} \sim_{\alpha_k} u_k \sigma_2^k.$$

Thus, by the only non-identical transformation relating prior and posteriors substitutions in the axioms, for any argument position k, $1 \leq k \leq n$, we have:

$$\sigma_i^k \sim_{\alpha_k} \sigma_i^0\{\tau_1/X_1\}\ldots\{\tau_\ell/X_\ell\}$$

where each of the variables $X_1 \ldots X_\ell$, with $0 \leq \ell$, is a variable possibly introduced in proving the validity of the fuzzy judgment corresponding to some argument position k. Therefore, since for any argument position k, $1 \leq k \leq n$:

1. σ_i^k affects only a *new* variable introduced in one of the axioms satisfying the validity of the subterm at argument position k; and,
2. such a newly introduced variable now occurring in u_k is always instantiated by the same term;

it comes that, at approximation degree α_k:

$$s_k\sigma_1^0 \sim_{\alpha_k} s_k\sigma_1^1 \sim_{\alpha_k} \ldots \sim_{\alpha_k} s_k\sigma_1^{k-1}$$

$$t_k\sigma_2^0 \sim_{\alpha_k} t_k\sigma_2^1 \sim_{\alpha_k} \ldots \sim_{\alpha_k} t_k\sigma_2^{k-1}$$

as well as, at approximation degree α_n:

$$u_k\sigma_1^k \sim_{\alpha_n} u_k\sigma_1^{k+1} \sim_{\alpha_n} \ldots \sim_{\alpha_n} u_k\sigma_1^n$$

$$u_k\sigma_2^k \sim_{\alpha_n} u_k\sigma_2^{k+1} \sim_{\alpha_n} \ldots \sim_{\alpha_n} u_k\sigma_2^n$$

which shows that in both cases we have, for all $k = 1, \ldots, n$:

$$s_k\sigma_1^0 \sim_{\alpha_k} u_k\sigma_1^n$$

$$t_k\sigma_2^0 \sim_{\alpha_k} u_k\sigma_2^n.$$

Therefore, for $k = n$:

$$f(s_1, \ldots, s_n)\sigma_1^0 \sim_{\alpha_n} f(u_1, \ldots, u_n)\sigma_1^n$$

$$f(t_1, \ldots, t_n)\sigma_2^0 \sim_{\alpha_n} f(u_1, \ldots, u_n)\sigma_2^n$$

which completes the proof of Condition 2.

Condition 3. This condition becomes, for all $k = 1, \ldots, n$:

$$\begin{array}{l} f(s_1, \ldots, s_n) \preceq_{\alpha_{k-1}} f(u_1, \ldots, u_n) \text{ and } \sigma_1^k \preceq_{\alpha_k} \sigma_1^{k-1} \\ g(t_1, \ldots, t_n) \preceq_{\alpha_{k-1}} g(u_1, \ldots, u_n) \text{ and } \sigma_2^k \preceq_{\alpha_k} \sigma_2^{k-1} \end{array}$$

from which, since $\alpha_k \leq \alpha_{k-1}$ for all $k = 1, \ldots, n$, it follows that:

$$\begin{array}{l} f(s_1, \ldots, s_n) \preceq_{\alpha_n} f(u_1, \ldots, u_n) \text{ and } \sigma_1^n \preceq_{\alpha_n} \sigma_1^0 \\ g(t_1, \ldots, t_n) \preceq_{\alpha_n} g(u_1, \ldots, u_n) \text{ and } \sigma_2^n \preceq_{\alpha_n} \sigma_2^0 \end{array}$$

or indifferently, using the same similarity class representative in both cases since $f \sim_{\alpha_n} g$ (because $f \sim_\beta g$ and $\alpha_n \leq \beta$):

$$g(t_1, \ldots, t_n) \preceq_{\alpha_n} f(u_1, \ldots, u_n) \text{ and } \sigma_2^n \preceq_{\alpha_n} \sigma_2^0$$

which completes the proof of Condition 3, and the proof of Theorem 3.5. □

Example 3.11 Fuzzy generalization with similar functors of same arities — Consider the signature Σ containing $\Sigma_0 = \{a, b, c, d\}$, and $\Sigma_2 = \{f, g\}$, and the closure $\sim$ of the similar pairs $a \sim_{.7} b$, $c \sim_{.6} d$, and $f \sim_{.8} g$. Let us apply the functor-weak generalization axioms and rule Figure 3.13 to $t_1 \stackrel{\text{def}}{=} g(c, d)$, and $t_2 \stackrel{\text{def}}{=} f(a, b)$; that is, let us find term t, substitutions $\sigma_i \in \textsc{Subst}_{\mathcal{T}}$ $(i = 1, 2)$, and similarity degree α in $[0.0, 1.0]$ such that $t\sigma_1 \sim_\alpha g(c, d)$ and $t\sigma_2 \sim_\alpha f(a, b)$. This is expressed as the following fuzzy judgment:

$$\binom{\emptyset}{\emptyset}_1 \vdash \binom{g(c,d)}{f(a,b)}\, t \binom{\sigma_1}{\sigma_2}_\alpha .$$

By Rule **Similarity Functors**, we infer that $t = g(u_1, u_2)$:[22]

$$\binom{\emptyset}{\emptyset}_1 \vdash \binom{g(c,d)}{f(a,b)}\, g(u_1, u_2) \binom{\sigma_1}{\sigma_2}_\alpha$$

which, replaced by the antecedents of Rule **Similarity Functors**, becomes (since $g \sim_{.8} f$):

$$\binom{\emptyset}{\emptyset}_{.8} \vdash \binom{c}{a} \uparrow_{.8} \binom{\emptyset}{\emptyset}\, u_1 \binom{\sigma_1'}{\sigma_2'}_{\alpha'}, \binom{\sigma_1'}{\sigma_2'}_{\alpha'} \vdash \binom{d}{b} \uparrow_{\alpha'} \binom{\sigma_1'}{\sigma_2'}\, u_2 \binom{\sigma_1}{\sigma_2}_\alpha .$$

Since the prior substitutions of the first judgment are empty, evaluating its fuzzy unapplication (using Expression (3.25) in which $\beta' = \alpha$) yields the sequence:

$$\binom{\emptyset}{\emptyset}_{.8} \vdash \binom{c}{a}\, u_1 \binom{\sigma_1'}{\sigma_2'}_{\alpha'}, \binom{\sigma_1'}{\sigma_2'}_{\alpha'} \vdash \binom{d}{b} \uparrow_{\alpha'} \binom{\sigma_1'}{\sigma_2'}\, u_2 \binom{\sigma_1}{\sigma_2}_\alpha .$$

By Axiom **Dissimilar Functors**, it comes that $u_1 = X_1$, a new variable, and the sequence becomes:

$$\binom{\emptyset}{\emptyset}_{.8} \vdash \binom{c}{a}\, X_1 \binom{\{\, c/X_1 \,\}}{\{\, a/X_1 \,\}}_{.8}, \binom{\{\, c/X_1 \,\}}{\{\, a/X_1 \,\}}_{.8} \vdash \binom{d}{b} \uparrow_{.8} \binom{\{\, c/X_1 \,\}}{\{\, a/X_1 \,\}}\, u_2 \binom{\sigma_1}{\sigma_2}_\alpha .$$

The validity of the first fuzzy judgment is thereby established. We proceed with the remaining fuzzy judgment evaluating its fuzzy unapplication using Equation (3.22). Since X_1 is such that $d \sim_{.6} X_1\{c/X_1\} = c$ and $a \sim_{.7} X_1\{b/X_1\} = b$, it satisfies the first of the conditions of Equation (3.22). Therefore, the new approximation degree of the judgment is $.8 \wedge .6 \wedge .7 = .6$, and $u_2 = X_1$ so that, by Equation (3.22):

$$\binom{d}{b} \uparrow_{.8} \binom{c/X_1}{a/X_1} = \binom{X_1}{X_1}_{.6}$$

so the second judgment in the sequence becomes:

$$\binom{\{\, c/X_1 \,\}}{\{\, a/X_1 \,\}}_{.6} \vdash \binom{X_1}{X_1}\, X_1 \binom{\{\, c/X_1 \,\}}{\{\, a/X_1 \,\}}_{.6} .$$

This validates the last judgment completing the fuzzy generalization whereby $t = g(X_1, X_1)$ is the least fuzzy generalizer of $t_1 = g(c, d)$, and $t_2 = f(a, b)$ at approximation degree .6 with $\sigma_1 = \{\, c/X_1 \,\}$ so that $t\sigma_1 = g(c, c) \sim_{.6} t_1$; and, $\sigma_2 = \{\, a/X_1 \,\}$ so that $t\sigma_2 = g(a, a) \sim_{.6} t_2$.

[22] This is a non-deterministic choice of a functor's similarity-class representative. We shall always take the left (or upper, in this notation) term's functor. This, of course, will also result in a non-deterministic choice of representative for any term elaborated in generalization modulo functor similarity. The lower the approximation degree, the larger the similarity class.

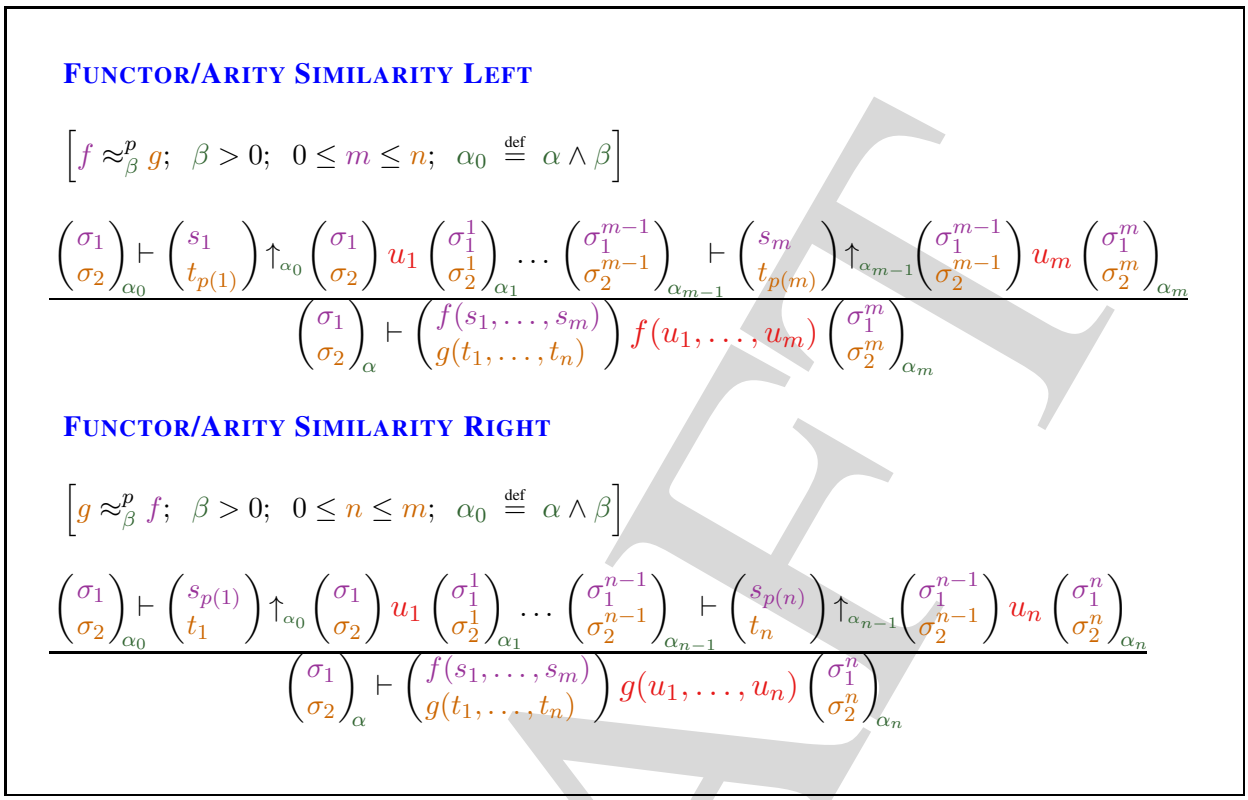


Figure 3.14: Functor/arity-weak generalization rules

Fuzzy functor/arity-weak generalization

In Figure 3.14, we give a fuzzy version of the generalization rules taking into account mismatches not only in functors, but also in arities; *i.e.*, number and/or order of arguments. We now assume that we are not only given a similarity relation $\sim\colon \Sigma \times \Sigma \to [0.0, 1.0]$ on the signature $\Sigma = \cup_{n\geq 0}\Sigma_n$, but also that functors of different arities may be similar with some non-zero similarity degree as specified by a one-to-one argument-position mapping for each pair of so-similar functors associating each argument position of the functor of least arity with a distinct argument position of the functor of larger arity. The only rule among those of Figure 3.13 that differs is the last one (SIMILAR FUNCTORS) which is now a pair of rules called FUNCTOR/ARITY SIMILARITY LEFT and FUNCTOR/ARITY SIMILARITY RIGHT as they account for non-identical correspondence among similar functors's argument positions whether in the left or in the right of the pair of terms to generalize, depending on which side has less arguments. If the arities are the same, the two rules are equivalent (each and all the arguments of the two terms are paired in bijection by a position permutation).

THEOREM 3.6 (FUNCTOR/ARITY-WEAK GENERALIZATION CORRECTNESS) *The fuzzy generalization rules of Figure 3.13 where Rule "SIMILAR FUNCTORS" is replaced with the rules in Figure 3.14 are correct.*

PROOF The argument in this proof has exactly the same structure as the argument for the

proof of Rule **SIMILAR FUNCTORS** of Figure 3.13. The only difference is that structural induction on a pair of terms with similar functors to generalize is always limited to the largest possible set of pairs of corresponding argument positions as specified by a one-to-one argument map from all the argument positions of the functor of lesser arity to those of the functor of larger arity, rather than the identity on equal cardinality sets of argument positions. Thus, in the following, parts of the proof that are omitted are identical to their corresponding parts in the proof of Rule **SIMILAR FUNCTORS**. Also, for reason of obvious symmetry, we need only provide the detailed proof of correctness of Rule **FUNCTOR/ARITY SIMILARITY LEFT**. The proof of correctness of Rule **FUNCTOR/ARITY SIMILARITY RIGHT** is the pointwise similar dual argument in the other direction.

Considering Rule **FUNCTOR/ARITY SIMILARITY LEFT**, as required by Definition 3.12, we must show that if all the fuzzy judgments in the numerator are valid, then the fuzzy judgment in the denominator is valid too. Since the proofs of Condition 1 and Condition 3 are the same for equal-arity functor similarity, we need only provide a proof of Condition 2 of Definition 3.12. Let us proceed by induction on the argument-position number k, for $k = 1, \ldots, m$, where m is the arity of f (the first of the two terms' functor, with the same or a smaller arity as required by the side condition).

For $m = 0$, this rule becomes Axiom (3.26). This fuzzy judgment is trivially valid at all approximation degrees: Condition 2 of Definition 3.11 becomes the reflexive similarity $c \approx_\alpha c$ and Condition 3 becomes the conjunction $c \preceq_\alpha c$ and $\sigma_i \preceq_\alpha \sigma_i$, for $i = 1, 2$. Thus, this satisfies both Condition 2 and Condition 3 for $m = 0$.

For $m > 0$, for each argument-position $k = 1, \ldots, m$, a fuzzy judgment in the rule's antecedent is of the form:

$$\begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix}_{\alpha_{k-1}} \vdash \begin{pmatrix} s_k \\ t_{p(k)} \end{pmatrix} \uparrow_{\alpha_{k-1}} \begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix} u_k \begin{pmatrix} \sigma_1^{k} \\ \sigma_2^{k} \end{pmatrix}_{\alpha_k};$$

that is, the form of Expression (3.24), whose formal meaning is given as Expression (3.25), which in the above case is equivalent to:

$$\begin{pmatrix} v_1^k \\ v_2^k \end{pmatrix}_{\beta_k} \stackrel{\text{def}}{=} \begin{pmatrix} s_k \\ t_{p(k)} \end{pmatrix} \uparrow_{\alpha_{k-1}} \begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \sigma_1^{k-1} \\ \sigma_2^{k-1} \end{pmatrix}_{\beta_k} \vdash \begin{pmatrix} v_1^k \\ v_2^k \end{pmatrix} u_k \begin{pmatrix} \sigma_1^{k} \\ \sigma_2^{k} \end{pmatrix}_{\alpha_k}$$

for some β_k s.t. $\alpha_{k-1} \leq \beta_k \leq \alpha_k$. Let us now assume that all the fuzzy judgment in the rule's antecedent are valid. That is, for $k = 1, \ldots, m$ (defining $\alpha_0 \stackrel{\text{def}}{=} \alpha \wedge \beta$), for $i = 1, 2$:

$$u_k \sigma_i^k \approx_{\alpha_k} v_i^k \sigma_i^{k-1} \tag{3.29}$$

and (defining $\sigma_i^0 \stackrel{\text{def}}{=} \sigma_i$):

$$v_i^k \preceq_\alpha u_k \text{ and } \sigma_i^k \preceq_\beta \sigma_i^{k-1}. \tag{3.30}$$

By Equation (3.22), this means that for all $k = 1, \ldots, m$, v_1^k, v_2^k, and α_k are defined by:

$$\begin{pmatrix} v_1^k \\ v_2^k \end{pmatrix}_{\alpha_k} \stackrel{\text{def}}{=} \begin{cases} \begin{pmatrix} X \\ X \end{pmatrix}_{\alpha_{k-1} \wedge \beta_1^k \wedge \beta_2^k} & \text{if } \exists\, X \in \mathcal{V} \text{ s.t. } \begin{pmatrix} s_k & \approx_{\beta_1^k} & X\sigma_1^{k-1} \\ t_{p(k)} & \approx_{\beta_2^k} & X\sigma_2^{k-1} \end{pmatrix}; \\ \begin{pmatrix} s_k \\ t_{p(k)} \end{pmatrix}_{\alpha_{k-1}} & \text{otherwise.} \end{cases}$$

for some β_1^k and β_2^k in $(0.0, 1.0]$. In other words, for each $k = 1, \ldots, m$, there are two cases:

1. $s_k \approx_{\beta_1^k} X\sigma_1^{k-1}$ and $t_{p(k)} \approx_{\beta_2^k} X\sigma_2^{k-1}$ for some variable X; then, by Axiom **FUZZY EQUAL VARIABLES**, we must have $\alpha_k = \alpha_{k-1} \wedge \beta_1^k \wedge \beta_2^k$, $u_k = X$, and $\sigma_i^k = \sigma_i^{k-1}$ for $i = 1, 2$; thus, $\alpha_k \leq \alpha_{k-1}$ and Similarity (3.29) becomes $u_k\sigma_i^k \approx_{\alpha_k} X\sigma_i^{k-1}$ So that:

$$\begin{aligned} s_k\sigma_1^{k-1} &\approx_{\alpha_k} X\sigma_1^{k-1}\sigma_1^{k-1} = X\sigma_1^{k-1} = X\sigma_1^k \approx_{\alpha_k} u_k\sigma_1^k, \\ t_{p(k)}\sigma_2^{k-1} &\approx_{\alpha_k} X\sigma_2^{k-1}\sigma_2^{k-1} = X\sigma_2^{k-1} = X\sigma_2^k \approx_{\alpha_k} u_k\sigma_2^k. \end{aligned}$$

2. There is no such variable X; in which case, $\alpha_k = \alpha_{k-1}$ and Similarity (3.29) becomes:

$$\begin{aligned} s_k\sigma_1^{k-1} &\approx_{\alpha_k} u_k\sigma_1^k, \\ t_{p(k)}\sigma_2^{k-1} &\approx_{\alpha_k} u_k\sigma_2^k. \end{aligned}$$

Thus, by the only non-identical transformation relating prior and posteriors substitutions in the axioms, for any argument position k, $1 \leq k \leq m$, we have:

$$\sigma_i^k \approx_{\alpha_k} \sigma_i^0\{\tau_1/X_1\}\ldots\{\tau_\ell/X_\ell\}$$

where each of the variables $X_1 \ldots X_\ell$, with $0 \leq \ell$, is a variable possibly introduced in proving the validity of the fuzzy judgment corresponding to some argument position preceding k. Therefore, since for any argument position k, $1 \leq k \leq m$:

1. σ_i^k affects only a *new* variable introduced in one of the axioms verifying the validity of the subterm at argument position k; and,
2. such a newly introduced variable now occurring in u_k is always instantiated by the same term;

it comes that, at approximation degree α_k:

$$\begin{aligned} s_k\sigma_1^0 &\approx_{\alpha_k} s_k\sigma_1^1 \approx_{\alpha_k} \ldots \approx_{\alpha_k} s_k\sigma_1^{k-1} \\ t_{p(k)}\sigma_2^0 &\approx_{\alpha_k} t_{p(k)}\sigma_2^1 \approx_{\alpha_k} \ldots \approx_{\alpha_k} t_{p(k)}\sigma_2^{k-1} \end{aligned}$$

as well as, at approximation degree α_m:

$$\begin{aligned} u_k\sigma_1^k &\approx_{\alpha_m} u_k\sigma_1^{k+1} \approx_{\alpha_m} \ldots \approx_{\alpha_m} u_k\sigma_1^m \\ u_k\sigma_2^k &\approx_{\alpha_m} u_k\sigma_2^{k+1} \approx_{\alpha_m} \ldots \approx_{\alpha_m} u_k\sigma_2^m \end{aligned}$$

This means that in both cases we have, for all $k = 1, \ldots, m$:

$$s_k\sigma_1^0 \quad \approx_{\alpha_m} \quad u_k\sigma_1^m$$
$$t_{p(k)}\sigma_2^0 \approx_{\alpha_m} \quad u_k\sigma_2^m.$$

Therefore, for $k = m$:

$$f(s_1, \ldots, s_m)\sigma_1^0 \quad \approx_{\alpha_m} \quad f(u_1, \ldots, u_m)\sigma_1^m$$
$$f(t_{p(1)}, \ldots, t_{p(m)})\sigma_2^0 \approx_{\alpha_m} \quad f(u_1, \ldots, u_m)\sigma_2^m$$

which completes the proof of Condition 2 of Theorem 3.6, and that of the theorem because of the facts stated at the outset regarding all other cases each of whose proof is identical to when arities are equal. □

Example 3.12 Fuzzy generalization with similar functors of different arities — Let us take again the functor signature of Example 3.9 where $\{ a, b, c, d \} \subseteq \Sigma_0$, $\{ f, g, \ell \} \subseteq \Sigma_2$, and $\{ h \} \subseteq \Sigma_3$, with similarity defined as the reflexive symmetric transitive closure of the following pairs, along with their argument alignment maps: $a \approx_{.7} b$, $c \approx_{.6} d$, $f \approx_{.9}^{\{ 1\mapsto 2, 2\mapsto 1 \}} g$ and $g \approx_{.9}^{\{ 1\mapsto 2, 2\mapsto 1 \}} f$, and $\ell \approx_{.8}^{\{ 1\mapsto 2, 2\mapsto 3 \}} h$ (and equal similarity degree for symmetric entries). With this signature and similarity, let us try to find the fuzzy generalization of $t_1 \stackrel{\text{def}}{=} h(X, g(Y, b), f(Y, c))$, and $t_2 \stackrel{\text{def}}{=} \ell(f(a, Z), g(d, c))$.

Thus we need to us find the most general term $t \in \mathcal{T}$ along with two most general substitutions $\sigma_i :\in \textsc{Subst}_{\mathcal{T}}(i = 1, 2)$, and the maximal similarity degree $\alpha \in [0.0, 1.0]$, such that $t\sigma_1 \approx_\alpha^{\mathcal{T}} h(X, g(Y, b), f(Y, c))$ and $t\sigma_2 \approx_\alpha^{\mathcal{T}} \ell(f(a, Z), g(d, c))$; that is, solve the following fuzzy generalization constraint problem:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix}_1 \vdash \begin{pmatrix} h(X, g(Y, b), f(Y, c)) \\ \ell(f(a, Z), g(d, c)) \end{pmatrix} t \begin{pmatrix} \sigma_1 \\ \sigma_2 \end{pmatrix}_\alpha .$$

Rule **Functor/Arity Similarity Right** entails $t = \ell(u_1, u_2)$, and because $\ell \approx_{.8}^{\{ 1\mapsto 2, 2\mapsto 3 \}} h$:

- u_1 is the fuzzy generalization of $\begin{pmatrix} g(Y, b) \\ g(d, c) \end{pmatrix} \uparrow_{.8} \begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix}$; that is, of $g(Y, b)$ and $g(d, c)$, such that:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix}_{.8} \vdash \begin{pmatrix} g(Y, b) \\ g(d, c) \end{pmatrix} u_1 \begin{pmatrix} \sigma_1^1 \\ \sigma_2^1 \end{pmatrix}_{\alpha_1} ;$$

  i.e., $u_1 = g(v_1, v_2)$ by Rule **Functor/Arity Similarity Left** with $g \approx_1^{\{ 1\mapsto 1, 2\mapsto 2 \}} g$, s.t.:[23]

  – $v_1 = U$ since by Rule **Fuzzy Variable-Term**:

$$\begin{pmatrix} \emptyset \\ \emptyset \end{pmatrix}_{.8} \vdash \begin{pmatrix} Y \\ d \end{pmatrix} U \begin{pmatrix} \{ Y/U \} \\ \{ d/U \} \end{pmatrix}_{.8} ;$$

[23] Note that, for two terms of equal arity, using either **Functor/Arity Similarity Left** or **Functor/Arity Similarity Right** is always equivalent. As explained in Figure 3.9, similar functors of equal arity have a pair of mutually inverse bijections map corresponding argument positions in each term with the other's. Therefore, choosing any of the two functors for the generalized term yields a term in the same similarity class. This is true *a fortiori* when the two functors are equal, as in this case, when the similarity degree is 1 and the argument map is the identity.

– $v_2 = V$ since with $b \not\approx c$ and by **DISSIMILAR FUNCTORS**:

$$\begin{pmatrix}\{\,Y/U\,\}\\ \{\,d/U\,\}\end{pmatrix}_{.8} \vdash \begin{pmatrix}b\\ c\end{pmatrix} V \begin{pmatrix}\{\,Y/U, b/V\,\}\\ \{\,d/U, c/V\,\}\end{pmatrix}_{.8};$$

and so $\sigma_1^1 = \{\,Y/U, b/V\,\}$, $\sigma_2^1 = \{\,d/U, c/V\,\}$, and $\alpha_1 = .8$; that is:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.8} \vdash \begin{pmatrix}g(Y,b)\\ g(d,c)\end{pmatrix} g(U,V) \begin{pmatrix}\{\,Y/U, b/V\,\}\\ \{\,d/U, c/V\,\}\end{pmatrix}_{.8};$$

- u_2 is the fuzzy generalization of $\begin{pmatrix}f(Y,c)\\ f(a,Z)\end{pmatrix} \uparrow_{.8} \begin{pmatrix}\{\,Y/U, b/V\,\}\\ \{\,d/U, c/V\,\}\end{pmatrix}$; that is, of $f(Y,c)$ and $f(a,Z)$, s.t. $u_2 = f(w_1, w_2)$ by Rule **FUNCTOR/ARITY SIMILARITY LEFT** with $f \approx_1^{\{1\mapsto 1, 2\mapsto 2\}} f$ (or Rule **FUNCTOR/ARITY SIMILARITY RIGHT**):

$$\begin{pmatrix}\{\,Y/U, b/V\,\}\\ \{\,d/U, c/V\,\}\end{pmatrix}_{.8} \vdash \begin{pmatrix}f(Y,c)\\ f(a,Z)\end{pmatrix} f(w_1, w_2) \begin{pmatrix}\sigma_1^2\\ \sigma_2^2\end{pmatrix}_{\alpha_2};$$

where:

– by Rule **FUZZY VARIABLE-TERM**:

$$\begin{pmatrix}\{\,Y/U, b/V\,\}\\ \{\,d/U, c/V\,\}\end{pmatrix}_{.8} \vdash \begin{pmatrix}Y\\ a\end{pmatrix} W \begin{pmatrix}\{\,Y/U, b/V, Y/W\,\}\\ \{\,d/U, c/V, a/W\,\}\end{pmatrix}_{.8};$$

so $w_1 = W$; and,

– by Rule **FUZZY VARIABLE-TERM**:

$$\begin{pmatrix}\{\,Y/U, b/V, Y/W\,\}\\ \{\,d/U, c/V, a/W\,\}\end{pmatrix}_{.8} \vdash \begin{pmatrix}c\\ Z\end{pmatrix} C \begin{pmatrix}\{\,Y/U, b/V, Y/W, c/C\,\}\\ \{\,d/U, c/V, a/W, Z/C\,\}\end{pmatrix}_{.8};$$

so $w_2 = C$;

thus $\sigma_1^2 = \{\,Y/U, b/V, Y/W, c/C\,\}$, $\sigma_2^2 = \{\,d/U, c/V, a/W, Z/C\,\}$, and $\alpha_2 = .8$.

Therefore,

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{1} \vdash \begin{pmatrix}h(X, g(Y,b), f(Y,c))\\ \ell(f(a,Z), g(d,c))\end{pmatrix} \ell(f(W,C), g(U,V)) \begin{pmatrix}\{\,Y/U, b/V, Y/W, c/C\,\}\\ \{\,d/U, c/V, a/W, Z/C\,\}\end{pmatrix}_{.8};$$

that is, $t = \ell(f(W,C), g(U,V))$, with:

$$\sigma_1 = \{\,Y/U, b/V, Y/W, c/C\,\} \text{ and } \sigma_2 = \{\,d/U, c/V, a/W, Z/C\,\}$$

and $\alpha = .8$, since:

$$t_1 = t\sigma_1 = \ell(f(Y,c), g(Y,b)) \approx_{.8}^{\mathcal{T}} h(X, g(Y,b), f(Y,c)) \text{ and } t_2 = t\sigma_2 = \ell(f(a,Z), g(d,c)).$$

Example 3.13 Example 3.12 with more expressive symbols — Let us again, as we did in Example 3.10 (gift-shop Prolog database), give more expressive names to functors of Example 3.12:

- $a/0 \stackrel{\text{def}}{=} \mathtt{violet}, b/0 \stackrel{\text{def}}{=} \mathtt{lilac}, c/0 \stackrel{\text{def}}{=} \mathtt{chocolate}, d/0 \stackrel{\text{def}}{=} \mathtt{candy}$,
- $f/2 \stackrel{\text{def}}{=} \mathtt{pair}, g/2 \stackrel{\text{def}}{=} \mathtt{couple}$,
- $\ell/2 \stackrel{\text{def}}{=} \mathtt{small\text{-}gift\text{-}bag}, h/3 \stackrel{\text{def}}{=} \mathtt{small\text{-}gift\text{-}box}$,

with the following similarity degrees and argument maps,:

- $\mathtt{violet} \approx_{.7} \mathtt{lilac}$,
- $\mathtt{chocolate} \approx_{.6} \mathtt{candy}$,
- $\mathtt{pair} \approx_{.9}^{\{1\mapsto 2, 2\mapsto 1\}} \mathtt{couple}$ and $\mathtt{couple} \approx_{.9}^{\{1\mapsto 2, 2\mapsto 1\}} \mathtt{pair}$,
- $\mathtt{small\text{-}gift\text{-}bag} \approx_{.8}^{\{1\mapsto 2, 2\mapsto 3\}} \mathtt{small\text{-}gift\text{-}box}$.

With these functors symbols, the generalization problem of Example 3.9 appears now with the terms:

$$\begin{array}{rcl} t_1 & \stackrel{\text{def}}{=} & \mathtt{small\text{-}gift\text{-}box}(\ X \\ & & \quad , \ \mathtt{couple}(Y, \mathtt{lilac}) \\ & & \quad , \ \mathtt{pair}(Y, \mathtt{chocolate}) \\ & & \quad) \\ t_2 & \stackrel{\text{def}}{=} & \mathtt{small\text{-}gift\text{-}bag}(\ \mathtt{pair}(\mathtt{violet}, Z) \\ & & \quad , \ \mathtt{couple}(\mathtt{candy}, \mathtt{chocolate}) \\ & & \quad) \end{array}$$

which are the results of applying the substitutions:

$$\sigma_1 = \{ \ Y/U, \mathtt{lilac}/V, Y/W, \mathtt{chocolate}/C \ \}$$

and

$$\sigma_2 = \{ \ \mathtt{candy}/U, \mathtt{chocolate}/V, \mathtt{violet}/W, Z/C \ \}$$

to the least fuzzy generalization:

$$\begin{array}{rcl} t & \stackrel{\text{def}}{=} & \mathtt{small\text{-}gift\text{-}bag}(\ \mathtt{pair}(W, C) \\ & & \quad , \ \mathtt{couple}(U, V) \\ & & \quad) \end{array}$$

obtained after normalization with a similarity degree .8.

Example 3.14 Fuzzy generalization with similar functors of different arities — 2nd example — Consider the signature Σ containing $\Sigma_0 = \{a, b, c, d\}$, $\Sigma_2 = \{f, g, l\}$, and $\Sigma_3 = \{h\}$, and the closure $\sim$ of the similar pairs $a \sim_{.7} c \sim_{.6} d$, $f \sim_{.8} g$, and $l \sim_{.9} h$. Let us take all argument-position mappings as the default (identity on least-arity set). Let us apply the fuzzy generalization axioms of Figure 3.13 and the rule of Figure 3.14 to $t_1 \stackrel{\text{def}}{=} h(g(b, Y), f(Y, c), V)$, and $t_2 \stackrel{\text{def}}{=} l(f(a, Z), g(c, d))$; that is, let us find term t, substitutions $\sigma_i \in \textsc{Subst}_{\mathcal{T}}$ $(i = 1, 2)$, and similarity degree α in $[0.0, 1.0]$, such that

$t\sigma_1 \sim_\alpha h(g(b,Y),f(Y,c),V)$ and $t\sigma_2 \sim_\alpha l(f(a,Z),g(c,d))$. This is expressed as the following fuzzy judgment:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_1 \vdash \begin{pmatrix}h(g(b,Y),f(Y,c),V)\\ l(f(a,Z),g(c,d))\end{pmatrix} t \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha .$$

By Rule **FUNCTOR/ARITY SIMILARITY RIGHT**, we can infer that $t = l(u_1,u_2)$:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_1 \vdash \begin{pmatrix}h(g(b,Y),f(Y,c),V)\\ l(f(a,Z),g(c,d))\end{pmatrix} l(u_1,u_2) \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha$$

which, when replaced by the rule's antecedents, since $h \sim_{.9} l$ and $1 \wedge .9 = .9$, becomes the sequence:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.9} \vdash \begin{pmatrix}g(b,Y)\\ f(a,Z)\end{pmatrix} \uparrow_{.9} \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} u_1 \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'}, \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} \uparrow_{\alpha'} \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha .$$

By evaluating the fuzzy unapplication in its first judgment, this sequence becomes:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.9} \vdash \begin{pmatrix}g(b,Y)\\ f(a,Z)\end{pmatrix} u_1 \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'}, \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} \uparrow_{\alpha'} \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha .$$

By Rule **FUNCTOR/ARITY SIMILARITY LEFT**,[24] it comes that $u_1 = g(u_3,u_4)$ and, since $g \sim_{.8} f$ and $.9 \wedge .8 = .8$, the sequence becomes:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.8} \vdash \begin{pmatrix}b\\ a\end{pmatrix} \uparrow_{.8} \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} u_3 \begin{pmatrix}\sigma_1''\\ \sigma_2''\end{pmatrix}_{\alpha''}, \begin{pmatrix}\sigma_1''\\ \sigma_2''\end{pmatrix}_{\alpha''} \vdash \begin{pmatrix}Y\\ Z\end{pmatrix} \uparrow_{\alpha''} \begin{pmatrix}\sigma_1''\\ \sigma_2''\end{pmatrix} u_4 \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'},$$

$$\begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} \uparrow_{\alpha'} \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha .$$

By evaluating the fuzzy unapplication in the first judgment, and using Rule **FUNCTOR/ARITY SIMILARITY LEFT** in the 0-arity case as Axiom (3.26), since $b \sim_{.7} a$ and $.8 \wedge .7 = .7$, we have $u_3 = b$, and the sequence becomes:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.7} \vdash \begin{pmatrix}b\\ a\end{pmatrix} b \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.7}, \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.7} \vdash \begin{pmatrix}Y\\ Z\end{pmatrix} \uparrow_{.7} \begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} u_4 \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'},$$

$$\begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{\alpha'} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} \uparrow_{\alpha'} \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha .$$

The validity of the first fuzzy judgment is thereby established. We proceed with the remaining sequence of fuzzy judgments evaluating the fuzzy unapplication in the first of its judgments, which sets $\alpha' = .7$:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.7} \vdash \begin{pmatrix}Y\\ Z\end{pmatrix} u_4 \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{.7}, \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix}_{.7} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} \uparrow_{.7} \begin{pmatrix}\sigma_1'\\ \sigma_2'\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_\alpha .$$

[24] Since f and g have equal arities, we could also use Rule **FUNCTOR/ARITY SIMILARITY RIGHT**. This would end in an equivalent final result, modulo functor similarities at the final approximation degree. In the remainder of this example, we shall omit making this remark, and choose the left rule over the right for equal-arity functors.

By Axiom **FUZZY VARIABLE-TERM**, we infer from this that $u_4 = X_1$, a new variable, and the judgments become:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{.7} \vdash \begin{pmatrix}Y\\ Z\end{pmatrix} X_1 \begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix}_{.7},$$

$$\begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix}_{.7} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} \uparrow_{.7} \begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha}.$$

The validity of the first fuzzy judgment of the above sequence is thereby established. We proceed with the remainder evaluating the fuzzy unapplication in the first of its judgments, which returns the same pair of terms with the similarity degree kept at .7:

$$\begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix}_{.7} \vdash \begin{pmatrix}f(Y,c)\\ g(c,d)\end{pmatrix} u_2 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha}.$$

and by Rule **FUNCTOR/ARITY SIMILARITY LEFT** with $u_2 = f(u_5, u_6)$, this becomes:

$$\begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix}_{.7} \vdash \begin{pmatrix}Y\\ c\end{pmatrix} \uparrow_{.7} \begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix} u_5 \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix}_{\beta}, \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix}_{\beta} \vdash \begin{pmatrix}c\\ d\end{pmatrix} \uparrow_{\beta} \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix} u_6 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha}.$$

Evaluating the fuzzy unapplication gives $\beta = .7$:

$$\begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix}_{.7} \vdash \begin{pmatrix}Y\\ c\end{pmatrix} u_5 \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix}_{.7}, \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix}_{.7} \vdash \begin{pmatrix}c\\ d\end{pmatrix} \uparrow_{.7} \begin{pmatrix}\theta_1\\ \theta_2\end{pmatrix} u_6 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha}.$$

and by Axiom **FUZZY VARIABLE-TERM**, we infer from this that $u_5 = X_2$, a new variable, which yields:

$$\begin{pmatrix}\{\ Y/X_1\ \}\\ \{\ Z/X_1\ \}\end{pmatrix}_{.7} \vdash \begin{pmatrix}Y\\ c\end{pmatrix} X_2 \begin{pmatrix}\{\ Y/X_1, Y/X_2\ \}\\ \{\ Z/X_1, c/X_2\ \}\end{pmatrix}_{.7},$$

$$\begin{pmatrix}\{\ Y/X_1, Y/X_2\ \}\\ \{\ Z/X_1, c/X_2\ \}\end{pmatrix}_{.7} \vdash \begin{pmatrix}c\\ d\end{pmatrix} \uparrow_{.7} \begin{pmatrix}\{\ Y/X_1, Y/X_2\ \}\\ \{\ Z/X_1, c/X_2\ \}\end{pmatrix} u_6 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha},$$

and establishes the penultimate judgment. The last remaining judgment, after evaluating its fuzzy unapplication, since $c \sim_{.6} d$ and $.7 \wedge .6 = .6$, is:

$$\begin{pmatrix}\{\ Y/X_1, Y/X_2\ \}\\ \{\ Z/X_1, c/X_2\ \}\end{pmatrix}_{.6} \vdash \begin{pmatrix}c\\ d\end{pmatrix} u_6 \begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha},$$

for which Axiom **FUZZY VARIABLE-TERM** allows us to infer that $u_6 = c$ and $\alpha = .6$:

$$\begin{pmatrix}\{\ Y/X_1, Y/X_2\ \}\\ \{\ Z/X_1, c/X_2\ \}\end{pmatrix}_{.6} \vdash \begin{pmatrix}c\\ d\end{pmatrix} c \begin{pmatrix}\{\ Y/X_1, Y/X_2\ \}\\ \{\ Z/X_1, c/X_2\ \}\end{pmatrix}_{.6}.$$

This validates the last judgment and completes the fuzzy generalization whereby $t = l(g(b, X_1), f(X_2, c))$ is the least fuzzy generalizer of $t_1 = h(g(b, Y), f(Y, c), V)$ and $t_2 = l(f(a, Z), g(c, d))$ at approximation degree .6, with:

- $\sigma_1 = \{\ Y/X_1, Y/X_2\ \}$ so that $t\sigma_1 = l(g(b, Y), f(Y, c)) \sim_{.6} t_1$; and,
- $\sigma_2 = \{\ Z/X_1, c/X_2\ \}$ so that $t\sigma_2 = l(g(b, Z), f(c, c)) \sim_{.6} t_2$.

3.8 Relation to Other Works

There have been other works dealing with the larger issue of integrating general equational theories into logical reasoning, not just the specific theory of fuzzy equivalence among terms. Among the most formally and operationally complete approach, pioneered by Goguen *et al.* in the seventies, is the set of works based on *initial algebras* [84].[25] We define an operator algebra as initial iff there exists a homomorphism from it to all other algebras that are semantic models of first-order terms freely built with these operators and variables [82]. Namely, initiality is the property that guarantees that the formal meaning of syntactic terms defined for $\mathcal{FOT}$s modulo congruence classes defined by equations is preserved for all interpretations. This result was later extended from equational systems to implicational systems by Mahr and Makowsky [119]. In this latter paper, it was also shown that Horn Logic (*i.e.*, an implicational system consitituting the formal basis of the Prolog language), is the largest class of logic that admits an initial algebra semantics.

While the initial algebra approach has shown its general applicability for term unification and generalization, including more recently with the work of Alpuente *et al.* over order-sorted signatures [30] and with equational theories [31], its specific application to fuzzy congruences has yet to be done. While it could conceivably be specified by instantiating the general scheme of initial semantics, this must be at the expense of both formal and operational simplicity when compared with our approach which can be expressed as a direct extension of conventional operations of unification and generalization of first-order terms. This is because it is specifically adapted to a fuzzy equivalence of terms rather than general-purpose reasoning modulo equational theories. In the latter more general approach, equations can be used as terminating rewrite rules and unification modulo this theory is made operational using the general equation-solving technique known as "*narrowing*" [77], [76]. However, for this to work, one must have a terminating and confluent set of rewrite rules. Such may be derived in some cases based on undecidable procedures such as Knuth-Bendix completion [109], or unterminating term rewriting [73]. Rather, we limit ourselves to the obviously decidable theory generated homomorphically on $\mathcal{FOT}$s from a fuzzy equivalence relation (a similarity) on a finite signature. This follows the intuition behind Maria Sessa's formal work on fuzzy $\mathcal{FOT}$ unification as well [157]. Also, we support arity and argument position mismatch for similar operators.[26] Be that as it may, one could envisage studying how E-unification for common equational theories such as associativity or commutativity could be extended to unification of terms with similar functors. However, this is another issue and we do not do so in this book's setting.

There have been also works in logic-based databases such as using a similarity degree while comparing syntactically unequal terms; for example, Francesca Arcelli *et al.*'s LIKELOG database logic programming language [37], [35]. These works, however, concern only *ground* terms (*i.e.*, with no variables), not first-order terms (*i.e.*, possibly having variables). Arcelli *et al.*'s notion of similarity distance between terms was later extended from ground terms to first-order terms by Shroeder and Gilbert in the fuzzy logic programming language FURY [81], [154], [155]. They use the same concept as Arcelli *et al.*'s fuzzy equivalence but derived from dynamically evaluat-

[25] An initial algebra is also called *free* algebra, or *syntactic* algebra, or *tree* algebra, or *term* algebra — because its elements are the syntactic term structures one can define recursively by nesting terms as arguments of other terms (*i.e.*, the model taking as interpretation homomorphism the identity function on terms).

[26] See Section 3.7.1.

ing so-called "*edit distance*" between strings on ground terms as well as first-order terms.[27] Thus, their objective is to derive dynamically an estimate of an "edit distance" between terms. The same comments also apply to work by Kutsia *et al.* [112], [113], where the objective is to check all the possibilities of dynamically matching $\mathcal{FOT}$s with *equal* function symbols having unspecified number of arguments (*i.e.*, the same sort of search objective pursued in FURY, where this is done for *unequal* symbols as well). This objective is not ours in that we are not trying to infer dynamically distances between terms. Rather, we *assume given* a matrix of similarity degrees for such a static similarity relation on term constructors and use this information exactly in the same manner as done by Maria Sessa in [157]. In Sessa's context (and ours), this information is *given statically*, not inferred dynamically. Finally, the main advantage of working from a given static matrix of similarity degrees rather than estimating syntactic edit distances (whether dynamically or statically) is that similarity may be semantic among syntactically unrelated but semantically close strings. The context of dynamic syntactic distance estimation is typically for applications of purely lexical variants such as estimating gene similarity in biology [81]. Ours concerns deriving approximate solutions to fuzzy equations given similarity among term constructors.

Our last reference to other work related to unification and generalization of graph data and type structures, is the set of work due to Aït-Kaci *et al.* (*i.e.*, [23] and [26]). In this context, nodes denote sorts (that are organized in a lattice) and arrows denote features (functional attributes between sort nodes); variables denote equations among (possibly cyclic) feature paths. Just as such subsumption and its lattice operations on $\mathcal{FOT}$s can be fuzzified, so can indeed the lattice of Order-Sorted Feature terms. We address this topic in Chapter 4. This is further extended in Chapter 5 to fuzzy lattice operations of $\mathcal{OSF}$ terms over similar sorts modulo feature alignment.[28] This is also relevant to non-aligned knowledge bases [114].

3.9 Recapitulation

In this chapter, we have developed a formal derivation of fuzzy lattice operations for the data structure known as first-order term. This is achieved by means of syntax-driven constraint normalization rules for both unification and generalization. These operations are then extended to enable arbitrary mismatch between similar terms whether functor-based, arity-based (number and order), or combinations. In Chapter 5, we shall see how to extend these operations to consistent *partial* mappings of argument positions between similar functors.[29]

This last lattice of $\mathcal{FOT}$s permits Fuzzy Logic Programming over arbitrary misaligned-data bases, or more generally Approximate Information Retrieval (using fuzzy unification) and Approximate Knowledge Acquisition (using fuzzy generalization) over heterogeneous but similar data models, whereby approximating modulo constructor similarity is consistently conjugated with approximating with less structure details.

For what concerns implementation, the prospects are many and discussed in Chapter 6.[30] The most immediate concerns implementation of such operations in the form of public libraries to

[27] See [153] this email discussion on the issue.
[28] Section 5.2.
[29] Section 5.1.
[30] Section 6.4.

complement extant tools for first-order terms and substitutions [99].

As for what perspectives this work may open, there are several avenues to explore. There are several other disciplines where this technology has potential for fuzzifying applications wherever $\mathcal{FOT}$s are used for their lattice-theoretic properties such as linguistics and learning. Finally, most promising is using this work's approach to more generic and more expressive knowledge structures for applications such as Information Retrieval (*e.g.*, in the line of [43]), or Data and Knowledge Base Management, Ontology Alignment, *etc.*, . . .

Chapter 4

Version of April 10, 2020

Order-Sorted Feature Terms

In the previous chapter, it may have come many times to the reader's mind that a $\mathcal{FOT}$ is just syntax for a *tree* where node labels are functors except possibly for some leaf nodes that are replaced with variables, and functor-labeled nodes have position-numbered arrows pointing to the root of the subterm tree at each argument position. It is so indeed. In fact, when the leaf nodes that have the same variable are joined, it is a rooted directed acyclic graph (or "*dag*").[1] As such, it is a special case of a more general kind of labeled rooted (possibly cyclic) graph — called a rooted *order-sorted feature* ($\mathcal{OSF}$) graph. Sort symbols are node labels. Sorts are partially ordered to reflect subsumption and form a lattice. Feature symbols labeling arrows represent functional attributes. These graphs are so ordered by endomorphic structure-preserving (sort, feature, and feature path equations) subsumption. Lattice-theoretic operations for these more general rooted graphs, labeled with partially-ordered sorts and with features, can be shown to extend those on more restricted kinds of rooted graphs such as $\mathcal{FOT}$s, labeled with functors, positions, and variables.

In this chapter, we elaborate on these notions. In Section 4.1, we start with an informal introduction to the $\mathcal{OSF}$ formalism (Section 4.1.1), then continue with a formal presentation (Section 4.1.2), leading to expressing $\mathcal{OSF}$ graph lattice operations declaratively as constraint normalization. In Section 4.1.3, we characterize these operations in terms of solving constraints where solutions are functional mappings between reference tags. These mappings generalize to $\mathcal{OSF}$ graphs the notion of $\mathcal{FOT}$ substitution.[2] In Section 4.2, we turn to fuzzifying $\mathcal{OSF}$ graph subsumption and its lattice operations: in Section 4.2.2 we explicate fuzzy $\mathcal{OSF}$ graph unification, and in Section 4.2.3 fuzzy $\mathcal{OSF}$ graph generalization.

4.1 $\mathcal{OSF}$ Formalism

The $\mathcal{OSF}$ formalism extends to typed attributed data structures the representation, syntax, and operations enjoyed by $\mathcal{FOT}$s as used in Logic Programming. It was developed to express rigorously the meaning and properties of graph structures used in the experimental language $\mathcal{LIFE}$ to

[1] A rooted (directed) graph is one with a distinguished node, called its *root*, from which all other nodes can be reached.

[2] For implementation issues, see [10], and [16].

represent data and knowledge as, respectively, elements and types [7], [17].

A formal semantics for $\mathcal{OSF}$ structures can be precisely expressed using each of these three mathematical frameworks — (1) algebraic (as terms), (2) logical (as clauses), and (3) operational (as graphs). In [23], the mathematical equivalence of these three formal semantics was established with explicit cross-interpretations. In this part, we shall rely on this result by (1) defining lattice-theoretic operations on the algebra of $\mathcal{OSF}$ terms, (2) defining corresponding constraint normalization rules for these operations on the logic of $\mathcal{OSF}$ clauses, and (3) deriving an implicit operational semantics from these declarative rules on $\mathcal{OSF}$ graph structures.

Next, we start with a brief informal description of the kind of labeled graphs that $\mathcal{OSF}$ structures are, and how they may be expressed syntactically in a manner that naturally extends the syntax of algebraic $\mathcal{FOT}$s. This term syntax is further transformed into a clausal language upon which unification can be rendered as constraint normalization. We then delve into more details defining this syntax formally as well as $\mathcal{OSF}$ term subsumption, unification, and generalization.

4.1.1 Informal background

Let us take an example (taken from [10]). Let us assume that we wish to describe a 30-year-old person with an id consisting of a name, itself made of two parts: a first name and a last name, both represented as strings. Let us also say that we wish to indicate that such a person has a spouse that is a person sharing his or her id's last name, and such that this latter person's spouse is the first person in question. We propose to represent this information as the graph in Figure 4.1.

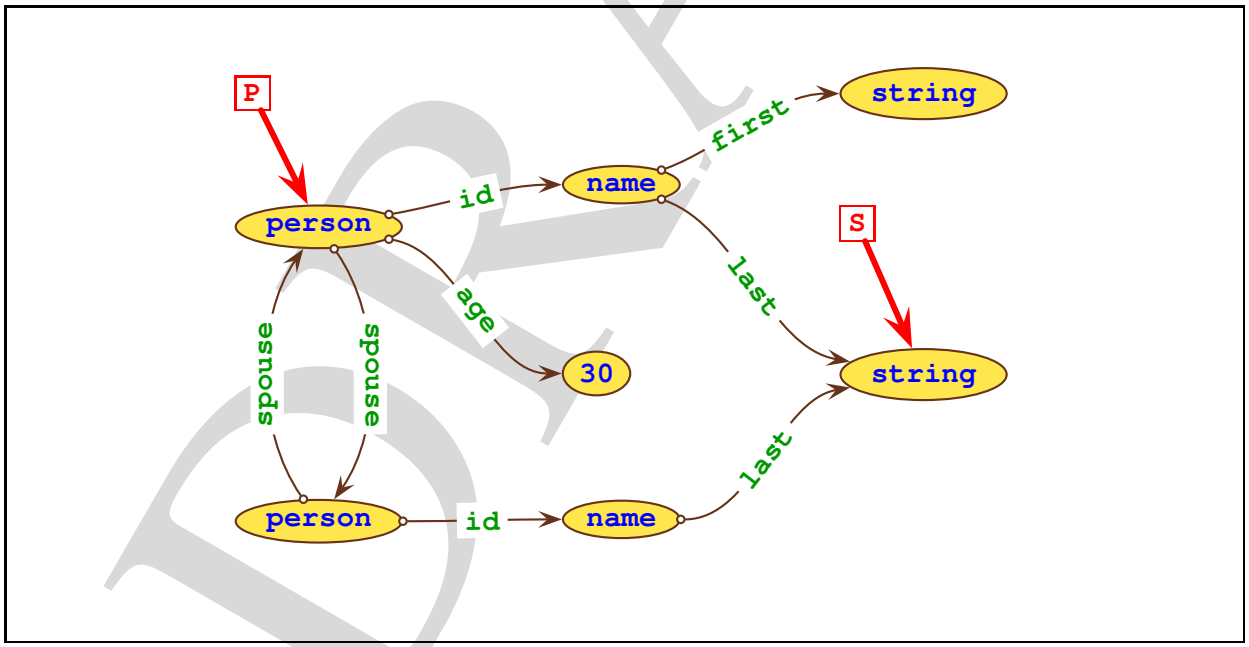


Figure 4.1: Example of $\mathcal{OSF}$ graph

This graph consists of a set of nodes and arcs between some nodes. This graph, however, is a *labeled* graph. There are two kinds of label symbols: one kind is for the nodes (*e.g.*, **person**, **name**, *etc.*), and the other is for the arcs. We call the symbols labeling nodes "*sorts*," and the

symbols labeling arcs "*features*" (*e.g.*, **spouse**, **age**, *etc.*). Intuitively, sort symbols denote sets, and feature symbols denote functions between these sets. We will identify value-denoting symbols such as **30** as sorts as well since they can be thought of as singleton sets — *i.e.*, in this case the set $\{30\}$ containing the single integer 30. We will also assume that sort symbols are partially ordered with a "*subsort*" relation "***is-a***" denoting set inclusion on the sets denoted by the sorts. This justifies calling such a graph an "*order-sorted feature*" graph; or, $\mathcal{OSF}$ graph for short.

In fact, an $\mathcal{OSF}$ graph can be defined as a rooted sorted *commutative diagram* of function compositions of the same nature as those used in mathematics. In other words, all feature-composition paths between two sort nodes commute.

The other labels such as ***P*** and ***S*** in the graph of Figure 4.1 are used as reference pointers to designate specific nodes — in this case the root node (*e.g.*, **P**) — or indicate an equational constraint among feature paths (*e.g.*, **P** and **S**). we shall call them *reference tags*. Formally, these will be assimilated to logical variables, *i.e.*, lexical references that could be consistently renamed in their context, without altering the meaning of the $\mathcal{OSF}$ graphs, terms, and constraints, in which they appear.

The above informal description of an $\mathcal{OSF}$ graph should make intuitive sense to anyone familiar with the kind of data structure used in object-oriented programming to represent typed object records. This is indeed a good way to think about it. It is this pragmatic understanding that we wish the reader to keep in mind. Our intention, however, is to go beyond merely *representing* and *using* structured types and objects: we not only want these to be convenient for computing, we also want them to be convenient for *reasoning* — while never losing this simple intuition. Thus, we follow a simple syntax to represent these graphs that will enable us to do so. For example, we represent the graph shown in Figure 4.1 using the syntax shown in Figure 4.2.

```
P : person(id → name(first → string,
                     last → S : string),
           age → 30,
           spouse → person(id → name(last → S),
                           spouse → P)).
```

Figure 4.2: $\mathcal{OSF}$ term syntax for the $\mathcal{OSF}$ graph of Figure 4.1

The reader with some knowledge of Logic Programming will have noted that this syntax generalizes that of Prolog terms — *i.e.*, of $\mathcal{FOT}$s. Indeed, a Prolog term can be seen as a restricted kind of $\mathcal{OSF}$ term, where sort symbols are data constructors; feature symbols are (implicit) subterm positions; and, reference tags are so-called "logical variables," and can only occur as leaves. It is to stress this fact that we call an expression like the one in Figure 4.2 an "$\mathcal{OSF}$ *term*." Note that, unlike a Prolog term where subterms are written following an implicit position order, an $\mathcal{OSF}$ term may be written up to permutation of its subterms since explicit feature labels allow specifying subterms in any order while still representing the same $\mathcal{OSF}$ graph. For example, the $\mathcal{OSF}$ term in Figure 4.3 could as well be used to represent the same $\mathcal{OSF}$ graph in Figure 4.1.

```
P : person(age → 30,
           spouse → person(spouse → P,
                           id → name(last → S : string)),
           id → name(last → S,
                     first → string)).
```

Figure 4.3: Equivalent $\mathcal{OSF}$ term syntax for the $\mathcal{OSF}$ graph of Figure 4.1

4.1.2 Formal background

In this section, we give a brief formal account of the notions illustrated in the foregoing informal description. The reader is referred to [23], [24], and [25] for all technical details such as mathematical proofs, and to [10] and [16] for more operational details such as implementation, as well as all further relevant references in them.

§ $\mathcal{OSF}$ Signature

An $\mathcal{OSF}$ *signature* is a quadruple $\langle \mathcal{S}, \preceq, \curlywedge, \mathcal{F} \rangle$ where:

- $\mathcal{S}$ is a set of *sorts* containing at least the two distinguished sorts $\top$ and $\bot$;
- $\preceq$ is a decidable partial order on $\mathcal{S}$ for which $\bot$ is unique least element and $\top$ is unique greatest element;
- $\langle \mathcal{S}, \preceq, \curlywedge \rangle$ is a lower semi-lattice ($s \curlywedge s'$ is called the greatest common subsort of s and s');
- $\mathcal{F}$ is a set of *feature symbols*.

Referring to the example in Figure 4.2, the set of sorts $\mathcal{S}$ contains set-denoting symbols such as **person**, **name**, and **string**. The set of features $\mathcal{F}$ contains function-denoting symbols (on the left of $\rightarrow$), such as **id**, **age**, **spouse**, **first**, **last**, *etc.*, ... The ordering on the sorts $\mathcal{S}$ denotes set inclusion and the infimum operation $\curlywedge$ denotes set intersection. Therefore, $\top$ denotes the all-inclusive sort (the set of all things), and $\bot$ denotes the all-exclusive sort (the set of no things). This is formalized next.

§ $\mathcal{OSF}$ Algebras

Given an $\mathcal{OSF}$ signature $\langle \mathcal{S}, \preceq, \curlywedge, \mathcal{F} \rangle$, an $\mathcal{OSF}$ *algebra* $\mathcal{A}$ is a mathematical structure consisting of a triplet:

$$\mathcal{A} \stackrel{\text{def}}{=} \langle D^{\mathcal{A}}, (s^{\mathcal{A}})_{s \in \mathcal{S}}, (f^{\mathcal{A}})_{f \in \mathcal{F}} \rangle \tag{4.1}$$

where:

- $D^{\mathcal{A}}$ is a non-empty set, called the *domain* of $\mathcal{A}$;
- for each sort symbol s in $\mathcal{S}$, $s^{\mathcal{A}}$ is a subset of the domain; in particular, $\top^{\mathcal{A}} = D^{\mathcal{A}}$ and $\bot^{\mathcal{A}} = \emptyset$;

- $(s_1 \curlywedge s_2)^{\mathcal{A}} = s_1^{\mathcal{A}} \cap s_2^{\mathcal{A}}$ for two sorts s_1 and s_2 in $\mathcal{S}$;
- for each feature f in $\mathcal{F}$, $f^{\mathcal{A}}$ is a total unary function from the domain into the domain; *i.e.*, $f^{\mathcal{A}} : D^{\mathcal{A}} \mapsto D^{\mathcal{A}}$.[3]

§ $\mathcal{OSF}$ Homomorphisms

In algebra, a homomorphism between two algebraic structures is a structure-preserving mapping. The essence of a structure-preserving mapping between $\mathcal{OSF}$ algebras is that it should preserve sort inclusion and feature application. Thus, an $\mathcal{OSF}$ *homomorphism* $\gamma : \mathcal{A} \mapsto \mathcal{B}$ between two $\mathcal{OSF}$ algebras $\mathcal{A}$ and $\mathcal{B}$ is a function $\gamma : D^{\mathcal{A}} \mapsto D^{\mathcal{B}}$ such that:

- $\forall s \in \mathcal{S},\ \ \gamma(s^{\mathcal{A}}) \ \subseteq s^{\mathcal{B}}$; and,
- $\forall f \in \mathcal{F}, \forall d \in D^{\mathcal{A}},\ \ \gamma(f^{\mathcal{A}}(d)) = f^{\mathcal{B}}(\gamma(d))$.

§ $\mathcal{OSF}$ Endomorphisms

The notion of interest for inheritance is that of $\mathcal{OSF}$ *endomorphism*. That is, when an $\mathcal{OSF}$ homomorphism γ is internal to an $\mathcal{OSF}$ algebra (*i.e.*, $\mathcal{A} = \mathcal{B}$), it is called an $\mathcal{OSF}$ endomorphism of $\mathcal{A}$. This means:

- $\forall s \in \mathcal{S},\ \ \gamma(s^{\mathcal{A}}) \ \subseteq s^{\mathcal{A}}$; and,
- $\forall f \in \mathcal{F}, \forall d \in D^{\mathcal{A}},\ \ \gamma(f^{\mathcal{A}}(d)) = f^{\mathcal{A}}(\gamma(d))$.

Such an endomorphism defines a natural pre-order $\preceq$ on the domain $D^{\mathcal{A}}$ of any $\mathcal{OSF}$ algebra $\mathcal{A}$ whereby $t_1 \preceq t_2$ iff $\exists \gamma : D^{\mathcal{A}} \to D^{A}$ such that $t_1 = \gamma(t_2)$. This is as pictured in Figure 4.4. As can be seen in this figure, interpreting a concept's attribute as a functional feature captures formally and precisely *inheritance of attributes* as used, *e.g.*, in object-oriented classes, semantic networks, and formal ontological logics defining *concept hierarchies*. Namely, a concept C_1 (the subconcept) inherits the attributes of a concept C_2 (its superconcept) *if and only if* there exists an $\mathcal{OSF}$ endormorphism (γ) taking the set s denoted by the superconcept C_2 to the set $\gamma(s)$ denoted by the subconcept C_1. Thus, for γ to be an $\mathcal{OSF}$ endomorphism means that, for any attribute f, the image concept $C_1.f$ is a subconcept of the image concept $C_2.f$. Formally, this translates as the equality $\gamma(f(s)) \ = \ f(\gamma(s))$. In other words, feature application and sort refinement commute: doing first the former and then the latter or *vice versa* always yields the same result.

Note the contravariance of the "`is a`" arrows on concepts (subsets of denotations) *vs.* the endomorphic maps between the $\mathcal{OSF}$ graphs denoting them. The latter go from the tag set (*i.e.*, the set of nodes) of the more general $\mathcal{OSF}$ graph to that of the less general graph.[4] This is understandable since the more general $\mathcal{OSF}$ graph has always at least as many feature arcs as the less general graph. It is then said that such an endomorphic mapping γ *realizes* the inheritance between the two $\mathcal{OSF}$ graphs corresponding to the less general graph's feature domains and ranges.

[3] See Appendix Section B.1 for partial features, and other extensions.

[4] In other words, the more constrained the graphs, the lesser the extent of their denotations. This duality phenomenon between constraints and solutions is well-known and can be paraphrased as, "*the more there are constraints, the less there are solutions.*"

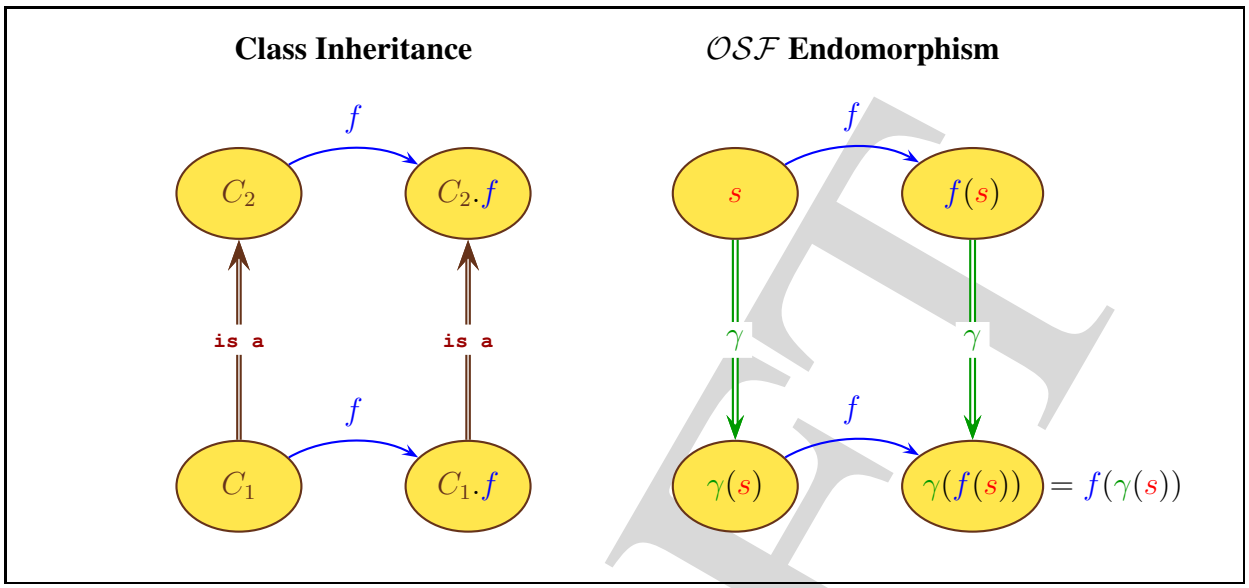


Figure 4.4: Subclass functional-attribute inheritance as $\mathcal{OSF}$ endomorphism

§ Syntax of $\mathcal{OSF}$ Terms

Definition 4.1 ($\mathcal{OSF}$ Term Syntax) *An $\mathcal{OSF}$ term t is an expression taking one of the three possible syntactic forms:*

1. *an unsorted variable X; or,*
2. *a simply sorted variable $X : s$; or,*
3. *an attributed sorted variable $X : s(f_1 \to t_1, \ldots, f_n \to t_n)$;*

where X is an element in a countably infinite set of variables $\mathcal{V}$, s is a sort in $\mathcal{S}$ and, $f_1, \ldots, f_n$ are features in $\mathcal{F}$ and $t_1, \ldots, t_n$ are $\mathcal{OSF}$ terms, for any $n \geq 1$.

Examples of this $\mathcal{OSF}$ term syntax definition are those displayed in Figure 4.2 and Figure 4.3. For such an $\mathcal{OSF}$ term t, the variable X is called the *root* tag of t, and is referred to as $\textbf{ROOT}(t)$. In what follows, we shall consider "*(logical) variable*" and "*($\mathcal{OSF}$) tag*" to be synonymous: a logical variable points to the root of an $\mathcal{OSF}$ term as a reference tag to an $\mathcal{OSF}$ graph data structure. The set of all tags occurring in an $\mathcal{OSF}$ term t is defined as $\textbf{\textit{Tags}}(t) \stackrel{\text{def}}{=} \{\, \textbf{ROOT}(t) \,\} \cup \bigcup_{i=1}^{n} \textbf{\textit{Tags}}(t_i)$. As justified by the following semantics given to this syntax, we shall consider the first form for an unsorted variable X that occurs nowhere else in its context as a sorted variable, as a shorthand for $X : \top$.

§ Semantics of $\mathcal{OSF}$ Terms

Given that $\mathcal{OSF}$ term syntax uses variables from a countably infinite set of variables $\mathcal{V}$, we must define the meaning of a term for all possible values these variables may take in the intended semantic domain.

In the remainder, we assume given an $\mathcal{OSF}$ signature $\langle \mathcal{S}, \preceq, \curlywedge, \mathcal{F} \rangle$, and by $\mathcal{OSF}$ algebra $\mathcal{A}$, we shall mean an $\mathcal{OSF}$ algebra $\mathcal{A} = \langle D^{\mathcal{A}}, (s^{\mathcal{A}})_{s \in \mathcal{S}}, (f^{\mathcal{A}})_{f \in \mathcal{F}} \rangle$ on this reference signature.

For such an $\mathcal{OSF}$ algebra $\mathcal{A}$ and a specific $\mathcal{A}$-valuation $\upsilon : \mathcal{V} \mapsto D^{\mathcal{A}}$, the *$\mathcal{A}$-denotation* $[\![t]\!]^{\mathcal{A},\upsilon}$ of an $\mathcal{OSF}$ term t such as above is given by:

$$[\![t]\!]^{\mathcal{A},\upsilon} \stackrel{\text{def}}{=} \{\, \upsilon(X) \,\} \cap s^{\mathcal{A}} \cap \bigcap_{1 \leq i \leq n} (f_i^{\mathcal{A}})^{-1}([\![t_i]\!]^{\mathcal{A},\upsilon}). \tag{4.2}$$

Hence, for a fixed $\mathcal{A}$-valuation υ, $[\![t]\!]^{\mathcal{A},\upsilon}$ is either the empty set or the singleton set $\{\, \upsilon(\mathbf{ROOT}(t)) \,\}$. In fact, it is *not* the empty set if and only if the value $\upsilon(\mathbf{ROOT}(t))$ lies in the denotation of the sort s, as well as every inverse image by $(f_i^{\mathcal{A}})^{-1}$ (the reciprocal of the denotation $f_i^{\mathcal{A}}$ of each feature f_i) of the denotation of the corresponding subterm $[\![t_i]\!]^{\mathcal{A},\upsilon}$ *under the same $\mathcal{A}$-valuation* υ. Thus, the denotation of an $\mathcal{OSF}$ term t for all possible valuations of the variables is given by the set:

$$[\![t]\!]^{\mathcal{A}} \stackrel{\text{def}}{=} \bigcup_{\upsilon : \mathcal{V} \mapsto D^{\mathcal{A}}} [\![t]\!]^{\mathcal{A},\upsilon}. \tag{4.3}$$

§ $\mathcal{OSF}$ Term Subsumption

Since we have both a formal syntax and its formal semantics, there are two ways we can define $\mathcal{OSF}$ term subsumption between two $\mathcal{OSF}$ terms t and t'.

Definition 4.2 (Syntactic $\mathcal{OSF}$ Term Subsumption) *An $\mathcal{OSF}$ term t is syntactically subsumed by an $\mathcal{OSF}$ term t' (written: $t \preceq t'$) if, and only if, there exists an $\mathcal{OSF}$ endomorphism γ such that $t = \gamma(t')$.*

Definition 4.3 (Semantic $\mathcal{OSF}$ Term Subsumption) *An $\mathcal{OSF}$ term t is semantically subsumed by an $\mathcal{OSF}$ term t' (written: $t [\![\preceq]\!] t'$) if, and only if, $[\![t]\!]^{\mathcal{A}} \subseteq [\![t']\!]^{\mathcal{A}}$ in any interpretation $\mathcal{OSF}$ algebra $\mathcal{A}$.*

The following theorem was established in [23] and states that the two definitions coincide.

Theorem 4.1 ($\mathcal{OSF}$ Term Subsumption) *For any $\mathcal{OSF}$ terms t and t',*

$$t \preceq t' \quad \textit{iff} \quad t [\![\preceq]\!] t'.$$

§ $\mathcal{OSF}$ Term Normal Form

An $\mathcal{OSF}$ term $t = X : s(f_1 \rightarrow t_1, \ldots, f_n \rightarrow t_n)$ is said to be "*in normal form*" whenever all the following properties hold:

- $s \in \mathcal{S} \setminus \{\bot\}$ (*i.e.*, s is a non-bottom sort in $\mathcal{S}$);
- $f_1, \ldots, f_n$ are pairwise distinct features in $\mathcal{F}$, for all $n \geq 2$;
- $t_1, \ldots, t_n$ are all $\mathcal{OSF}$ terms in *normal* form, for all $n \geq 1$; and,
- no variable in $\textbf{\textit{Tags}}(t)$ is sorted more than once by a non-top sort.[5]

[5] That is, if $X \in \textbf{\textit{Tags}}(t)$ occurs in t both as $X : s$ and $X : s'$, then $s = \top$ or $s' = \top$.

With these criteria, we note hence that all the examples of $\mathcal{OSF}$ terms shown in the previous section are, formally speaking, in normal form. Such a normal form ensures that it is devoid of actual or potential inconsistencies. We shall call an $\mathcal{OSF}$ term in normal form a "ψ-*term*," and designate as Ψ the set of all ψ-terms.[6] How to *normalize* an $\mathcal{OSF}$ term into a semantically equivalent ψ-term is explained next.

§ From $\mathcal{OSF}$ Terms to $\mathcal{OSF}$ Constraints

A logical reading of an $\mathcal{OSF}$ term is immediate as its information content can be characterized by a simple formula. For this purpose, we need a simple clausal language as follows.

An atomic $\mathcal{OSF}$ *constraint* is one of (1) $X : s$, (2) $X \doteq X'$, or (3) $X.f \doteq X'$, where X and X' are variables in $\mathcal{V}$, s is a sort in $\mathcal{S}$, and f is a feature in $\mathcal{F}$. A (conjunctive) $\mathcal{OSF}$ *constraint* is a conjunction (*i.e.*, a set) of atomic $\mathcal{OSF}$ constraints $\phi_1 \ \& \ \ldots \ \& \ \phi_n$. Given an $\mathcal{OSF}$ algebra $\mathcal{A}$, we say that an $\mathcal{OSF}$ constraint ϕ is *satisfiable* in $\mathcal{A}$ with a valuation $\upsilon : \mathcal{V} \mapsto D^{\mathcal{A}}$ (and we write this as "$\mathcal{A}, \upsilon \models \phi$") whenever:

$$\begin{array}{lll} \mathcal{A}, \upsilon \models X : s & \textbf{iff} & \upsilon(X) \in s^{\mathcal{A}}; \\ \mathcal{A}, \upsilon \models X \doteq Y & \textbf{iff} & \upsilon(X) = \upsilon(Y); \\ \mathcal{A}, \upsilon \models X.f \doteq Y & \textbf{iff} & f^{\mathcal{A}}(\upsilon(X)) = \upsilon(Y); \\ \mathcal{A}, \upsilon \models \phi \ \& \ \phi' & \textbf{iff} & \mathcal{A}, \upsilon \models \phi \ \textbf{and} \ \mathcal{A}, \upsilon \models \phi'. \end{array} \tag{4.4}$$

Given an $\mathcal{OSF}$ term $t = X : s(f_1 \to t_1, \ldots, f_n \to t_n)$, we can always associate to it a specific corresponding $\mathcal{OSF}$ constraint $\varphi(t)$ as follows:

$$\begin{array}{lcllcllcl} \varphi(t) & \stackrel{\text{def}}{=} & X : s & \& & X.f_1 \doteq \textbf{ROOT}(t_1) & \& & \ldots & \& & X.f_n \doteq \textbf{ROOT}(t_n) \\ & & & \& & \varphi(t_1) & & & \& & \varphi(t_n) \end{array} \tag{4.5}$$

for any $n \geq 0$. We say that $\varphi(t)$ is obtained from the $\mathcal{OSF}$ term t by *dissolving* it into its clausal form. The following theorem, also established in [23], states that the algebraic denotation of an $\mathcal{OSF}$ term as a set and the logical semantics of its dissolved form obtained by Expression (4.5) coincide exactly.

Theorem 4.2 (Coinciding Algebraic and Logical Semantics of $\mathcal{OSF}$ Terms)

$$[\![t]\!]^{\mathcal{A}} \ = \ \{ \, \upsilon(\textbf{ROOT}(t)) \ \mid \ \upsilon : \mathcal{V} \to D^{\mathcal{A}} \textit{ and } \mathcal{A}, \upsilon \ \models \ \varphi(t) \, \}.$$

4.1.3 $\mathcal{OSF}$-term lattice structure

As seen in the two previous sections, the $\mathcal{FOT}$ can be extended and made more expressive as a knowledge structure into a rooted $\mathcal{OSF}$ graph written as an $\mathcal{OSF}$ term. It is more expressive as a model of typed data and their types, as well as approximations thereof as generic set-denoting

[6] The expression "ψ-term" was introduced originally by the first author in his PhD thesis [4] as a shorthand for "*Property Structure Inheritance Term*," as a formalization of some parts of Ron Brachman's "*Structured Inheritance Networks*" (or "*SI-Nets*") defined informally in his PhD thesis [53].

concepts ordered by set inclusion. Formally, it is a representation scheme that is a conservative extension of, yet more generic than, the one offered by $\mathcal{FOT}$s and their lattice-theoretic properties exposed in 1970 simultaneously, though independently, by Reynolds in [141] and Plotkin in [134] and [135]. A more general term lattice was first defined in [3]; then, in [4] and [6], it was further extended to $\mathcal{OSF}$ structures, which express order-sorted functional commutative diagrams among sets as used in mathematics to express equational constraints on functional feature compositions. These are just a special case of ψ-terms and their $\mathcal{OSF}$ lattice operations as made explicit in [8].[7] These operations (*viz.*, unification and generalization) can thus be extended from $\mathcal{FOT}$s to $\mathcal{OSF}$ terms, obtaining improved expressivity while keeping the same complexity,[8] to provide a simple and intuitive semantics for knowledge and data structures of partial descriptions.

The subsumption ordering defined on ψ-terms is an extension of the subsumption ordering on $\mathcal{FOT}$s. Specifically, for an $\mathcal{OSF}$ term t to subsume an $\mathcal{OSF}$ term t' (*i.e.*, $t' \preceq t$), there must be a mapping γ from the tags of t to the tags of t' — *i.e.*, $\gamma : \boldsymbol{Tags}(t) \mapsto \boldsymbol{Tags}(t')$ — that respects sorting and preserves feature applications. Such a mapping is called a *sort-consistent feature-preserving endomorphic mapping* and is the formal notion extending the concept of $\mathcal{FOT}$ substitution to ψ-terms. To emphasize the existence of this mapping γ whenever $t' \preceq t$, we shall also write $t' = \gamma(t)$.

More precisely, $\mathcal{OSF}$ endomorphic maps define a partial order on $\mathcal{OSF}$ terms "up to tag renaming" just as $\mathcal{FOT}$s are partially ordered by variable instantiation "up to variable renaming." It is easy to verify that it is a partial order on $\mathcal{OSF}$ terms modulo tag renaming. Indeed, it is reflexive (since the identity on $\boldsymbol{Tags}(t)$ is a tag map for any $\mathcal{OSF}$ term t); it is transitive (by composition of tag maps); and is symmetric since $t \preceq t'$ and $t' \preceq t$ is by definition the same as $\exists\gamma : \boldsymbol{Tags}(t) \to \boldsymbol{Tags}(t')$ s.t. $\gamma(t) = t'$ and $\exists\gamma' : \boldsymbol{Tags}(t') \to \boldsymbol{Tags}(t)$ s.t. $\gamma'(t') = t$ where γ and γ' are inverse bijections on $\mathcal{V}$; that is, inverse tag renamings. Therefore, this is equality on the equivalence class of $\mathcal{OSF}$ terms modulo tag renaming. Given two $\mathcal{OSF}$ terms t and t' such that $t \preceq t'$ with the endomorphic map $\gamma : \boldsymbol{Tags}(t) \mapsto \boldsymbol{Tags}(t')$, the application of an endomorphic map to $\mathcal{OSF}$ clauses is defined inductively as follows:

$$\left.\begin{array}{rcl} \gamma(X : s) & \stackrel{\text{def}}{=} & \gamma(X) : s \\ \gamma(X.f \doteq Y) & \stackrel{\text{def}}{=} & \gamma(X).f \doteq \gamma(Y) \\ \gamma(X \doteq Y) & \stackrel{\text{def}}{=} & \gamma(X) \doteq \gamma(Y) \\ \gamma(\phi_1 \ \& \ \phi_2) & \stackrel{\text{def}}{=} & \gamma(\phi_1) \ \& \ \gamma(\phi_2) \end{array}\right\} \tag{4.6}$$

In practice, using this definition, such a mapping γ associates each tag symbol occurring in a ψ-term t to a tag symbol occurring in a ψ-term $t' = \gamma(t)$ in such as way that, whenever:

$$\{\, X : d, X.f \doteq Y, Y : r \,\} \ \subseteq \ \varphi(t),$$

then necessarily, for some sorts d' and r' s.t. $d' \preceq d$ and $r' \preceq r$:

$$\{\, \gamma(X) : d', \gamma(X).f \doteq \gamma(Y), \gamma(Y) : r' \,\} \ \subseteq \ \varphi(t') = \varphi(\gamma(t)) = \gamma(\varphi(t)).$$

This, as pictured in the diagram in Figure 4.4 shown above, captures formally attribute inheritance as used in object-oriented programming and knowledge representation, with the added bonus of

[7] *Op. cit.*, Pages 47–60, Section 3.1 on order-sorted feature constraints.

[8] In fact, sorting focuses, and thus optimizes, reasoning [19].

providing an effective operational method for performing *conjunctive deduction* (*viz.*, unification) and *disjunctive induction* (*viz.*, generalization) using such structures. Indeed, as shown in Figure 4.5, endomorphic mappings define lattice operations on ψ-terms whose sets of tags have been consistently renamed apart in exactly the same way as "renaming substitutions" do for $\mathcal{FOT}$s.[9]

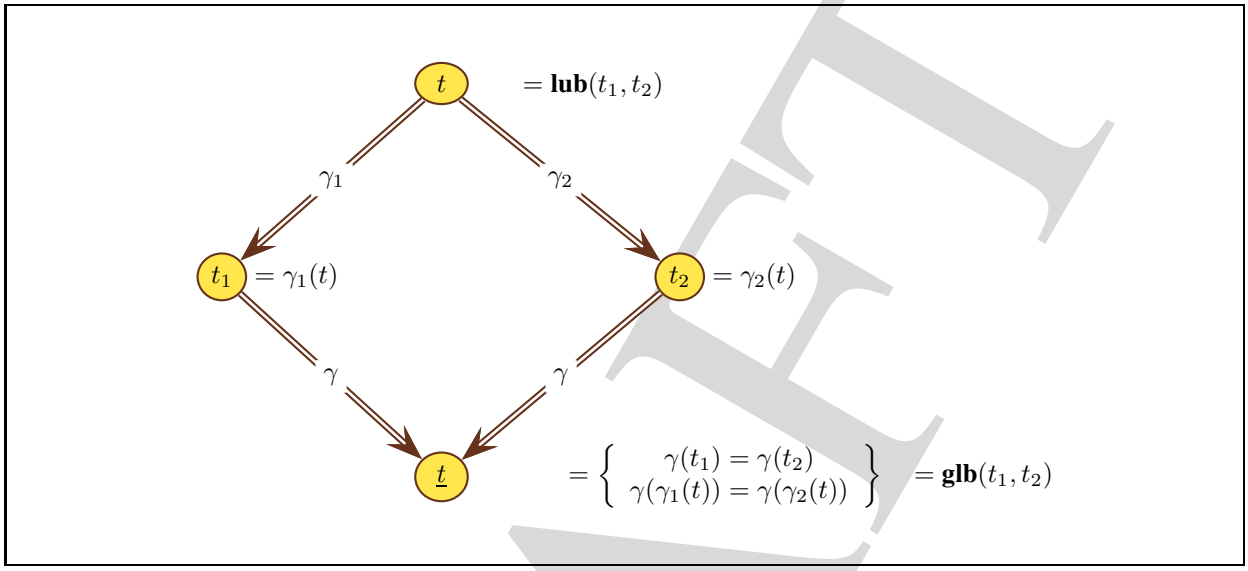


Figure 4.5: $\mathcal{OSF}$ subsumption lattice operations

Just like $\mathcal{FOT}$s with substitutions, whenever two ψ-terms t and t' are such that $\exists\gamma$ s.t. $t' = \gamma(t)$ and $\exists\gamma'$ s.t. $t = \gamma'(t')$, they are said to be *identical "up to tag renaming."* This is because necessarily $\gamma' = \gamma^{-1}$ and $\gamma = \gamma'^{-1}$ make up a pair of mutually inverse "*one-to-one onto*" tag mappings (bijections). How to compute these endomorphic mappings by $\mathcal{OSF}$ constraint normalization is shown next.

§ $\mathcal{OSF}$ Unification: Deduction by Constraint Normalization

Definition 4.4 (Solved $\mathcal{OSF}$ Constraint) *An $\mathcal{OSF}$ constraint ϕ is said to be in solved form if for every variable X, ϕ contains:*

- *at most one sort constraint $X : s$ and $s \neq \bot$, for any X that occurs in ϕ;*
- *at most one feature constraint $X.f \doteq Y$, for any $X.f$ that occurs in ϕ;*

and whenever $X \doteq Y \in \phi$, then X does not appear anywhere else in ϕ.

As explained in [23], given an $\mathcal{OSF}$ constraint ϕ, non-deterministically applying any applicable rule among the rules shown in Figure 4.6 until none applies always terminates either in the inconsistent constraint $\mathfrak{false}$ or in a solved $\mathcal{OSF}$ constraint. The notation $\phi[X/Y]$ in Rule **Tag Elimination** stands for the constraint ϕ in which all occurrences of the tag Y have been replaced with the tag X. As a result of applying this rule, the tag being eliminated (Y) will occur nowhere else in the normal form except as the right-hand side of constraint $X \doteq Y$. In [23], it is formally established that the rules in Figure 4.6 are:

[9] See Figure 3.1 in Section 3.4.

SORT INTERSECTION

$$\frac{\phi \;\&\; X : s \;\&\; X : s'}{\phi \;\&\; X : s \curlywedge s'}$$

FEATURE FUNCTIONALITY

$$\frac{\phi \;\&\; X.f \doteq Y \;\&\; X.f \doteq Y'}{\phi \;\&\; X.f \doteq Y \;\&\; Y \doteq Y'}$$

INCONSISTENT SORT

$$\frac{\phi \;\&\; X : \bot}{\mathfrak{false}}$$

TAG ELIMINATION

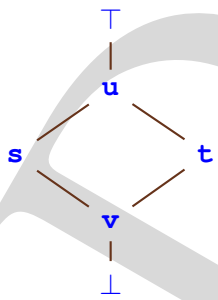
$$\frac{[Y \in \textit{Tags}(\phi)] \qquad \phi \;\&\; X \doteq Y}{\phi[X/Y] \;\&\; X \doteq Y}$$

Figure 4.6: Constraint normalization rules for $\mathcal{OSF}$ unification

1. *solution-preserving* — for each rule, the set of solutions of the posterior constraint is equal to the set of solutions of the prior constraint;
2. *finitely terminating* — normalizing with these rules always terminates after a finite number of formula transformations;
3. *confluent* — they always end up with the same constraint up to consistent tag renaming.

Furthermore, they always result in a normal form that is either the inconsistent constraint $\mathfrak{false}$ or a consistent $\mathcal{OSF}$ constraint in solved form. These rules are all we need to perform the unification of two ψ-terms. Namely, two ψ-terms t_1 and t_2 are unifiable if and only if the normal form of the $\mathcal{OSF}$ constraint $\mathbf{ROOT}(t_1) \doteq \mathbf{ROOT}(t_2) \;\&\; \varphi(t_1) \;\&\; \varphi(t_2)$ is not $\mathfrak{false}$.

Example 4.1 $\mathcal{OSF}$ **signature and terms** — Let us consider the sort signature:



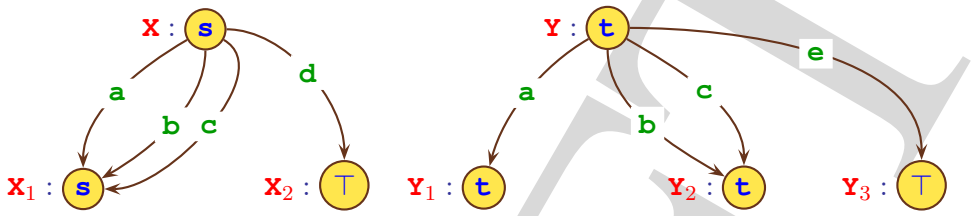
and the two ψ-terms:

$$\psi_1 \stackrel{\text{def}}{=} \mathtt{X} : \mathtt{s}\,(\,\mathtt{a} \rightarrow \mathtt{X}_1 : \mathtt{s},\ \mathtt{b} \rightarrow \mathtt{X}_1,\ \mathtt{c} \rightarrow \mathtt{X}_1,\ \mathtt{d} \rightarrow \mathtt{X}_2\,)$$

and:

$$\psi_2 \stackrel{\text{def}}{=} \mathtt{Y} : \mathtt{t}\,(\,\mathtt{a} \rightarrow \mathtt{Y}_1 : \mathtt{t},\ \mathtt{b} \rightarrow \mathtt{Y}_2 : \mathtt{t},\ \mathtt{c} \rightarrow \mathtt{Y}_2,\ \mathtt{e} \rightarrow \mathtt{Y}_3\,)$$

corresponding to the following two $\mathcal{OSF}$ graphs:



and corresponding dissolved forms:

$$\begin{aligned}\varphi(\psi_1) = \ & \mathtt{X} : \mathtt{s} \ \&\ \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \ \&\ \mathtt{X}_1 : \mathtt{s} \\ & \&\ \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ & \&\ \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ & \&\ \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \ \&\ \mathtt{X}_2 : \top\end{aligned} \qquad \begin{aligned}\varphi(\psi_2) = \ & \mathtt{Y} : \mathtt{t} \ \&\ \mathtt{Y}.\mathtt{a} \doteq \mathtt{Y}_1 \ \&\ \mathtt{Y}_1 : \mathtt{t} \\ & \&\ \mathtt{Y}.\mathtt{b} \doteq \mathtt{Y}_2 \ \&\ \mathtt{Y}_2 : \mathtt{t} \\ & \&\ \mathtt{Y}.\mathtt{c} \doteq \mathtt{Y}_2 \\ & \&\ \mathtt{Y}.\mathtt{e} \doteq \mathtt{Y}_3 \ \&\ \mathtt{Y}_3 : \top.\end{aligned}$$

Example 4.2 $\mathcal{OSF}$ **term unification** — Using the dissolved forms of the terms ψ_1 and ψ_2 defined in Example 4.1, let us normalize the clause formed as:

$$\textsc{root}(\psi_1) \doteq \textsc{root}(\psi_2) \ \&\ \varphi(\psi_1) \ \&\ \varphi(\psi_2);$$

namely,

$$\begin{aligned}\mathtt{X} \doteq \mathtt{Y} \ \&\ \ & \mathtt{X} : \mathtt{s} \ \&\ \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \ \&\ \mathtt{X}_1 : \mathtt{s} \\ & \&\ \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ & \&\ \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ & \&\ \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \ \&\ \mathtt{X}_2 : \top\end{aligned} \ \&\ \begin{aligned}& \mathtt{Y} : \mathtt{t} \ \&\ \mathtt{Y}.\mathtt{a} \doteq \mathtt{Y}_1 \ \&\ \mathtt{Y}_1 : \mathtt{t} \\ & \&\ \mathtt{Y}.\mathtt{b} \doteq \mathtt{Y}_2 \ \&\ \mathtt{Y}_2 : \mathtt{t} \\ & \&\ \mathtt{Y}.\mathtt{c} \doteq \mathtt{Y}_2 \\ & \&\ \mathtt{Y}.\mathtt{e} \doteq \mathtt{Y}_3 \ \&\ \mathtt{Y}_3 : \top\end{aligned}$$

with any applicable rule in Figure 4.6:

1. apply Rule **Tag Elimination**:[10]

$$\begin{aligned}\underline{\mathtt{X} \doteq \mathtt{Y}} \ \&\ \ & \mathtt{X} : \mathtt{s} \ \&\ \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \ \&\ \mathtt{X}_1 : \mathtt{s} \\ & \&\ \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ & \&\ \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ & \&\ \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \ \&\ \mathtt{X}_2 : \top\end{aligned} \ \&\ \begin{aligned}& \mathtt{Y} : \mathtt{t} \ \&\ \mathtt{Y}.\mathtt{a} \doteq \mathtt{Y}_1 \ \&\ \mathtt{Y}_1 : \mathtt{t} \\ & \&\ \mathtt{Y}.\mathtt{b} \doteq \mathtt{Y}_2 \ \&\ \mathtt{Y}_2 : \mathtt{t} \\ & \&\ \mathtt{Y}.\mathtt{c} \doteq \mathtt{Y}_2 \\ & \&\ \mathtt{Y}.\mathtt{e} \doteq \mathtt{Y}_3 \ \&\ \mathtt{Y}_3 : \top\end{aligned}$$

[10] We underline the pattern being normalized in the conjunction of constraints. Since normalizing the conjuncts in any order will terminate in the same normal form modulo tag renaming, we shall proceed in a left-to-right order. We shall also move all eliminated-tag renamings to the end of the constraint as they are *de facto* in normal form and eventually provide the substitution resolving the original constraint to normalize.

2. apply Rule **SORT INTERSECTION**:

$$\begin{array}{l} \underline{\mathtt{X} : \mathtt{s}} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \mathtt{X}_1 : \mathtt{s} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top \end{array} \quad \& \quad \begin{array}{l} \underline{\mathtt{X} : \mathtt{t}} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{Y}_1 \;\&\; \mathtt{Y}_1 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{Y}_2 \;\&\; \mathtt{Y}_2 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{Y}_2 \\ \qquad \&\; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top \end{array}$$
$$\&\; \mathtt{X} \doteq \mathtt{Y}$$

3. apply Rule **FEATURE FUNCTIONALITY**:

$$\begin{array}{l} \mathtt{X} : \mathtt{v} \;\&\; \underline{\mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1} \;\&\; \mathtt{X}_1 : \mathtt{s} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top \end{array} \quad \begin{array}{l} \underline{\mathtt{X}.\mathtt{a} \doteq \mathtt{Y}_1} \;\&\; \mathtt{Y}_1 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{Y}_2 \;\&\; \mathtt{Y}_2 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{Y}_2 \\ \qquad \&\; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top \end{array}$$
$$\&\; \mathtt{X} \doteq \mathtt{Y}$$

4. apply Rule **TAG ELIMINATION**:

$$\begin{array}{l} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \mathtt{X}_1 : \mathtt{s} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top \end{array} \quad \begin{array}{l} \underline{\mathtt{X}_1 \doteq \mathtt{Y}_1} \;\&\; \mathtt{Y}_1 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{Y}_2 \;\&\; \mathtt{Y}_2 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{Y}_2 \\ \qquad \&\; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top \end{array}$$
$$\&\; \mathtt{X} \doteq \mathtt{Y}$$

5. apply Rule **SORT INTERSECTION**:

$$\begin{array}{l} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \underline{\mathtt{X}_1 : \mathtt{s}} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top \end{array} \quad \begin{array}{l} \underline{\mathtt{X}_1 : \mathtt{t}} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{Y}_2 \;\&\; \mathtt{Y}_2 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{Y}_2 \\ \qquad \&\; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top \end{array}$$
$$\&\; \mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1$$

6. apply Rule **FEATURE FUNCTIONALITY**:

$$\begin{array}{l} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \mathtt{X}_1 : \mathtt{v} \\ \qquad \&\; \underline{\mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top \end{array} \quad \begin{array}{l} \\ \qquad \&\; \underline{\mathtt{X}.\mathtt{b} \doteq \mathtt{Y}_2} \;\&\; \mathtt{Y}_2 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{Y}_2 \\ \qquad \&\; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top \end{array}$$
$$\&\; \mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1$$

7. apply Rule **TAG ELIMINATION**:

$$\begin{array}{l} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \mathtt{X}_1 : \mathtt{v} \\ \qquad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 \\ \qquad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top \end{array} \quad \begin{array}{l} \\ \qquad \&\; \underline{\mathtt{X}_1 \doteq \mathtt{Y}_2} \;\&\; \mathtt{Y}_2 : \mathtt{t} \\ \qquad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{Y}_2 \\ \qquad \&\; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top \end{array}$$
$$\&\; \mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1$$

8. apply Rule **SORT INTERSECTION**:

$$\begin{array}{lll} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \underline{\mathtt{X}_1 : \mathtt{v}} & & \\ \quad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 & \& & \underline{\mathtt{X}_1 : \mathtt{t}} \\ \quad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 & \& \; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 & \\ \quad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top & \& \; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top & \\ \;\&\; \mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1 & \& \; \mathtt{X}_1 \doteq \mathtt{Y}_2 & \end{array}$$

9. apply Rule **FEATURE FUNCTIONALITY**:[11]

$$\begin{array}{lll} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \mathtt{X}_1 : \mathtt{v} & & \\ \quad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 & & \\ \quad \&\; \underline{\mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1} & \& \; \underline{\mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1} & \\ \quad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top & \& \; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top & \\ \;\&\; \mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1 & \& \; \mathtt{X}_1 \doteq \mathtt{Y}_2 & \end{array}$$

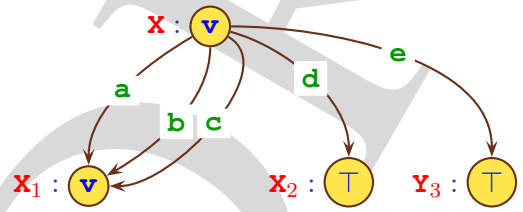
10. and the normalization terminates, resulting in the following simplified normal form:[12]

$$\begin{array}{lll} \mathtt{X} : \mathtt{v} \;\&\; \mathtt{X}.\mathtt{a} \doteq \mathtt{X}_1 \;\&\; \mathtt{X}_1 : \mathtt{v} & & \\ \quad \&\; \mathtt{X}.\mathtt{b} \doteq \mathtt{X}_1 & & \\ \quad \&\; \mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1 & & \\ \quad \&\; \mathtt{X}.\mathtt{d} \doteq \mathtt{X}_2 \;\&\; \mathtt{X}_2 : \top & \& \; \mathtt{X}.\mathtt{e} \doteq \mathtt{Y}_3 \;\&\; \mathtt{Y}_3 : \top & \\ \;\&\; \mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1 & \& \; \mathtt{X}_1 \doteq \mathtt{Y}_2 & \end{array}$$

This normal form corresponds to the ψ-term:

$$\psi_1 \wedge \psi_2 \stackrel{\text{def}}{=} \mathtt{X} : \mathtt{v}\,(\,\mathtt{a} \to \mathtt{X}_1 : \mathtt{v},\, \mathtt{b} \to \mathtt{X}_1,\, \mathtt{c} \to \mathtt{X}_1,\, \mathtt{d} \to \mathtt{X}_2,\, \mathtt{e} \to \mathtt{Y}_3\,) \tag{4.7}$$

that is the $\mathcal{OSF}$-graph:



(4.8)

resulting from eliminating tags with the tag substitutions: $\mathtt{X}/\mathtt{Y}$, $\mathtt{X}_1/\mathtt{Y}_1$, and $\mathtt{X}_1/\mathtt{Y}_2$ corresponding to the tag-renaming constraints left by Rule **TAG ELIMINATION** that remain in the normal form; *viz.*, $\mathtt{X} \doteq \mathtt{Y} \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_1 \;\&\; \mathtt{X}_1 \doteq \mathtt{Y}_2$.

[11] Note that instead of using Rule **FEATURE FUNCTIONALITY** as done here, this step could simply be logical simplification (using idempotence of logical conjunction: $\phi \;\&\; \phi \to \phi$ for any formula ϕ, taking $\mathtt{X}.\mathtt{c} \doteq \mathtt{X}_1$ for ϕ in this case). Of course, confluence of normalization garantees that both are correct and lead to termination.

[12] Using logical simplification, whereby $\mathtt{T} \doteq \mathtt{T} \to \mathit{true}$ for any tag $\mathtt{T}$ (in this case $\mathtt{X}_1 \doteq \mathtt{X}_1 \to \mathit{true}$); and, $\phi \;\&\; \mathit{true} \to \phi$, for any formula ϕ.

Well-scoped $\mathcal{OSF}$ term unification In order to respect the specific scope of a given ψ-term, we shall always assume each argument of the **glb** and **lub** lattice operations on two given ψ-terms to have distinct independent tag scopes before dissolution into their respective $\mathcal{OSF}$-clause forms. This is equivalent to "renaming apart" in $\mathcal{FOT}$-term lattice operations. This enables specifying these operations with tag mappings between two distinct tag sets for their domain and range, introducing new variables in the **glb** and **lub** as $\mathcal{OSF}$-clause unification or generalization proceeds.

Note that the constraint normalization rules of Figure 4.6 on Page 85 compute a tag binding (as specified by Rule TAG ELIMINATION) that replaces any tag by its class representative. This corresponds to computing the **glb** over ψ-terms shown in Figure 4.5 on Page 84 *up to consistent tag renaming*, whereby the lattice is in fact the quotient set of ψ-terms modulo consistent tag renaming. Thus, it is assumed that the ψ-term resulting from the $\mathcal{OSF}$ unification operation is obtained using a scoped endomorphism γ; *i.e.*, from scoped ψ-terms ψ_1 and ψ_2 to scoped ψ-term $\psi_1 \wedge \psi_2$ such that $\gamma : \boldsymbol{Tags}(\psi_1) \cup \boldsymbol{Tags}(\psi_2) \rightarrow \boldsymbol{Tags}(\psi_1 \wedge \psi_2)$.[13]

Producing a well-scoped tag endomorphism from the normal form obtained by normalizing the $\mathcal{OSF}$ constraint:

$$\mathbf{ROOT}(t) \doteq \mathbf{ROOT}(t') \;\;\&\;\; \varphi(t) \;\;\&\;\; \varphi(t') \tag{4.9}$$

where t and t' are two ψ-terms such that $\boldsymbol{Tags}(t) \cap \boldsymbol{Tags}(t') = \emptyset$ (*i.e.*, whose respective tags have been renamed apart), is done as follows:

1. let $\Pi \stackrel{\text{def}}{=} \{T_1, \ldots, T_m\}$, $0 \leq m \leq n$, $n \stackrel{\text{def}}{=} |\boldsymbol{Tags}(t) \cup \boldsymbol{Tags}(t')|$, be the resulting partition of $\boldsymbol{Tags}(t) \cup \boldsymbol{Tags}(t')$; that is:
   - for each $i = 1, \ldots, m$, $T_i \subseteq \boldsymbol{Tags}(t) \cup \boldsymbol{Tags}(t')$;
   - for all i, j s.t. $1 \leq i < j \leq m$, $T_i \cap T_j = \emptyset$; and,
   - $\bigcup_{i=1}^{m} T_i = \boldsymbol{Tags}(t) \cup \boldsymbol{Tags}(t')$;
2. let $\{\mathtt{V}_1, \ldots, \mathtt{V}_m\}$ be a set of m mutually distinct new variable names and, for each $i = 1, \ldots, m$, associate the unique name $\mathtt{V}_i$ with one of the set of tags T_i in the partition;
3. replace each tag name $\mathtt{X} \in \boldsymbol{Tags}(t) \cup \boldsymbol{Tags}(t')$ in the normal form with the unique new tag name associated with its congruence class in the partition Π.

This eventual renaming of the normal form always results in a well-scoped tag endomorphism from $\boldsymbol{Tags}(t) \cup \boldsymbol{Tags}(t')$ to $\boldsymbol{Tags}(t \wedge t') \stackrel{\text{def}}{=} \{\mathtt{V}_1, \ldots, \mathtt{V}_m\}$.

Example 4.3 Scoped $\mathcal{OSF}$ term unification — Consider the terms ψ_1 and ψ_2 defined in Example 4.1 and the normal form corresponding to the ψ-term (4.7). Now, since:

$$\boldsymbol{Tags}(\psi_1) = \{\, \mathtt{X}, \mathtt{X}_1, \mathtt{X}_2 \,\}$$

[13] The tag set of an $\mathcal{OSF}$ term $X : s(f_1 \rightarrow t_1, \ldots, f_n \rightarrow t_n)$, $n \geq 0$, is defined as $\boldsymbol{Tags}(X : s(f_1 \rightarrow t_1, \ldots, f_n \rightarrow t_n)) \stackrel{\text{def}}{=} \{X\} \cup \bigcup_{i=1}^{n} \boldsymbol{Tags}(t_i)$. Note that this recursive definition is well-founded since, we have $\boldsymbol{Tags}(X : s) \stackrel{\text{def}}{=} \{X\}$ in the case $n = 0$.

and:

$$\textit{Tags}(\psi_2) = \{\, \mathtt{Y}, \mathtt{Y}_1, \mathtt{Y}_2, \mathtt{Y}_3 \,\}$$

the final congruence classes for each tag form the following partition of $\textit{Tags}(\psi_1) \cup \textit{Tags}(\psi_2)$:

$$\{\, \mathtt{X}, \mathtt{Y} \,\},\ \{\, \mathtt{X}_1, \mathtt{Y}_1, \mathtt{Y}_2 \,\},\ \{\, \mathtt{X}_2 \,\},\ \{\, \mathtt{Y}_3 \,\}.$$

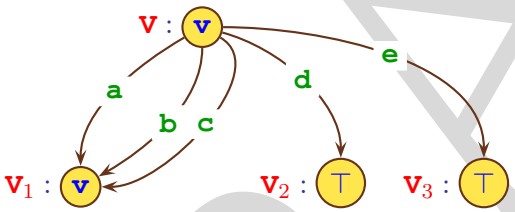
Hence, we can define the following new tag names, one per congruence class:

$$\begin{array}{l} \mathtt{V} \stackrel{\text{def}}{=} \{\, \mathtt{X}, \mathtt{Y} \,\}, \\ \mathtt{V}_1 \stackrel{\text{def}}{=} \{\, \mathtt{X}_1, \mathtt{Y}_1, \mathtt{Y}_2 \,\}, \\ \mathtt{V}_2 \stackrel{\text{def}}{=} \{\, \mathtt{X}_2 \,\}, \\ \mathtt{V}_3 \stackrel{\text{def}}{=} \{\, \mathtt{Y}_3 \,\} \end{array}$$

and replace each tag name in the ψ-term normal form Expression (4.7) and the graph it corresponds to in Expression (4.8) with its congruence class' new tag name:

$$\mathtt{V} : \mathtt{v}\,(\, \mathtt{a} \to \mathtt{V}_1 : \mathtt{v},\ \mathtt{b} \to \mathtt{V}_1,\ \mathtt{c} \to \mathtt{V}_1,\ \mathtt{d} \to \mathtt{V}_2,\ \mathtt{e} \to \mathtt{V}_3 \,) \tag{4.10}$$

that is the $\mathcal{OSF}$-graph:



(4.11)

i.e., the well-scoped ψ-term:

$$\psi_1 \wedge \psi_2 \;=\; \mathtt{V} : \mathtt{v}\,(\, \mathtt{a} \to \mathtt{V}_1 : \mathtt{v},\ \mathtt{b} \to \mathtt{V}_1,\ \mathtt{c} \to \mathtt{V}_1,\ \mathtt{d} \to \mathtt{V}_2,\ \mathtt{e} \to \mathtt{V}_3 \,)$$

that is the **glb** of ψ_1 and ψ_2 with the tag mapping γ defined formally as mapping each original tag in $\textit{Tags}(\psi_1) \cup \textit{Tags}(\psi_1)$ to its new tag representative. Namely, in this example's case:

$$\begin{array}{ll} \gamma(\mathtt{X}) = \mathtt{V}, & \gamma(\mathtt{Y}) = \mathtt{V}, \\ \gamma(\mathtt{X}_1) = \mathtt{V}_1, & \gamma(\mathtt{Y}_1) = \mathtt{V}_1, \\ \gamma(\mathtt{X}_2) = \mathtt{V}_2, & \gamma(\mathtt{Y}_2) = \mathtt{V}_1, \\ & \gamma(\mathtt{V}_3) = \mathtt{Y}_3 \end{array}$$

as shown in Figure 4.7 (which realizes the lower part of Figure 4.5).

§ $\mathcal{OSF}$ Generalization: Induction by Constraint Normalization

Just like $\mathcal{FOT}$s, ψ-terms also possess an operation of generalization that is dual to unification ([4], [26]). The operation is very similar, with the additional taking into account of common symbolic

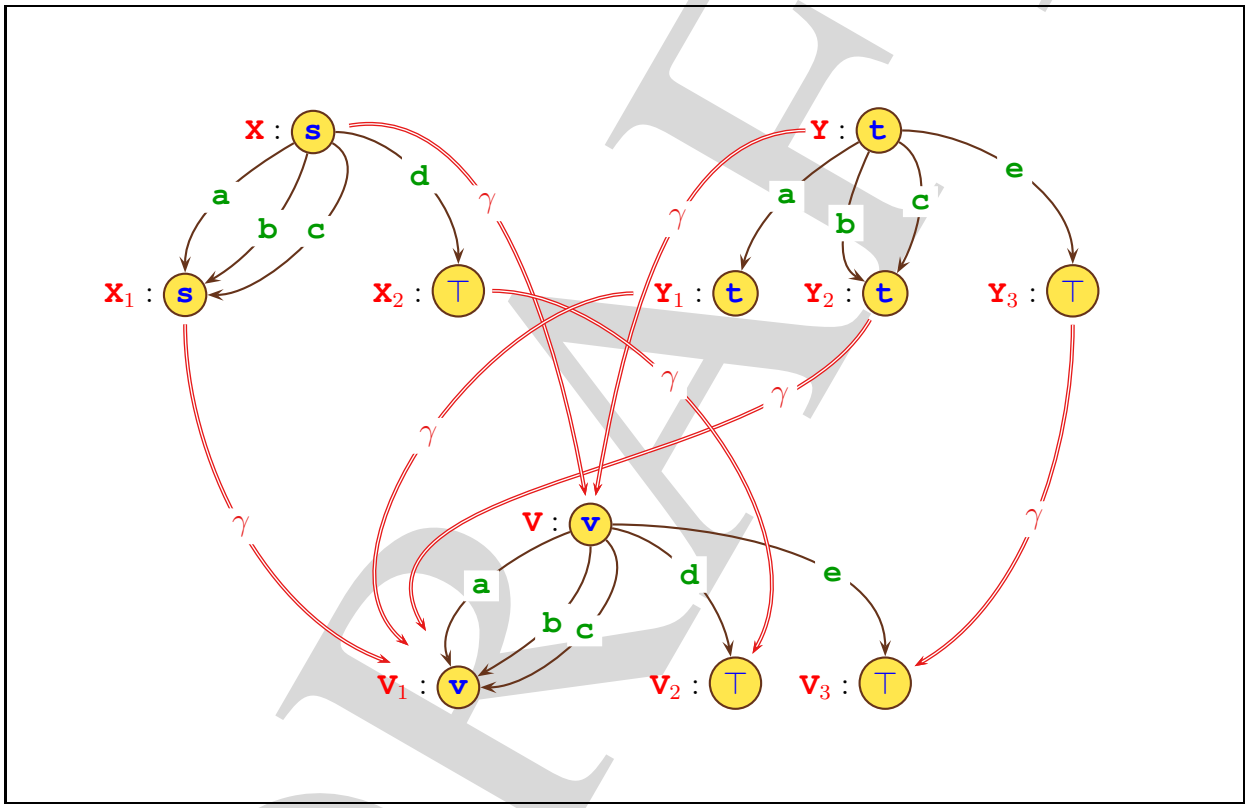


Figure 4.7: $\mathcal{OSF}$ graph endomorphism realizing scoped $\mathcal{OSF}$ unification

features rather then all contiguous positions, as well as partially ordered sorts denoting sets and the sort ordering denoting set inclusion.

What follows is a formal specification of the $\mathcal{OSF}$ generalization operation. This formulation is different than the one used in [26]. It is equivalent to it, however, expressed with this now familiar notation we used for $\mathcal{FOT}$ generalization in Section 3.6, extended to $\mathcal{OSF}$ terms. That is,

$$\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \psi \begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix}$$

where ψ is the ψ-term generalizing the ψ-terms ψ_1 and ψ_2, and (γ_1, γ_2) and (γ'_1, γ'_2) are, respectively, the judgement's pairs of *prior* and *posterior* tag maps.[14] In order to do this, all the generalization constraint normalization rules must be reformulated into syntax-directed judgment axioms and rules directly on the syntactic form of ψ-terms (*i.e.*, $\mathcal{OSF}$ terms in normal form) defined in Equation (4.1), rather than on dissolved $\mathcal{OSF}$ constraints as done in [26]. The dissolved form is more convenient for unification while traditional term syntax is more convenient for our formulation of generalization, as for that of $\mathcal{FOT}$s (and the fuzzification of the corresponding lattice operation).

Since our definition of the generalizing operation's definition will assume that its arguments are ψ-terms (*i.e.*, $\mathcal{OSF}$ terms in normal form), each tag symbol occurring in either term will always be the root tag of a unique ψ-term. This property is easily verified to be preserved by the axiom and the rule of Figure 4.8. Thus, a sortless occurrence of a tag always refers to (*i.e.*, is the root tag of) this unique ψ-term. If none of a tag symbol's occurrences is sorted, then this tag is the root of a common occurrence of the most general sort: $\top$ (*i.e.*, "*anything*").

Axiom **EQUAL TAGS** in Figure 4.8 simply states that generalizing a pair made of the same ψ-term results in this ψ-term and the posterior tag maps are the same as the prior ones.

Rule **UNEQUAL TAGS** in Figure 4.8 uses an "unapply" operation '$\uparrow$' that is defined as follows:

$$\begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \uparrow \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \stackrel{\text{def}}{=} \begin{cases} \begin{pmatrix}X:\ldots\\ X:\ldots\end{pmatrix} & \text{if } \exists\, X \text{ s.t. } \gamma_i(X) = \mathbf{ROOT}(\psi_i), \text{ for } i = 1, 2; \\ \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} & \text{otherwise.} \end{cases} \tag{4.12}$$

This has the same purpose as the unapply operation used in $\mathcal{FOT}$ generalization judgments: identify in the prior pair of tag maps (γ_1, γ_2) whether or not they already map a common variable (X) to the roots of the pair of ψ-terms to be generalized (ψ_1, ψ_2). If so, the result of the unapplication is the pair made of the same ψ-term rooted in X $(X:\ldots, X:\ldots)$; if not, it is the original pair of ψ-terms (ψ_1, ψ_2).

Rule **UNEQUAL TAGS** of Figure 4.8 states that generalizing two ψ-terms $\psi_1 \stackrel{\text{def}}{=} X : s\,(f_i \to \psi_i)_{i=0}^m$ and $\psi_2 \stackrel{\text{def}}{=} Y : t\,(g_j \to \xi_j)_{j=0}^n$ results in the ψ-term $\psi_1 \vee \psi_2 \stackrel{\text{def}}{=} Z : s \vee t\,(h_k \to \chi_k)_{k=0}^p$, where the set of features of the resulting ψ-term is the intersection of the sets of features of ψ_1 and ψ_2 (*i.e.*, the features they have in common), and Z is a new tag name, when each subterm is the result of generalizing each of the corresponding pairs of subterms under all common features.

[14] A tag map is a tag-to-tag substitution.

EQUAL TAGS

$$\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash \begin{pmatrix}\psi\\ \psi\end{pmatrix} \psi \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}$$

UNEQUAL TAGS

$$\left[\begin{array}{l} X \neq Y; \\ m, n, p \geq 0 \text{ and } \{h_1, \ldots, h_p\} \stackrel{\text{def}}{=} \{f_1, \ldots, f_m\} \cap \{g_1, \ldots, g_n\} \text{ s.t. } h_k \stackrel{\text{def}}{=} f_k = g_k \text{ for all } k = 1, \ldots, p; \\ \gamma_1^0 \stackrel{\text{def}}{=} \gamma_1 \circ \{X/Z\} \text{ and } \gamma_2^0 \stackrel{\text{def}}{=} \gamma_2 \circ \{Y/Z\}, \text{ where } Z \text{ is a new tag name} \end{array}\right]$$

$$\frac{\begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \vdash \begin{pmatrix}\psi_1\\ \xi_1\end{pmatrix} \uparrow \begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \chi_1 \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \quad \ldots \quad \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \vdash \begin{pmatrix}\psi_p\\ \xi_p\end{pmatrix} \uparrow \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \chi_p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}}{\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash \begin{pmatrix}X : s\,(f_i \rightarrow \psi_i)_{i=1}^m\\ Y : t\,(g_j \rightarrow \xi_j)_{j=1}^n\end{pmatrix} Z : s \vee t\,(h_k \rightarrow \chi_k)_{k=1}^p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}}$$

Figure 4.8: Judgment-based $\mathcal{OSF}$ generalization axiom and rule

Note that in this rule's condition, $h_k \stackrel{\text{def}}{=} f_k = g_k$ for all $k = 1, \ldots, p$ imposes that the pair of ψ-term being generalized have their features ordered so that their common features f_k and g_k (if there are any — *i.e.*, if $p > 0$) have the same index $k \in \{1, \ldots, p\}$. This can always be done since ψ-term's features can be specified in any order. Note also that this rule's numerator's sequence of judgements can be in any order of the p common features, as long as each subterm judgment is validated with its pair of prior tag maps equal to the preceding judgment's pair of posterior tag maps. Finally, note also that when the set of common features is empty (*i.e.*, $p = 0$), its numerator reduces to true and Rule **UNEQUAL TAGS** reduces to the following conditional single-judgment axiom:

UNEQUAL TAGS W/ NO COMMON FEATURE

$$\left[\begin{array}{l} X \neq Y; \\ m, n \geq 0 \text{ and } \{f_1, \ldots, f_m\} \cap \{g_1, \ldots, g_n\} = \emptyset \\ \gamma_1^0 \stackrel{\text{def}}{=} \gamma_1 \circ \{X/Z\} \text{ and } \gamma_2^0 \stackrel{\text{def}}{=} \gamma_2 \circ \{Y/Z\}, \text{ where } Z \text{ is a new tag name} \end{array}\right]$$

$$\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash \begin{pmatrix}X : s\,(f_i \rightarrow \psi_i)_{i=1}^m\\ Y : t\,(g_j \rightarrow \xi_j)_{j=1}^n\end{pmatrix} Z : s \vee t \begin{pmatrix}\gamma_1^{0}\\ \gamma_2^{0}\end{pmatrix}$$

i.e., it is a fully proven single judgment with the corresponding endomorphic mapping extended with a newly introduced tag (Z) mapped to the respective root tags (X and Y). As seen in Example 4.4, this axiom together with Axiom **EQUAL TAGS** of Figure 4.8 provide the fully proven judgments that constitute the leaves of an $\mathcal{OSF}$-generalization proof tree.

Example 4.4 $\mathcal{OSF}$ **term generalization** — Let us consider the same sort signature and partial order, together with the same ψ-terms ψ_1 and ψ_2 of Example 4.2. Let us take the following $\mathcal{OSF}$ generalization judgment constraint to resolve:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \psi_1 \vee \psi_2 \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}$$

in which the ψ-term $\psi_1 \vee \psi_2$ and the generalizing endomorphic tag maps γ_1 and γ_2 are to be determined by normalizing this judgment according to the axiom and rule of Figure 4.8.

We start with the judgment:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X} : \mathtt{s}\,(\mathtt{a} \to \mathtt{X}_1 : \mathtt{s},\ \mathtt{b} \to \mathtt{X}_1,\ \mathtt{c} \to \mathtt{X}_1,\ \mathtt{d} \to \mathtt{X}_2)\\ \mathtt{Y} : \mathtt{t}\,(\mathtt{a} \to \mathtt{Y}_1 : \mathtt{t},\ \mathtt{b} \to \mathtt{Y}_2 : \mathtt{t},\ \mathtt{c} \to \mathtt{Y}_2,\ \mathtt{e} \to \mathtt{Y}_3)\end{pmatrix} \psi_1 \vee \psi_2 \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}.$$

Applying Rule **UNEQUAL TAGS**, since $\mathtt{s} \vee \mathtt{t} = \mathtt{u}$, keeping only common features and introducing a new tag $\mathtt{U}$, this yields $\psi_1 \vee \psi_2 = \mathtt{U} : \mathtt{u}(\mathtt{a} \to \psi',\ \mathtt{b} \to \psi'',\ \mathtt{c} \to \psi''')$ and this judgment becomes the following sequence of three judgments:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U}\}\\ \{\mathtt{Y}/\mathtt{U}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_1 : \mathtt{t}\end{pmatrix} \uparrow \begin{pmatrix}\{\mathtt{X}/\mathtt{U}\}\\ \{\mathtt{Y}/\mathtt{U}\}\end{pmatrix} \psi' \begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix},$$

$$\begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_2 : \mathtt{t}\end{pmatrix} \uparrow \begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix} \psi'' \begin{pmatrix}\gamma''_1\\ \gamma''_2\end{pmatrix},$$

$$\begin{pmatrix}\gamma''_1\\ \gamma''_2\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_2 : \mathtt{t}\end{pmatrix} \uparrow \begin{pmatrix}\gamma''_1\\ \gamma''_2\end{pmatrix} \psi''' \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}.$$

Evaluating the unapplication in the first of these judgments, it becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U}\}\\ \{\mathtt{Y}/\mathtt{U}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_1 : \mathtt{t}\end{pmatrix} \psi' \begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix},$$

to which we can apply again Rule **UNEQUAL TAGS**, since $\mathtt{s} \vee \mathtt{t} = \mathtt{u}$, introducing a new tag $\mathtt{U}_1$, this first judgment becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U}\}\\ \{\mathtt{Y}/\mathtt{U}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_1 : \mathtt{t}\end{pmatrix} \mathtt{U}_1 : \mathtt{u} \begin{pmatrix}\{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1\}\\ \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1\}\end{pmatrix}.$$

This, in turn, makes the second judgment in the sequence become:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1\}\\ \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_2 : \mathtt{t}\end{pmatrix} \uparrow \begin{pmatrix}\{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1\}\\ \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1\}\end{pmatrix} \psi'' \begin{pmatrix}\gamma''_1\\ \gamma''_2\end{pmatrix},$$

and after evaluating the unapplication, it becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1\}\\ \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_2 : \mathtt{t}\end{pmatrix} \psi'' \begin{pmatrix}\gamma''_1\\ \gamma''_2\end{pmatrix},$$

to which we can apply again Rule **UNEQUAL TAGS**, since $\mathtt{s} \vee \mathtt{t} = \mathtt{u}$, introducing a new tag $\mathtt{U}_2$ this second judgment becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1\}\\ \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1 : \mathtt{s}\\ \mathtt{Y}_2 : \mathtt{t}\end{pmatrix} \mathtt{U}_2 : \mathtt{u} \begin{pmatrix}\{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1, \mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1, \mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix}.$$

This then makes the third judgment in the initial sequence become:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U},\mathtt{X}_1/\mathtt{U}_1,\mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U},\mathtt{Y}_1/\mathtt{U}_1,\mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{X}_1:\mathtt{s}\\ \mathtt{Y}_2:\mathtt{t}\end{pmatrix} \uparrow \begin{pmatrix}\{\mathtt{X}/\mathtt{U},\mathtt{X}_1/\mathtt{U}_1,\mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U},\mathtt{Y}_1/\mathtt{U}_1,\mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix} \psi''' \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}.$$

But now, because both $\mathtt{X}_1/\mathtt{U}_2$ and $\mathtt{Y}_2/\mathtt{U}_2$ are in the two respective prior tag maps, the unapplication in this judgment comes to:

$$\begin{pmatrix}\mathtt{X}_1:\mathtt{s}\\ \mathtt{Y}_2:\mathtt{t}\end{pmatrix} \uparrow \begin{pmatrix}\{\mathtt{X}/\mathtt{U},\mathtt{X}_1/\mathtt{U}_1,\mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U},\mathtt{Y}_1/\mathtt{U}_1,\mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix} = \begin{pmatrix}\mathtt{U}_2:\mathtt{u}\\ \mathtt{U}_2:\mathtt{u}\end{pmatrix}$$

which makes the third judgment become:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U},\mathtt{X}_1/\mathtt{U}_1,\mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U},\mathtt{Y}_1/\mathtt{U}_1,\mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{U}_2:\mathtt{u}\\ \mathtt{U}_2:\mathtt{u}\end{pmatrix} \psi''' \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix};$$

and using Axiom **EQUAL TAGS**, this third judgment becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{U},\mathtt{X}_1/\mathtt{U}_1,\mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U},\mathtt{Y}_1/\mathtt{U}_1,\mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{U}_2:\mathtt{u}\\ \mathtt{U}_2:\mathtt{u}\end{pmatrix} \mathtt{U}_2:\mathtt{u} \begin{pmatrix}\{\mathtt{X}/\mathtt{U},\mathtt{X}_1/\mathtt{U}_1,\mathtt{X}_1/\mathtt{U}_2\}\\ \{\mathtt{Y}/\mathtt{U},\mathtt{Y}_1/\mathtt{U}_1,\mathtt{Y}_2/\mathtt{U}_2\}\end{pmatrix}.$$

and terminates the proof.

This results in the ψ-term $\psi_1 \vee \psi_2$ that is the generalization of ψ_1 and ψ_2:

$$\psi_1 \vee \psi_2 \;=\; \mathtt{U}:\mathtt{u}(\mathtt{a} \to \mathtt{U}_1:\mathtt{u},\ \mathtt{b} \to \mathtt{U}_2:\mathtt{u},\ \mathtt{c} \to \mathtt{U}_2)$$

with the corresponding tag maps γ_1 and γ_2:[15]

$$\begin{array}{ll}\gamma_1(\mathtt{U}) = \mathtt{X}, & \gamma_2(\mathtt{U}) = \mathtt{Y},\\ \gamma_1(\mathtt{U}_1) = \mathtt{X}_1, & \gamma_2(\mathtt{U}_1) = \mathtt{Y}_1,\\ \gamma_1(\mathtt{U}_2) = \mathtt{X}_1, & \gamma_2(\mathtt{U}_2) = \mathtt{Y}_2\end{array}$$

as pictured in Figure 4.9.

Putting Figure 4.9 together with Figure 4.7, as we do in Figure 4.10, shows the lattice structure of the $\mathcal{OSF}$ graphs together with the three endomorphic tag maps γ, γ_1, and γ_2 that realize it as required by the lattice diagram show in Figure 4.5.[16]

[15] Or, $\gamma_1 = \{\mathtt{X}/\mathtt{U}, \mathtt{X}_1/\mathtt{U}_1, \mathtt{X}_1/\mathtt{U}_2\}$ and $\gamma_2 = \{\mathtt{Y}/\mathtt{U}, \mathtt{Y}_1/\mathtt{U}_1, \mathtt{Y}_2/\mathtt{U}_2\}$, when writing an endomorphism as a tag substitution.

[16] For a detailed example with more realistic sort signature and feature names showing the result of applying these normalization rules compute the **glb** and the **lub** of two ψ-terms with meaningful symbols, see Example B.2 in Appendix B.2.

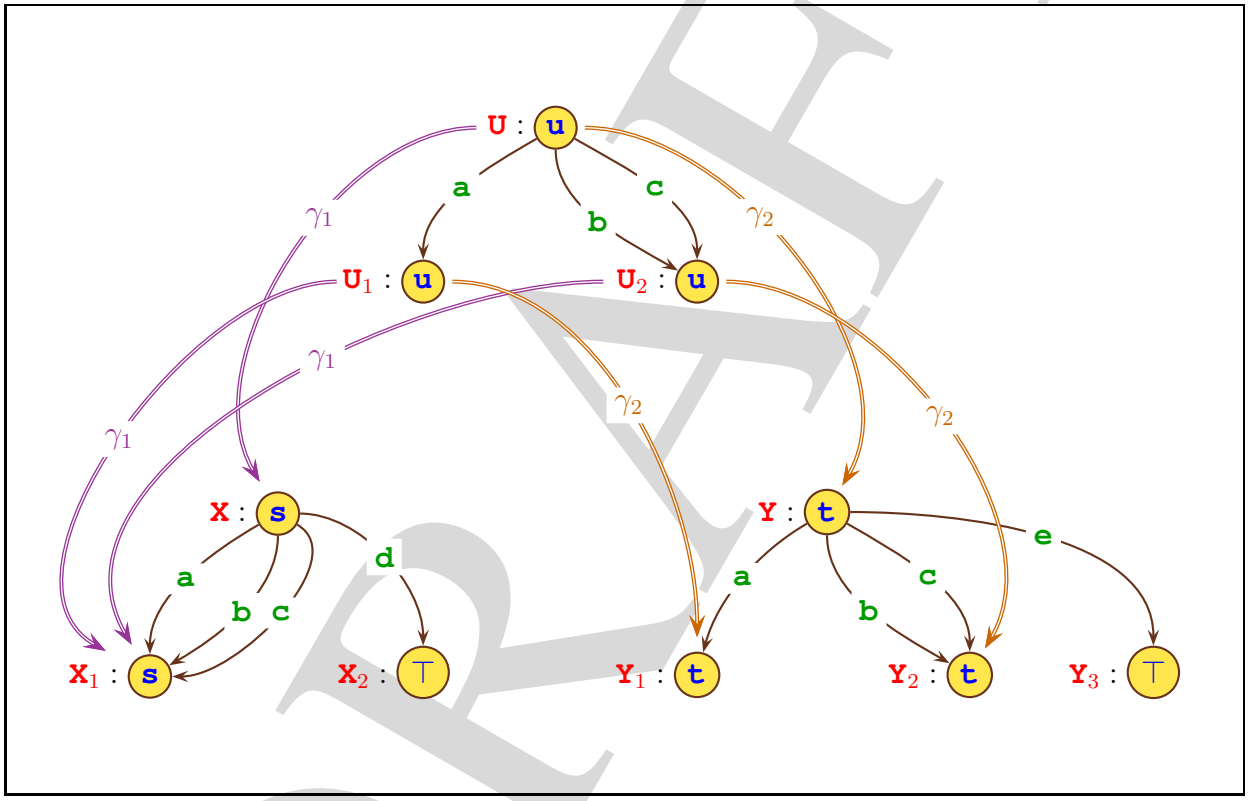


Figure 4.9: Pair of $\mathcal{OSF}$ graph endomorphisms realizing $\mathcal{OSF}$ generalization

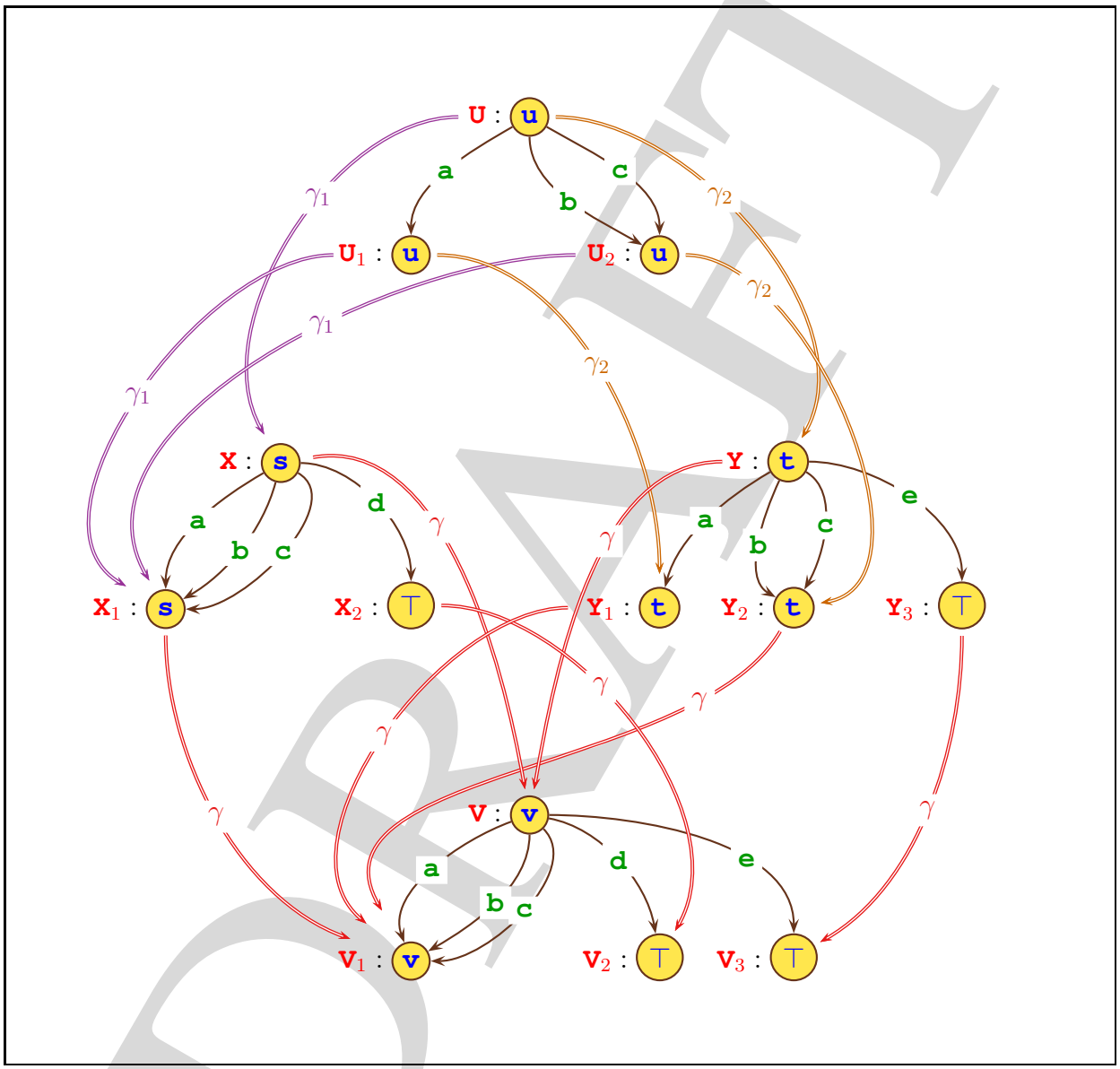


Figure 4.10: Scoped $\mathcal{OSF}$-graph inheritance-lattice endomorphisms

4.2 Fuzzifying $\mathcal{OSF}$-Term Subsumption

We are now ready to develop the same scheme of fuzzy declensions on $\mathcal{OSF}$ terms that we performed on the $\mathcal{FOT}$s in Chapter 3, Section 3.7, however dealing with (and taking advantage of) the more general lattice-theoretic semantics we have defined for our syntax of $\mathcal{OSF}$ terms and its derived operational calculus on $\mathcal{OSF}$ constraints in Section 4.1. We shall proceed as we did for $\mathcal{FOT}$s, up to partially ordered sort and unconstrained symbol feature arities. So let us start by making some important observations regarding some advantages in our approach to fuzzifying $\mathcal{FOT}$ unification and generalization.

1. The term structure itself (its syntax) is not fuzzified. For unification, only a conjunctive set E of equations (pairs of first-order terms — including substitutions) is given a similarity degree α. This is denoted as the fuzzy-weighted set E_α. For generalization, only a pair of tag substitutions in a judgment is given such a similarity degree α.

2. In unification rules the similarity degree of a conjunctive set of equations, and in generalization rule and axioms of a pair of tag mappings, can never increase from prior to posterior forms.

3. There is a similarity relation $\sim$ on functors f and g (as a half-matrix of similarity degrees in $[0.0, 0.1]$).[17]

4. For each pair of functors f/m and g/n with $f \neq g$ and $0 \leq m \leq n$, whenever $f \sim_\alpha g$ with $\alpha \in (0.0, 1.0]$ there is a one-to-one mapping $p : \{1, \ldots, m\} \to \{1, \ldots, n\}$ associating each argument position of f to a unique distinct argument position of g. This mapping is the identity on $\{1, \ldots, m\}$ by default; it is undefined for dissimilar functors. In axioms and rules, when terms with similar functors with possible arity mismatch are equated, this argument-position mapping realigns misaligned subterms; subterms in the higher-arity term that are in excess are ignored.

Then, fuzzifying lattice operations for $\mathcal{FOT}$s consisted in adapting their crisp normalization rules to carry a similarity degree according to the above observations when normalizing a $\mathcal{FOT}$ equation set or proving a $\mathcal{FOT}$ generalization judgment.

When considering $\mathcal{OSF}$ terms, we can proceed similarly, but instead of $\mathcal{FOT}$ unification, let us consider what this means for the ruleset Figure 4.6 and 5.5, and instead of $\mathcal{FOT}$ generalization, what this means for the axioms and rule in Figure 4.8, which enforce constraint consistency when subsumption is realized by endomorphic tag mappings, which are sets of variable/variable equations — *i.e.*, $X \doteq Y$ — respecting sorts and feature application. These rules and axioms operate taking into account the following observations.

1. The $\mathcal{OSF}$ term itself is not fuzzified. For $\mathcal{OSF}$ unification, only a conjunctive set ϕ of atomic constraints (each of either of the forms $X : s$, $X.f \doteq Y$, and $X \doteq Y$) is given a global similarity degree α as the fuzzy formula ϕ_α. For $\mathcal{OSF}$ generalization, only a pair of tag substitutions in a judgment is given such a similarity degree α.

[17] Only one direction is needed; the other is equal by symmetry — see Chapter 2, Section 2.2.2.

2. In unification rules the similarity degree of a conjunctive set of atomic $\mathcal{OSF}$ constraints, and in generalization rule and axioms of a pair of $\mathcal{OSF}$ tag mappings, can never increase from prior to posterior forms.

3. There is a similarity relating pairs of sorts s and s' as a half-matrix of similarity degrees in $[0.0, 0.1]$.

So this looks pretty much the same as for $\mathcal{FOT}$s, except for one important detail: in the case of $\mathcal{OSF}$ terms, a similarity relation on a signature of partially ordered featured sorts must be also consistent with the ordering $\preceq$ on sorts. This means that, for all sorts s, s', t, t' in $\mathcal{S}$, the following *fuzzy sort-lattice consistency conditions* must hold for **lub**s and **glb**s when they exist:

$$\left.\begin{array}{l} \textbf{if } s \sim_\alpha s' \textbf{ and } t \sim_\beta t' \textbf{ then } (s \curlywedge t) \sim_{\alpha\wedge\beta} (s' \curlywedge t'), \\ \textbf{if } s \sim_\alpha s' \textbf{ and } t \sim_\beta t' \textbf{ then } (s \curlyvee t) \sim_{\alpha\wedge\beta} (s' \curlyvee t'); \end{array}\right\} \tag{4.13}$$

Note that the similarity degree in both foregoing fuzzy sort-lattice consistency conditions on the lattice operations on sorts, uses fuzzy conjunction $(\wedge)$ of approximation degrees. While this may be expected for $\curlywedge$, it could appear odd for $\curlyvee$. However, it is correct to use $\wedge$ in both cases as we do because the homomorphic constraints expressed by Condition (4.13) apply to the *logical conjunction* "**and**" in the statement combining their premisses. In fact, the following (incorrect) constraint:

$$\textbf{if } s \sim_\alpha s' \textbf{ and } t \sim_\beta t' \textbf{ then } (s \curlyvee t) \sim_{\alpha\vee\beta} (s' \curlyvee t')$$

could make the fuzzy degree of an expression resulting from the fuzzy conjunction of two fuzzy expressions be greater than the degree of each — which is, again, incoherent since conjoining more fuzzy information can only decrease the resulting overall fuzzy degree.

In particular, a consequence of the fuzzy sort-lattice consistency conditions (4.13) is the following *fuzzy sort-order consistency condition* for any sorts $s, t, s', t' \in \mathcal{S}$ such that $s \preceq t$ and $s' \preceq t'$:

$$\textbf{if } s \sim_\alpha s' \textbf{ and } t \sim_\beta t' \textbf{ then } s \sim_{\alpha\wedge\beta} s'. \tag{4.14}$$

This is illustrated generically in Figure 4.11 with abstract sorts and similarity weights, and with possible specific sorts and similarity weights defining an instance case pictured as Figure 4.12.

For example, given a similarity on sorts such that:

$$\begin{array}{l} \texttt{employee} \sim_{.8} \texttt{assistant} \\ \texttt{student} \sim_{.9} \texttt{apprentice} \end{array}$$

and an ordering on sorts such that:

$$\begin{array}{lcl} \texttt{helper} & \stackrel{\text{def}}{=} & \texttt{assistant} \curlyvee \texttt{apprentice} \\ \texttt{intern} & \stackrel{\text{def}}{=} & \texttt{assistant} \curlywedge \texttt{apprentice} \end{array}$$

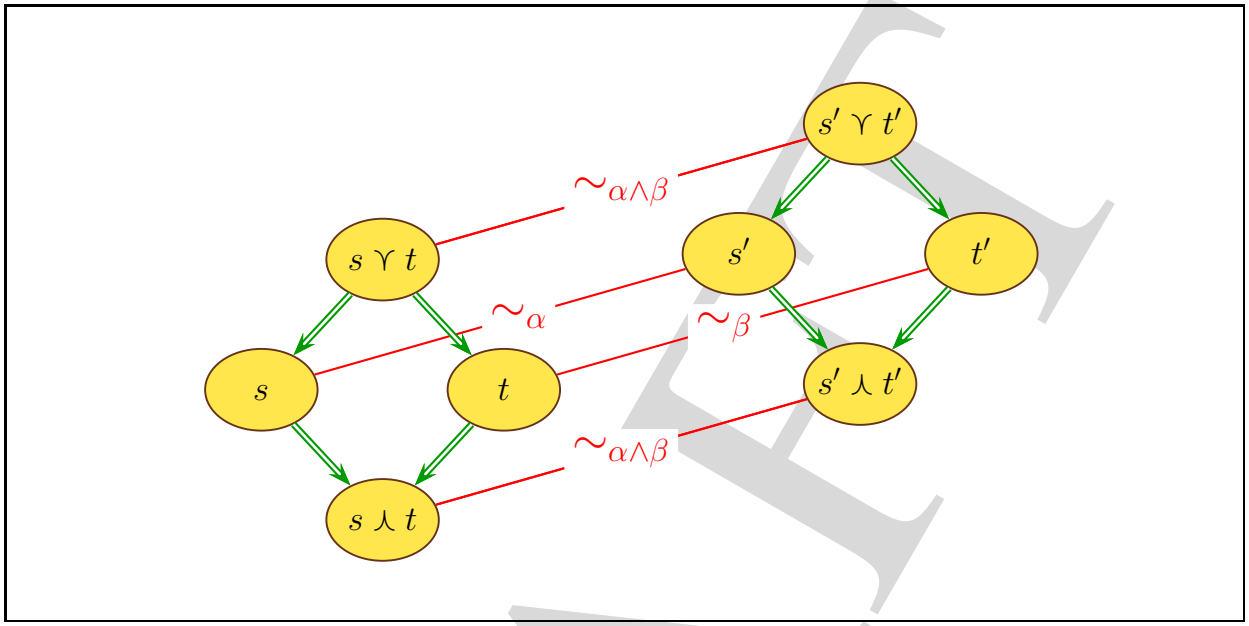


Figure 4.11: Order-inconsistent sort similarity

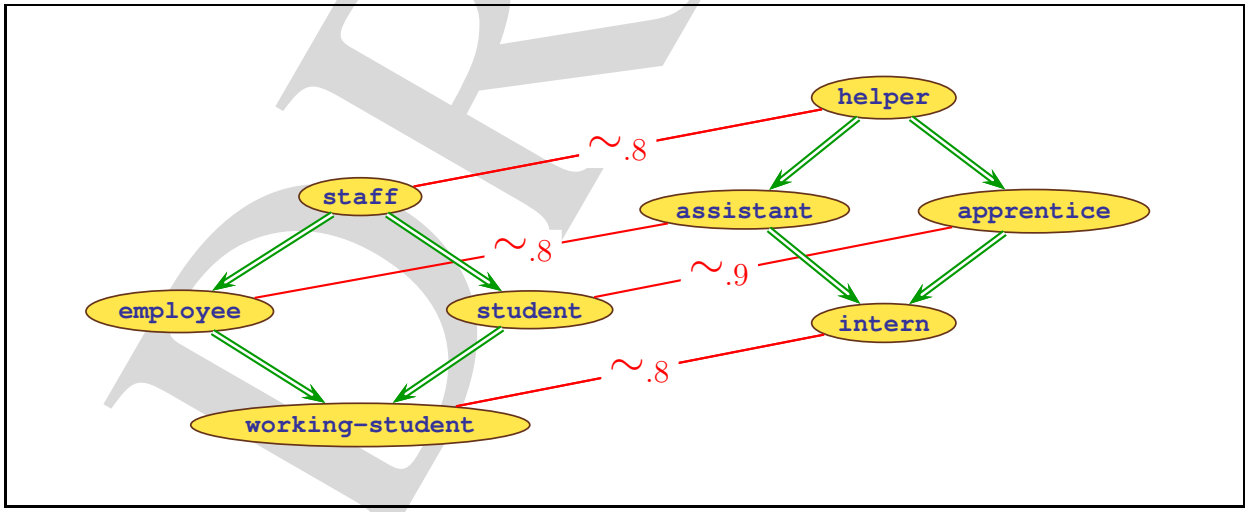


Figure 4.12: Order-inconsistent sort similarity example

and

$$\begin{aligned} \texttt{staff} &\stackrel{\text{def}}{=} \texttt{employee} \curlyvee \texttt{student} \\ \texttt{working-student} &\stackrel{\text{def}}{=} \texttt{employee} \curlywedge \texttt{student} \end{aligned}$$

then, necessarily for a consistent set of sorts, it must be that:

$$\texttt{staff} \sim_{.8} \texttt{helper}.$$

But then, since **student** is a subsort of **staff** and since **apprentice** is a subsort of **helper**, order-consistency entails that the $\sim_{.8}$ similarity is inherited by all their respective subsorts. In particular, this implies $\texttt{student} \sim_{.8 \wedge .9} \texttt{apprentice}$; namely, $\texttt{student} \sim_{.8} \texttt{apprentice}$. Similarly, order-consistency mandates that:

$$\texttt{working-student} \sim_{.8} \texttt{intern}.$$

Therefore, in order to ensure that a sort similarity $\sim$ is always consistent with the subsort ordering for all degree $\alpha \in \textbf{DEGREES}^{\sim}$, this necessitates that after declaring a few pairs of sorts to be similar at a given approximation degree, all their respective subsorts must also undergo an order-consistency closure. This closure consists in propagating similarities of all pairs of sorts to their subsorts as mandated by Condition (4.14). This is made formally precise next, while related considerations regarding how to compute and implement the fuzzy transitive closure of pairs of sorts that are declared similar in a fuzzy taxonomy are discussed in Section 4.2.4.

4.2.1 Fuzzy *vs.* subsort approximation

Before we proceed into further technicalities concerning fuzzy $\mathcal{OSF}$ term-lattice operations as fuzzy-constraint solving, let us discuss important implications of what a fuzzy ordering on sorts means. Then, using what we understand this to mean formally, let us make some specific remarks that will be helpful in understanding and justifying the correctness of the constraint-solving rules and axioms we shall propose.[18]

A fuzzy ordering $\preceq$ on the set of sorts $\mathcal{S}$ means by definition that it is a fuzzy relation on $\mathcal{S}$ that is reflexive, anti-symmetric, and transitive. This implicitly defines the following fuzzy relations on $\mathcal{S}$:

- a similarity $\sim$ on $\mathcal{S}$ defined, for any $\alpha \in [0.0, 0.1]$, as:

$$\sim_\alpha \stackrel{\text{def}}{=} \preceq_\alpha \wedge \succeq_\alpha \tag{4.15}$$

  where $\succeq_\alpha$ is the fuzzy relation on $\mathcal{S}$ defined as: $\succeq_\alpha \stackrel{\text{def}}{=} \preceq_\alpha^{-1}$;

- a fuzzy partial order $\preceq$ on $\mathcal{S}$: a fuzzy set $\Pi^{\sim} \stackrel{\text{def}}{=} \{\, \Pi_\alpha^{\sim} \mid \alpha \in \textbf{DEGREES}^{\sim} \,\}$ of partitions of $\mathcal{S}$ with partial orders $\preceq_\alpha$ on each partition $\Pi_\alpha^{\sim}$ of $\mathcal{S}$ generated by $\sim$ (*i.e.*, Zadeh's "partition tree"), and defined at approximation degree $\alpha \in [0.0, 01]$ as:

$$[s]_\alpha^{\sim} \preceq_\alpha [t]_\alpha^{\sim} \text{ iff } \exists\, s' \in \mathcal{S},\ \exists\, t' \in \mathcal{S} \text{ s.t. } s \sim_\alpha s' \text{ and } s' \preceq_\alpha t' \text{ and } t' \sim_\alpha t. \tag{4.16}$$

[18] The reader is also invited to refer to Chapter 2, Section 2.2.1 for a quick reminder of important facts this section relies on.

This last condition may look harder to read than what it actually means, and can be understood more easily as the following color-highlighted diagram:

$$[s]^{\sim}_{\alpha} \preceq_{\alpha} [t]^{\sim}_{\alpha} \quad \overset{\text{def}}{\textit{iff}} \quad \exists\, s' \in \mathcal{S},\, \exists\, t' \in \mathcal{S} \;\text{ s.t. :} \begin{cases} t' & \sim_{\alpha} & t \\ & \preceq_{\alpha} & \\ s & \sim_{\alpha} & s' \end{cases}$$

where a subsort is below its supersort and similar sorts are on the same level.

The two following lemmas are also a direct consequence of the above properties.

LEMMA 4.1 (SORT-SUBSUMPTION FUZZY SYMMETRY) *For any sorts s and t in $\mathcal{S}$, and any approximation degrees α and β in $[0.0, 1.0]$, if $s \preceq_{\alpha} t$ and $t \preceq_{\beta} s$, then $s \sim_{\alpha\wedge\beta} t$.*

PROOF
[To be completed later. . .] □

LEMMA 4.2 (SORT-SUBSUMPTION FUZZY TRANSITIVITY) *For any sorts s, t, and u in $\mathcal{S}$, and any approximation degrees α and β in $[0.0, 1.0]$, if $s \preceq_{\alpha} t$ and $t \preceq_{\beta} u$, then $s \sim_{\alpha\wedge\beta} u$.*

PROOF
[To be completed later. . .] □

An important consequence is that, when considering a similarity on a sort signature $\mathcal{S}$ that is partially ordered by a defined sort subsumption, this must necessarily obey some consistency conditions for the similarity and the sort ordering. In particular, as the approximation degree α decreases from 1.0 to 0.0, the set of sorts $[s]^{\sim}_{\alpha}$ (denoting the similarity class of a sort s at approximation degree α) may only increase in size. Indeed, as α decreases, similarity classes of sorts may only coalesce, forming coarser and coarser similarity partitions.[19] It is not difficult to establish that the following properties are always true for any order-consistent similarity $\sim$ on a set of partially ordered sorts $\mathcal{S}, \preceq$.

PROPOSITION 4.1 (FUZZY SORT SUBSUMPTION CONTRAVARIANCE) *For all sort s in $\mathcal{S}$, and all approximation degrees α and β in $[0.0, 0.1]$:*

$$\alpha \leq \beta \quad \textit{iff} \quad [s]^{\sim}_{\beta} \subseteq [s]^{\sim}_{\alpha}\,. \tag{4.17}$$

PROOF
[To be completed later. . .] □

[19] This is the set of partitions $\Pi^{\sim}_{\alpha}$, $\alpha \in$ **DEGREES**$^{\sim}$ defined in Chapter 2, Section 2.2.2 and Section 2.2.3. This is also what Zadeh calls the similarity's "*partition tree*" [177]. See also [74], Chapter II, Section 3 on Fuzzy Relations (Page 77).

In words, the contravariance in Condition (4.17) of Proposition 4.1 states that the smaller the approximation degree, the larger the similarity class.

As a consequence, Corollary 4.1 states that all sorts are indistinguishable at approximation degree 0.0, since then all sort classes coalesce into a single similarity class equal to the whole set of sorts — which is what Condition (4.18) expresses.

COROLLARY 4.1 (FULLY SIMILAR SORTS) *For any sort $s \in \mathcal{S}$:*

$$[s]^{\sim}_{0.0} \;=\; \mathcal{S}\,. \tag{4.18}$$

Order-consistency between the fuzzy partial order on sorts $\preceq$ and inclusion $\subseteq$ on sort-similarity classes is established in the next proposition as a monotonic order isomorphism.

PROPOSITION 4.2 (SORT SUBSUMPTION MONOTONICITY) *For all sorts s and t in $\mathcal{S}$ and all approximation degree α in $[0.0, 0.1]$:*

$$s \preceq_{\alpha} t \quad \textit{iff} \quad [s]^{\sim}_{\alpha} \subseteq [t]^{\sim}_{\alpha}\,. \tag{4.19}$$

PROOF This follows from Theorem 4.1 (Page 81). □

On the other hand, as indicated by Condition (4.19) in Proposition 4.2, the approximation degree is covariant with the subsort ordering. When in addition the partially ordered sort signature is also a lattice $\mathcal{S}, \preceq, \curlywedge, \curlyvee$, this is equivalent to the validity of the following two propositions.

PROPOSITION 4.3 (FUZZY SORT-CONGRUENCE APPROXIMATION) *For all sort s in $\mathcal{S}$, and all approximation degrees α and β in $[0.0, 0.1]$:*

$$\begin{aligned} [s]^{\sim}_{\alpha\vee\beta} \;&=\; [s]^{\sim}_{\alpha} \cap [s]^{\sim}_{\beta}\,, \\ [s]^{\sim}_{\alpha\wedge\beta} \;&=\; [s]^{\sim}_{\alpha} \cup [s]^{\sim}_{\beta}\,. \end{aligned} \tag{4.20}$$

PROOF
[To be completed later. . .] □

Example 4.5 Fuzzy sort-congruence approximation — Let us take $s \stackrel{\text{def}}{=} \texttt{person}$, $\alpha = .6$, and $\beta = .4$, with $\vee \stackrel{\text{def}}{=} \max$ and $\wedge \stackrel{\text{def}}{=} \min$. Then, fuzzy sort subsumption contravariance (Proposition 4.1) is clearly satisfied since:

$$\begin{aligned} [\texttt{person}]^{\sim}_{.6} \cap [\texttt{person}]^{\sim}_{.4} \;&=\; [\texttt{person}]^{\sim}_{.6}\,, \\ [\texttt{person}]^{\sim}_{.6} \cup [\texttt{person}]^{\sim}_{.4} \;&=\; [\texttt{person}]^{\sim}_{.4}\,. \end{aligned}$$

PROPOSITION 4.4 (FUZZY SORT-CONGRUENCE LATTICE) *For all sorts s and t in $\mathcal{S}$ and all approximation degree α in $[0.0, 0.1]$:*

$$\begin{aligned} [s \curlyvee t]^{\sim}_{\alpha} &= [s]^{\sim}_{\alpha} \cup [t]^{\sim}_{\alpha} \,, \\ [s \curlywedge t]^{\sim}_{\alpha} &= [s]^{\sim}_{\alpha} \cap [t]^{\sim}_{\alpha} \,. \end{aligned} \tag{4.21}$$

PROOF
[To be completed later. . .] □

Example 4.6 Fuzzy sort-congruence lattice — Referring to the sorts in Figure 4.12, let us take $s \stackrel{\text{def}}{=} \texttt{employee}$, $t \stackrel{\text{def}}{=} \texttt{student}$ in Proposition 4.4. It is then obvious that, as stated by Proposition 4.4, for any $\alpha \in [0.0, 0.1]$:

$$\begin{aligned} [\texttt{staff}]^{\sim}_{\alpha} &= [\texttt{employee}]^{\sim}_{\alpha} \cup [\texttt{student}]^{\sim}_{\alpha} \,, \\ [\texttt{working-student}]^{\sim}_{\alpha} &= [\texttt{employee}]^{\sim}_{\alpha} \cap [\texttt{student}]^{\sim}_{\alpha} \,. \end{aligned}$$

Figure 4.13 and Figure 4.14 illustrate graphically how fuzzy subset approximation varies for three possible sorts **a**, **b**, and **c** as the approximation degree decreases from fully crisp (*i.e.*, 1.0 — when *no distinct sorts in $\mathcal{S}$ are similar*), to fully fuzzy (*i.e.*, 0.0 — when *all non-empty sorts in $\mathcal{S}$ are similar*). Figure 4.13, shows a typical possible example of fuzzy subsorting. Each row varies from fully crisp (degree 1.0) at the top, to fully fuzzy (degree 0.0) at the bottom as indicated in the leftmost column, while each of the other columns on the right indicates the fuzzy denotation of (from left to right) the sort $\top$ (which denotes the whole set of sorts $\mathcal{S}$) and sorts **a**, **b**, and **c**. As the approximation degree varies from crisp 1.0 to lower values, first to β, then to fuzzier $\alpha \leq \beta$, and ultimately to 0.0, each column shows a conceivable consistent variation of the denotation any sort $s \in \mathcal{S}$ (*e.g.*, in our example, $s = \texttt{a}$, $s = \texttt{b}$, and $s = \texttt{c}$):

$$0.0 \leq \alpha \leq \beta \leq 1.1 \implies \begin{cases} \top_{1.0} \subseteq \top_{\beta} \subseteq \top_{\alpha} \subseteq \top_{0.0} \,, \\ s_{1.0} \subseteq s_{\beta} \subseteq s_{\alpha} \subseteq s_{0.0} \,, \\ \bot_{1.0} \subseteq \bot_{\beta} \subseteq \bot_{\alpha} \subseteq \bot_{0.0} \,. \end{cases}$$

Since it is always true that $[\![\top_{\alpha}]\!] = \mathcal{S}$ at any approximation level $\alpha \in [0.0, 0.1]$,[20] the column of fuzzy top denotations on the left in Figure 4.13 is always the full set $\mathcal{S}$. So are all the sort denotations in the bottom row when all sorts are similar, since $[\![s_{0.0}]\!] = \mathcal{S}$ for any sort $s \in \mathcal{S}$. This means that whenever $\alpha \leq \beta$ for some $\alpha \in [0.0, 1.0]$ and $\beta \in [0.0, 1.0]$, then necessarily $s_{\beta} \preceq s_{\alpha}$ (*i.e.*, equivalently, $[\![s_{\beta}]\!] \subseteq [\![s_{\alpha}]\!]$ by definition), for any sort s in $\mathcal{S}$.

This combined consistent approximation effect of conjugating both fuzzy approximation and subsort approximation on sort denotations, together with coalescing sort-similarity classes, is illustrated in Figure 4.14, showing the sort-similarity lattice orderings of these classes for each

[20] This is a consequence of Condition (4.32) on Page 111 in Section 4.2.4, and Condition (2.24) defining a similarity class in Chapter 2, Section 2.2.

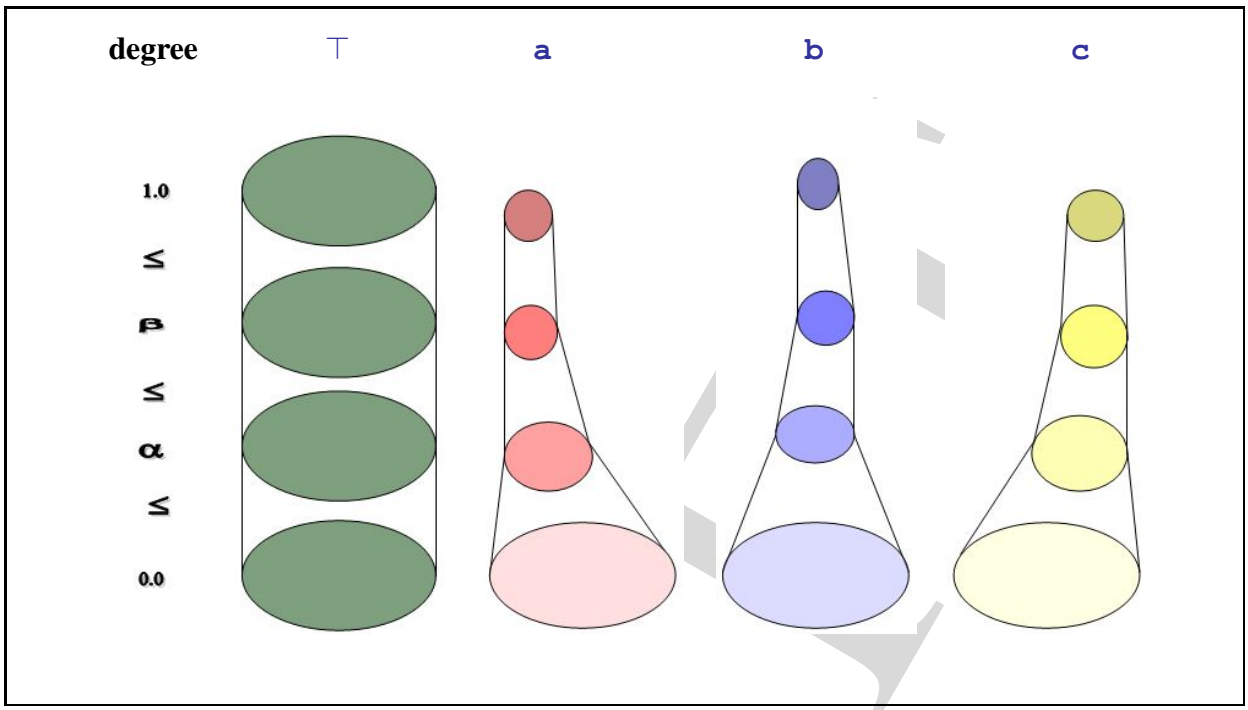


Figure 4.13: Fuzzy subset approximations

fuzzy level going from crisp at the top to fully fuzzy at the bottom. A good intuitive way to construe this effect is as that of a zooming lens or a magnifying-glass: approximation level 0.0 is farthest and most myopic (*i.e.*, "*seeing the forest for the trees*"), while level 1.0 is closest and sharpest (*i.e.*, "*seeing the trees for the forest*").

4.2.2 Fuzzy $\mathcal{OSF}$-term unification

We are, finally, ready to provide the constraint normalization rules for fuzzy $\mathcal{OSF}$-term unification and generalization. We first define formally what it means for two ψ-terms to be similar, modulo a subsort ordering and an order-consistent similarity (fuzzy equivalence) relation on the sorts.

§ Similarity of ψ-Terms

Let two ψ-terms ψ and ψ' defined as:

$$\psi \stackrel{\text{def}}{=} X : s(f_1 \to \psi_1, \ldots, f_n \to \psi_n)$$
$$\psi' \stackrel{\text{def}}{=} X' : s'(f'_1 \to \psi'_1, \ldots, f'_{n'} \to \psi'_{n'})$$

$(n, n' \geq 0)$.

Definition 4.5 *For $\alpha \in [0.0, 0.1]$, and two ψ-terms ψ and ψ' of the form above, we define recursively the fuzzy binary relation $\sim_\alpha$ on Ψ as $\psi \sim_\alpha \psi'$ iff $\alpha \stackrel{def}{=} \beta \wedge \bigwedge_{i=0}^{n} \beta_i$ where:*

$$s \sim_\beta s' \tag{4.22}$$

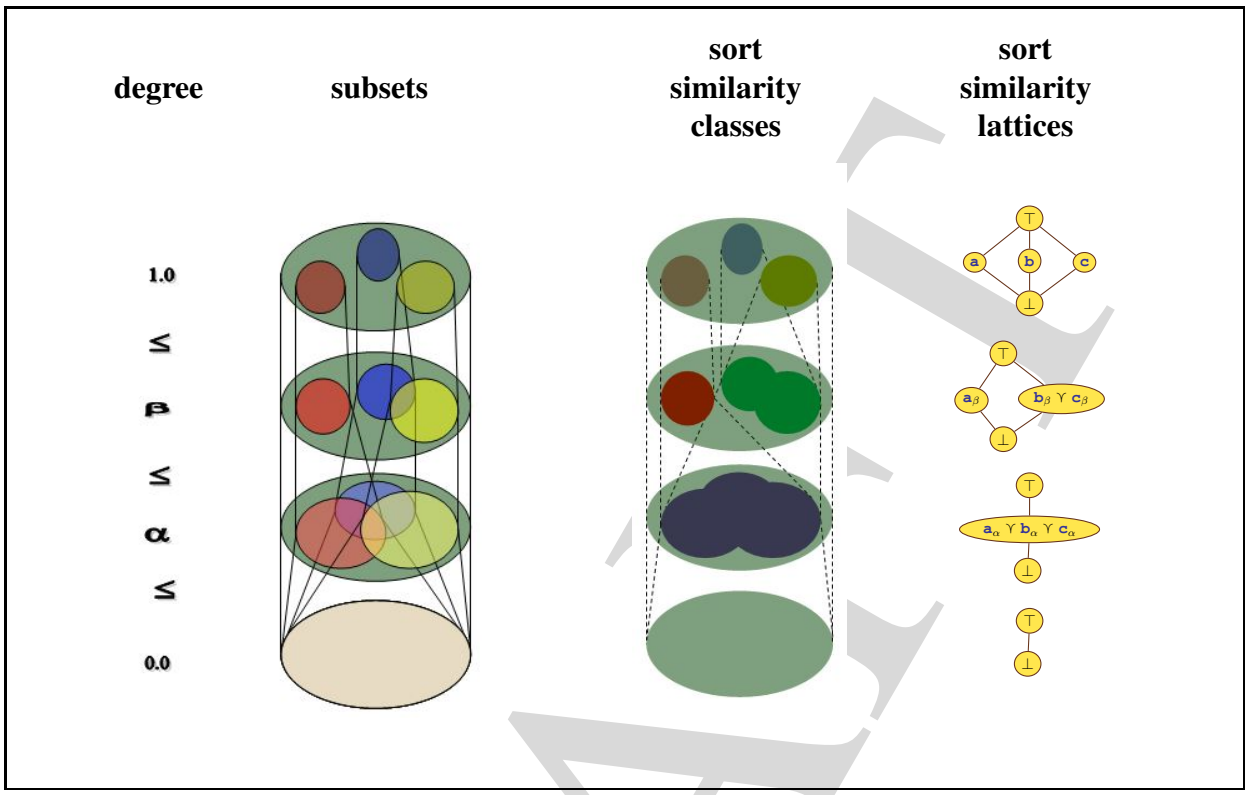


Figure 4.14: Fuzzy subset approximation lattice

for some $\beta \in (0.0, 1.0]$, *and:*

$$\psi_i \sim_{\beta_i} \psi'_{i'}[X/X'] \tag{4.23}$$

where $\beta_i \in (0.0, 1.0]$, *for any* $i \in \{1, \ldots, n\}$ *such that* $f_i \in \{f'_1, \ldots, f'_{n'}\}$.

The fuzzy $\mathcal{OSF}$ unification rules are shown in Figure 4.15.

THEOREM 4.3 (SIMILARITY OF ψ-TERMS) *The fuzzy binary relation* $\sim_\alpha$ *defined by Definition 4.5 is a similarity on the set of* ψ*-terms* Ψ.

PROOF **Reflexivity:** This follows because for any sort s, $s \sim_{1.0} s$, and because $\psi_i \sim_{1.0} \psi_i[X/X]$ reduces to the tautology $\psi_i \sim_{1.0} \psi_i$, for any f_i in $\mathcal{F}$, and any $X \in \mathcal{V}$, and thus also in particular for all $i = 1, \ldots, n$ for any $n \geq 0$.

Symmetry:

[To be completed later. . .] □

Because of Theorem 4.3, we shall say that ψ and ψ' are α*-similar* iff $\psi \sim_\alpha \psi'$.

SIMILAR SORT INTERSECTION

$$\frac{[s \sim_\beta t,\ 0 \leq \beta \leq 1] \qquad (\phi\ \&\ X : s\ \&\ X : t)_\alpha}{(\phi\ \&\ X : s \curlywedge t)_{\alpha \wedge \beta}}$$

FEATURE FUNCTIONALITY

$$\frac{(\phi\ \&\ X.f \doteq X'\ \&\ X.f \doteq X'')_\alpha}{(\phi\ \&\ X.f \doteq X'\ \&\ X' \doteq X'')_\alpha}$$

INCONSISTENT SORT

$$\frac{(\phi\ \&\ X \doteq \bot)_\alpha}{\mathfrak{false}_0}$$

TAG ELIMINATION

$$\frac{[Y \in \textbf{\textit{Tags}}(\phi)] \qquad (\phi\ \&\ X \doteq Y)_\alpha}{(\phi[X/Y]\ \&\ X \doteq Y)_\alpha}$$

NULL SIMILARITY DEGREE

$$\frac{\phi_0}{\mathfrak{false}_0}$$

Figure 4.15: Constraint normalization rules for fuzzy $\mathcal{OSF}$ unification

THEOREM 4.4 (CORRECTNESS OF FUZZY $\mathcal{OSF}$ UNIFICATION) *Given a fuzzy $\mathcal{OSF}$ constraint ϕ_α with $\alpha \in [0.0, 0.1]$, the process of non-deterministically applying to it any applicable rule shown in Figure 4.15 as long as one applies, always terminates in a fuzzy $\mathcal{OSF}$ constraint $\phi'_{\alpha'}$ such that either $\phi' = \mathfrak{false}$ and $\alpha' = 0$; or, $0 < \alpha' \leq \alpha$ and $\phi \sim_{\alpha'} \phi'$.*

PROOF
[To be completed later. . .] □

4.2.3 Fuzzy $\mathcal{OSF}$-term generalization

Axiom **FUZZY EQUAL TAGS** in Figure 4.16 states that generalizing a pair made of the same ψ-term results in this ψ-term and the posterior tag maps and approximation degree are the same as the prior ones.

Note that, Rule **FUZZY UNEQUAL TAGS** in Figure 4.16 uses a "fuzzy unapply" operation '$\uparrow_\alpha$' that takes a pair of ψ-terms with unequal root tags and an approximation degree α and returns a pair of (possibly identical) ψ-terms and a possibly lesser approximation degree. It is defined as

FUZZY EQUAL TAGS

$$\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}_{\alpha} \vdash \begin{pmatrix}\psi\\ \psi\end{pmatrix} \psi \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}_{\alpha}$$

FUZZY UNEQUAL TAGS

$$\left[\begin{array}{l} X \neq Y;\ s \sim_{\beta} t;\ \alpha_0 \stackrel{\text{def}}{=} \alpha \wedge \beta; \\ m, n, p \geq 0 \text{ and } \{h_1, \ldots, h_p\} \stackrel{\text{def}}{=} \{f_1, \ldots, f_m\} \cap \{g_1, \ldots, g_n\} \\ \text{s.t. } h_k \stackrel{\text{def}}{=} f_k = g_k \text{ for all } k = 1, \ldots, p; \\ \gamma_1^0 \stackrel{\text{def}}{=} \gamma_1 \circ \{X/Z\} \text{ and } \gamma_2^0 \stackrel{\text{def}}{=} \gamma_2 \circ \{Y/Z\}, \text{ where } Z \text{ is a new tag name} \end{array}\right]$$

$$\frac{\begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix}_{\alpha_0} \vdash \begin{pmatrix}\psi_1\\ \xi_1\end{pmatrix} \uparrow_{\alpha_0} \begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \chi 1 \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix}_{\alpha_1} \cdots \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix}_{\alpha_{p-1}} \vdash \begin{pmatrix}\psi_p\\ \xi_p\end{pmatrix} \uparrow_{\alpha_{p-1}} \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \chi_p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}_{\alpha_p}}{\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}_{\alpha} \vdash \begin{pmatrix}X : s\,(f_i \rightarrow \psi_i)_{i=1}^{m} \\ Y : t\,(g_j \rightarrow \xi_j)_{j=1}^{n}\end{pmatrix} Z : s \vee t\,(h_k \rightarrow \chi_k)_{k=1}^{p} \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}_{\alpha_p}}$$

Figure 4.16: Fuzzy $\mathcal{OSF}$ generalization axiom and rule

follows:

$$\begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \stackrel{\text{def}}{=} \begin{cases} \begin{pmatrix}X : \ldots\\ X : \ldots\end{pmatrix}_{\alpha \wedge \alpha_1 \wedge \alpha_2} & \text{if } \exists X \text{ s.t. } \mathbf{ROOT}(\psi_i) = \gamma_i(X) \text{ for } i = 1, 2; \\ \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix}_{\alpha} & \text{otherwise.} \end{cases} \tag{4.24}$$

This has the same purpose as the fuzzy unapplication operation used in fuzzy $\mathcal{FOT}$ generalization judgments: identify in the prior pair of tag maps (γ_1, γ_2) whether or not they already map a common tag (X) to the roots of the pair of ψ-terms to be generalized (ψ_1, ψ_2) each at, respectively, approximation α_1 and α_2. If so, the result of the unapplication is the pair made of the same ψ-term rooted in X ($X : \ldots$) at a posterior approximation degree equal to the conjoined value of the prior approximation degree α and those; *i.e.*, $\alpha \wedge \alpha_1 \wedge \alpha_2$; if not, it is the original pair of ψ-terms (ψ_1, ψ_2) at the unchanged prior approximation degree.

This rule basically states that generalizing two ψ-terms $\psi_1 \stackrel{\text{def}}{=} X : s\,(f_i \rightarrow \psi_i)_{i=0}^{m}$ and $\psi_2 \stackrel{\text{def}}{=} Y : t\,(g_j \rightarrow \xi_j)_{j=0}^{n}$ results in the ψ-term $\psi_1 \vee \psi_2 \stackrel{\text{def}}{=} Z : s \vee t\,(h_k \rightarrow \chi_k)_{k=0}^{p}$, where the set of features of the resulting ψ-term is the intersection of the corresponding sets of features of ψ_1 and ψ_2 (*i.e.*, the corresponding features they have in common), and Z is a new tag name.

As was the case for $\mathcal{FOT}$s, note that fuzzy $\mathcal{OSF}$ unapplication defined by Equation (4.24) returns a pair of terms and a (possibly lesser) approximation degree, unlike crisp unapplication

defined by Equation (4.12) that returns only a pair of terms. Because of this, when we write a fuzzy $\mathcal{OSF}$ generalization judgment such as:

$$\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash_{\alpha} \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \psi \begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix}_{\beta} \tag{4.25}$$

as we do in Rule **FUZZY UNEQUAL TAGS**, this is shorthand to indicate that the posterior similarity degree β is *at most* the one returned by the fuzzy $\mathcal{OSF}$ unapplication $\begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}$. Formally, the notation of the fuzzy $\mathcal{OSF}$ generalization judgment (4.25) is equivalent to:

$$\begin{pmatrix}\psi'_1\\ \psi'_2\end{pmatrix}_{\beta'} \stackrel{\text{def}}{=} \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \uparrow_{\alpha} \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}_{\beta'} \vdash \begin{pmatrix}\psi'_1\\ \psi'_2\end{pmatrix} \psi \begin{pmatrix}\gamma'_1\\ \gamma'_2\end{pmatrix}_{\beta} \tag{4.26}$$

for some β' such that $\beta \leq \beta' \leq \alpha$. This is because a fuzzy $\mathcal{OSF}$ term unapplication invoked while proving the validity of a fuzzy $\mathcal{OSF}$ generalization judgment may require, by Expression (4.24), lowering the *prior* approximation degree of the judgment. This is therefore applicable to pairs of subterms having some features in common. It consists in generalizing each of the corresponding pairs of subterms under all common features. Note that this can be done in any order, as long as each subterm judgment is validated with its pair of prior tag maps equal to its pair of posterior tag maps.

THEOREM 4.5 (CORRECTNESS OF FUZZY $\mathcal{OSF}$ GENERALIZATION) *The process of using any applicable constrained Horn clause shown in Figure 4.16 as long as one applies starting with the fuzzy $\mathcal{OSF}$ judgment to establish:*

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix}_{1.0} \vdash \begin{pmatrix}\psi_1\\ \psi_2\end{pmatrix} \psi \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}_{\alpha}$$

to prove this judgment's validity always terminates with $\psi = \psi_1 \wedge_{\alpha} \psi_2$, together with $\gamma_1 : \textbf{\textit{Tags}}(\psi) \to \textbf{\textit{Tags}}(\psi_1)$ and $\gamma_2 : \textbf{\textit{Tags}}(\psi) \to \textbf{\textit{Tags}}(\psi_2)$ such that $\psi_1 = \gamma_1(\psi)$ and $\psi_2 = \gamma_2(\psi)$.

PROOF Correctness of Rule **FUZZY UNEQUAL TAGS** is established inductively. That is, we must prove that if we assume that all the prior fuzzy judgments of this rule are valid under all the rule's side conditions, then its posterior fuzzy judgment is valid.

[To be completed later. . .] □

4.2.4 Implementation

An $\mathcal{OSF}$ constraint ϕ in solved form is always satisfiable in a canonical interpretation structure; *viz.*, the $\mathcal{OSF}$ graph algebra Ψ [23]. As a consequence, the $\mathcal{OSF}$ constraint normalization rules yield a decision procedure for the satisfiability of $\mathcal{OSF}$ constraints. This decision procedure is also operationally efficient [16]. One important reason for its efficiency is that computing sort

intersection as specified by Rule SORT INTERSECTION can be done in constant time by encoding sorts as binary vectors as shown in [15]. This results in tremendous speed performance when compared to encoding a class taxonomy's partial order using First-Order Logic monadic implication, even when resorting to proof "*memoing*" as done, for example, in [107], since this requires dynamically memorizing arbitrary proofs, thereby facing a hefty overhead price both in space and time. Indeed, resorting to bitvector-encoded ordered sorts rather than monadic implications is the key providing immediate deductive response thanks to static transitive closure on the "`is-a`" ordering and static consistent typing propagation of features to subtypes (see [14] and [33]).

However, isn't this valuable implementation trick lost with fuzzy, rather than binary, truth values? Some is, clearly, though not totally as we discuss next; and the gain fortunately outweighs the loss.

§ CLOSURE OF DECLARED FUZZY TAXONOMIES

In the crisp case, declaring an ordering on sorts defines a set of pairs. The complete ordering itself is then generated as the reflexive-transitive closure of this declared set of pairs ("$s_1 \preceq s_2$") when these are consistent (*e.g.*, when there is no cycle) or cycles are detected and reported [14]. This is taken to great advantage to compile it statically for the efficient computation of Boolean lattice operations on sorts when each sort is represented as a bit vector of as many bits as there are sorts, and carries a bit for each index of a sort it subsumes. Thus, the time and space complexity of all three Boolean lattice operations is quasi-constant on the size of the taxonomy, since this amounts to compiling each sort into a native binary word of size equal to the total number of sorts [15].

For a fuzzy subsumption ordering on sorts, the same kind of reflexive-transitive closure may be statically computed. However, the information carried by each pair of the fuzzy relation is no longer $\{0, 1\}$-valued but $[0.0, 1.0]$-valued (the value of the similarity degree α in declaring "$s_1 \preceq_\alpha s_2$"), and may no longer be represented as a bit. Now, instead of a bit vector, it is a fuzzy-bit vector; *i.e.*, a vector of real values in the closed interval $[0.0, 1.0]$ representing the fuzzy set $\{\, \alpha/s \mid \alpha \in [0.0, 1.0] \text{ for all } s \in \mathcal{S} \,\}$. The bitwise Boolean operations on bit-vectors are now fuzzified into $\wedge$, $\vee$, and $\alpha \to (1.0 - \alpha)$ on fuzzy set elements' fuzzy weights.[21] Each of these operations works on fuzzy sets to yield the fuzzy set of sorts obtained from applying the operation to the corresponding truth values of each sort. Namely:

$$X \wedge Y \stackrel{\text{def}}{=} \{\, (\alpha \wedge \beta)/s \mid \alpha/s \in X \text{ and } \beta/s \in Y, \text{ for all } s \in \mathcal{S} \,\} \tag{4.27}$$

$$X \vee Y \stackrel{\text{def}}{=} \{\, (\alpha \vee \beta)/s \mid \alpha/s \in X \text{ and } \beta/s \in Y, \text{ for all } s \in \mathcal{S} \,\} \tag{4.28}$$

$$\overline{X} \stackrel{\text{def}}{=} \{\, (1.0 - \alpha)/s \mid \alpha/s \in X, \text{ for all } s \in \mathcal{S} \,\} \tag{4.29}$$

for all X and Y fuzzy sets over a reference set of sorts $\mathcal{S}$. This is also the case with a similarity degree α and a fuzzy set X over $\mathcal{S}$:

$$\alpha \wedge X \stackrel{\text{def}}{=} \{\, (\alpha \wedge \beta)/s \mid \beta/s \in X, \text{ for all } s \in \mathcal{S} \,\} \tag{4.30}$$

[21] We use the notation "$x \to e$" to denote a nameless function associating the expression e to the argument x; *i.e.*, what the adepts of Functional Programming write as $\lambda x.e$ (here, $\lambda\alpha.(1.0 - \alpha)$) and call a functional abstraction or λ-expression.

and:

$$\alpha \vee X \stackrel{\text{def}}{=} \{ (\alpha \vee \beta)/s \mid \beta/s \in X, \text{ for all } s \in \mathcal{S} \}. \tag{4.31}$$

Note that we seldom need to represent explicitly a 0.0-similarity degree fuzzy element (*i.e.*, of the form $0.0/s$) and neither do we need to store it explicitly in a fuzzy set representation. In particular, in all the foregoing definitions given as Equation (4.27)–Equation (4.31), by " *for all* $s \in \mathcal{S}$" it is assumed that whenever $\_/s \notin X$, for some sort $s \in \mathcal{S}$ and fuzzy set X on $\mathcal{S}$, this is formally equivalent to $0.0/s \in X$.

It will always be assumed that a top sort ("$\top$") and a bottom sort ("$\bot$") are implicitly declared such that:

$$s \preceq_{1.0} \top \tag{4.32}$$

and,

$$s \neq \top \Rightarrow \top \preceq_{0.0} s \tag{4.33}$$

as well as:

$$\bot \preceq_{1.0} s \tag{4.34}$$

and,

$$s \neq \bot \Rightarrow s \preceq_{0.0} \bot \tag{4.35}$$

for all sorts $s \in \mathcal{S}$, in order to express respectively that there is no fuzziness in the sort ordering of $\top$ as the greatest (and all-encompassing) sort, and $\bot$ as the least (and all-excluding sort).

In [15] and [14], the encoding of crisp-ordered sorts as bit vectors is given in pseudo-code as a reflexive-transitive closure of the set of pairs of sort declarations of the form "$s_i \preceq s_j$." As expected, the process of propagating the similarity degrees declared in the fuzzy partial order of sorts is also a reflexive-transitive closure procedure. One will easily see that it is a direct homomorphic adaptation of the bit-vector procedure reviewed in [14] obtained by transforming the Boolean bit-vector representation and operations into their homomorphic fuzzy-set generalizations. It is given as the pseudocode procedure `CloseFuzzyTaxonomy` expressed as Algorithm 1.

The class `Sort` is the type representing partially-ordered symbols making up a concept taxonomy. We will also assume that known sorts are stored in a global (static) hash table, called `taxonomy`, associating strings (sort names) to `Sort` objects. A global (static) method `getSort(String)` will return a sort given its name.

The class `Sort` has a field called "`children`" of type `Set⟨Sort,double⟩` containing, for any sort, the sets of sorts that are its immediate children in the taxonomy, each paired with a non-zero similarity degree. Thus, for every sort object, this set is filled with sorts by processing fuzzy "$\preceq$" expressions of the form $s_1 \preceq_\alpha s_2$ used to declare that sort s_1 is subsumed by (or is a subsort of) sort s_2 with similarity degree $\alpha \in (0.0, 1.0]$; namely, $(s_1, \alpha) \in s_2$.`children`. The class `Sort` has another field called "`parents`" of type `Set⟨Sort⟩` containing, for any sort, the sets of sorts that are its immediate parents in the taxonomy. There is no need to record the similarity degrees as well in the `parents` sets because the similarity degrees will only be accessed through the `children` sets while closing a fuzzy taxonomy.

In addition, the class `Sort` has:

```
1  procedure CloseFuzzyTaxonomy ()
2      Set⟨Sort⟩ layer ← ⊥.parents;
3      while layer ≠ ∅ do
4          foreach Sort s ∈ layer do
5              s.fuzzyset ← {1.0/s} ∨ ⋁_{α/u∈s.children}(α ∧ u.fuzzyset);
6              s.closed ← true;
7          end
8          layer ← ⋃_{s∈layer} s.parents;
9          foreach s ∈ layer do
10             if ∃_/u ∈ s.children such that ¬u.closed then
11                 layer.remove(s);
12             end
13         end
14     end
15 end
```

Algorithm 1: Encoding of a fuzzy sort taxonomy as fuzzy-set codes

- an integer field called "`index`" that is a sort's unique characteristic rank in the array `taxonomy` containing all the sorts;
- a field called "`fuzzyset`" of type `Set⟨Sort,double⟩` initialized to the empty fuzzy set (*i.e.*, equivalent to all pairs of distinct declared sorts having 0.0 similarity degree); this represents the fuzzy set computed by reflexive-transitive closure. Upon completion of the closure, it ends up containing, for each sort $s_i \in$ `taxonomy`, the similarity degree $\alpha_{ij} \in (0.0, 1.0]$ of its $\preceq$ relationship with all sort $s_j \in$ `taxonomy` (*i.e.*, such that $s_i \preceq_{\alpha_{ij}} s_j$);
- a `Boolean` field called "`closed`" indicating whether this sort has been closed or not (so it is initially set to **false**).

§ Optimizing Closure and Lattice Operations

There is an immediate issue that we should keep in mind with using the foregoing "sort-as-fuzzy-set" representation and the closing procedure on these fuzzy sets. Namely, while Algorithm 1 is clearly formally correct as a lattice-homomorphic image of the crisp case, the motivation for casting sorts into the Boolean lattice of bit-vector codes seems compromised in the new representation of sorts as fuzzy sets exposed in [15], and used in [14] and in [10]. After closing it, a sort's bit-vector represents the set of its lower bounds. Indeed, this enabled optimizing set-lattice operations on ordered set-denoting sorts (very fast operations on bit vectors), with a minimal sort representation (a bit vector being essentially a non-negative integer), which can be further compacted using a given declared ***is-a*** ordering's specific topology [15]. With a fuzzy partial-order, however, a sort is no longer identified with a bit vector but with a $(0.0, 1.0]$-fuzzy set. Therefore, a

compact fuzzy-set representation upon which an efficient intersection operation may be computed must be provided in order to minimize impairing the efficient-implementation motivation.

We discuss here a sensible data-structure representation for the fuzzy set encoding a fuzzy-ordered sort in a finite set of a declared fuzzy taxonomy, supporting a better-than-*naïve* implementation of its lattice operations.

We shall call "*reference base*" the set of minimal upper bounds of $\bot$; namely, the set of sorts in $\bot$.`parents`, the first `layer` in Algorithm 1. We may also refer to the reference base as the set of instances (*i.e.*, each instance identifies a singleton-denoting sort).

Note that in Algorithm 1, the class `Sort`'s field `fuzzyset` is actually a fuzzy set where the fuzzy elements are pairs α/s where s may be any sort in `taxonomy`, not just a sort in the reference base. However, each sort formally denotes a fuzzy distribution on this reference base. So we may also find it useful to identify the reference base as a global array called `base` of N non-negative integers. This number N is the number of elements in the reference base; *viz.*, $N \stackrel{\text{def}}{=} |\bot.\texttt{parents}|$. Each value `base[i]`, $\texttt{i} = 1, \ldots, N$ is the index of a sort in the static hash table `taxonomy` that is minimal (*i.e.*, in $\bot$.`parents`). Hence, rather than a field `fuzzyset`, the class `Sort` is given a field called `fuzzybase` to represent this fuzzy set as an array of $N \leq \texttt{taxonomy.size()}$ of $[0.0, 1.0]$-values for each index in `base`. In other words, for a sort s, s.`fuzzybase` is an array of N similarity degrees and s.`fuzzybase[i]` is the similarity degree of `base[i]`.

For any sort `s`, the array `s.fuzzybase` is approximated by the binary vector we shall define as a new field of type `BitCode` for the class `Sort` called `crispvalue`, a bit vector such that:[22]

$$\texttt{s.crispvalue}[i] \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \texttt{s.fuzzybase}[i] > 0; \\ 0 & \text{otherwise.} \end{cases}$$

This information is therefore straightforward for any closed fuzzy sort taxonomy and can be used in the abstract interpretation of the three fuzzy Boolean lattice operations on sorts to restrict enumeration of a fuzzy set's elements only to non-zero indices using the bit-vector operations defined in [15] and [14].

4.3 Recapitulation

In this chapter, we reviewed the $\mathcal{OSF}$ constraint formalism and proposed fuzzy $\mathcal{OSF}$ constraint normalization systems for unification and generalization of $\mathcal{OSF}$ graphs that take into account declared similarity of meanings among sort symbols.

Since an $\mathcal{OSF}$ term is just linear syntax for representing a rooted sorted graph, and since this representation was shown to be equivalent to expressing such a graph as a logical $\mathcal{OSF}$ constraint, fuzzy interpretation of $\mathcal{OSF}$ term lattice operations (*viz.*, unification and generalization) can be formulated as fuzzy $\mathcal{OSF}$-constraint solving. Solving these constraints computes the most general tag substitutions and the most general fuzzy approximation degree where these constraints admit solutions. Different kinds of sort similarities (with or without aligned features) can thus be used

[22] This is representable, for example, as a Java class such as `hlt.osf.util.BitCode`, which extends the standard Java class `java.util.BitSet`.

effectively for approximation in applications that typically rely on identifying data and knowledge structures as labeled feature graphs.

Chapter 5

Version of April 10, 2020

Constructor Similarity Modulo Schema Alignment

In the previous two chapters, we considered two possible ways the mutually corresponding structure of two terms operated on (either $\mathcal{FOT}$s or $\mathcal{OSF}$ terms) is identified for similar constructors. In the case of $\mathcal{FOT}$s, we considered the case when argument positions are to be aligned thanks to an injective mapping from the argument-position set of the functor of lesser arity to the one of greater arity. However, when only *partial* argument-position mappings are defined between similar functors, the rules presented there do not apply without necessary consistency conditions defining what it means to be "*partially similar*." Regarding $\mathcal{OSF}$ terms, we only considered the case when features of similar sorts are identical. For non-aligned similar sorts (*i.e.*, for similar sorts where similiraty of $\mathcal{OSF}$ structure identifies features by different names), the rules we gave there do not apply. This, however, is a major requirement when comparing structures obeying different feature schemas.

In Section 5.1, we propose a solution for the issue of partial-argument constructor similarity. In Section 5.2, we show how this solution also resolves the issue of differing feature schemas for similar sorts.

5.1 $\mathcal{FOT}$ Argument Alignment

In Chapter 3, we gave declarative presentations for three lattice structures over $\mathcal{FOT}$s (one crisp and two fuzzy) in the form of axioms and rules. These axioms and rules specify the six corresponding dual lattice operations as constraints in these algebraic structures. An executable semantics for each operation is thus obtained for free as constraint solving. The latter may be summarized as follows, for each of the three $\mathcal{FOT}$ lattice structures:[1]

1. **for conventional signatures** (no operator similarity besides identity):

   — we presented the declarative $\mathcal{FOT}$ unification rules due to Herbrand and to Martelli & Montanari;

[1] We use the "✓" check symbol to indicate what items are contribution of this work.

✓ we provided a declarative constraint-based version of generalization equivalent to the original procedural methods due to Reynolds and Plotkin;

2. **for signatures with "weak" similarity** (all pairs of similar operators have the same number and order of arguments):

   — we presented "weak" fuzzy unification as constraint normalization using declarative rules due to Maria Sessa;

   ✓ we provided a "weak" fuzzy generalization as a constraint solving using a declarative specification for the dual operation of Sessa's "weak" unification;

3. **for signatures with possibly misaligned similarity** (similar operators possibly with different number or order of arguments):

   ✓ we extended the above constraint-driven declarative "weak" fuzzy unification to $\mathcal{FOT}$ s with possible different/mixed arities;

   ✓ we extended the above constraint-driven declarative "weak" fuzzy generalization of $\mathcal{FOT}$s with possible different/mixed arities.

This last pair of lattice operations on $\mathcal{FOT}$ modulo a similarity involving operators with misaligned or unordered arguments extends the previous pair of "weak" operations given argument maps specified for similar operators. That is, a similar pair of functors has a similarity degree as well as an injective argument-realigning map for each pair of operators in the signature. If unspecified, this map is the identity from the term with less arguments to the one with more arguments. In effect, this third lattice of $\mathcal{FOT}$s is closer to permit Fuzzy-Logic Programming querying with misaligned databases, or more generally Information Retrieval (using fuzzy unification) and Approximate Knowledge Acquisition (using fuzzy generalization) over heterogeneous but similar data models. In the remainder of this section, we will develop yet another lattice of $\mathcal{FOT}$s which is even closer to such models in that it allows an even more expressive similarity between functors than the ones we presented above, all of which are again special cases of this more generic $\mathcal{FOT}$ similarity.

The third $\mathcal{FOT}$ lattice above, being the most expressive of the three, tolerates similarity pairs of functors with different arities and assumes given a similarity matrix $\approx$ indexed by the functor signature Σ and for each pair of functors f/m and g/n such that $0 \leq m \leq n$, an injective map $\mu_{fg} : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$ satisfying consistency conditions (3.16), (3.17), and (3.18) on page 44. However, this lattice, although less constraining than the former, still requires that μ_{fg} be a *total function* map associating to *each* argument position in $\{1, \ldots, m\}$ a unique argument position in $\{1, \ldots, n\}$. This constraint, as we see next, may be relaxed to accommodate similar functors with *partial* argument alignments.

Example 5.1 Partial-map non-aligned similar functors — Consider two functors $\mathtt{foo} \in \Sigma_5$ and $\mathtt{bar} \in \Sigma_4$, and a non-zero approximation degree $\alpha \in (0.0, 1.0]$, where this similarity may be homomorphically extended from these functors to terms they construct *only* when, at this approximation degree α, $\mathtt{foo}$'s 3^{rd} argument is similar to $\mathtt{bar}$'s 4^{th} argument, and when $\mathtt{foo}$'s 4^{th} argument is similar to $\mathtt{bar}$'s

2^{nd} argument. This means that there are two mutually inverse partial bijective maps between the argument positions of functors foo and bar specifying which argument position of one corresponds to which unique argument position of the other; *viz.*, $\mu^{\alpha}_{foo,bar} : \{3,4\} \to \{1,2,3,4\} = \{3 \mapsto 4, 4 \mapsto 2\}$ and $\mu^{\alpha}_{bar,foo} : \{2,4\} \to \{1,2,3,4,5\} = \{2 \mapsto 4, 4 \mapsto 3\}$. We shall denote such a partial non-aligned functor similarity with the symmetric pairs $foo \approx^{\mu^{\alpha}_{foo,bar}}_{\alpha} bar$ and $bar \approx^{\mu^{\alpha}_{bar,foo}}_{\alpha} foo$ as in the $\mathcal{FOT}$ similarity Expression (5.1).

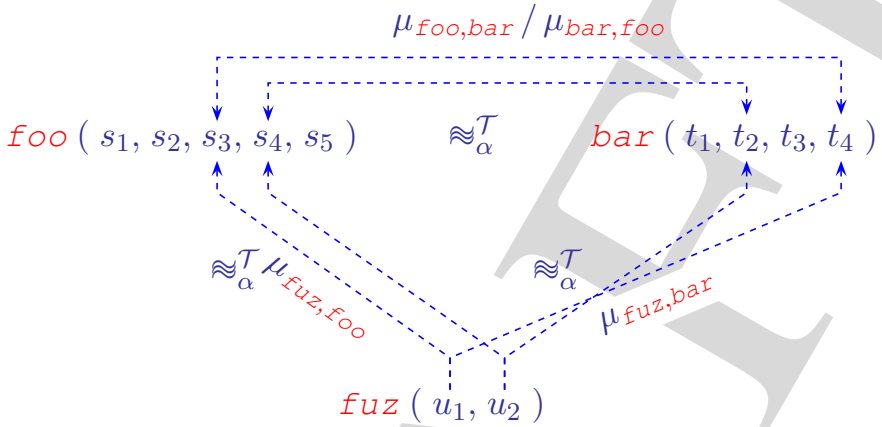
$$\begin{array}{ccc} & \mu_{foo,bar} \,/\, \mu_{bar,foo} & \\ \downarrow & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \downarrow \\ foo\,(\,s_1,\,s_2,\,s_3,\,s_4,\,s_5) & \approx^{\mathcal{T}}_{\alpha} & bar\,(\,t_1,\,t_2,\,t_3,\,t_4\,) \\ \uparrow & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \uparrow \\ & \mu_{foo,bar} \,/\, \mu_{bar,foo} & \end{array} \tag{5.1}$$

The formalism we have developed in the previous sections cannot apply for such non-aligned similar functors with only partial argument-position maps because the assumptions we made for the unification rules of Figure 3.11 to be correct do not hold. Indeed, these rules work because whenever an equation between two constructed terms has a term of lesser arity on the right, Rule **FUZZY EQUATION ORIENTATION** swaps its sides into an equation with the lesser-arity term on the left. And, for a such an equation as the latter, Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** replaces this equation only by equations between subterms at corresponding positions, taking *all* arguments of the lesser-arity *from position* 1, and *all the way up to its full arity*. This is no longer possible with partial maps between two similar functors' non-aligned argument positions. Indeed, a lesser arity functor's partial argument maps may not be defined for argument position 1, nor for consecutive positions, nor up to the functor's arity. For the same reason, the fuzzy generalization rules of Figure 3.14 will not apply either. Indeed, both Rule **FUNCTOR/ARITY SIMILARITY LEFT** and **FUNCTOR/ARITY SIMILARITY RIGHT** specify the least generalizer to be constructed using the least-arity functor and the generalizers of all its arguments; this, clearly, is no longer possible with partial argument-position maps between the subterms of non-aligned similar functors.

However, in some specific situations, it may be possible to come back to the previously studied $\mathcal{FOT}$ lattice — which requires that in any equation between two constructed terms one of the two terms always be a lesser-arity functor's with a total injective map from the set of all its argument positions to a subset of the larger-arity functor's term's set of argument positions. Indeed, the justification for the need of reorienting some equations using Rule **FUZZY EQUATION ORIENTATION** of Figure 3.11 is that Rule **FUZZY NON-ALIGNED-ARGUMENT TERM DECOMPOSITION** may then easily choose a term's functor's similarity class representative as the one of least arity; *viz.*, the one on the left-hand side. However, these assumptions do not hold in general in our new situation. Indeed, this is possible only if each functor similarity class at approximation degree α happens to have a least-arity functor term representative with *total* argument-position maps to all other members of the similarity class at this approximation degree.

Example 5.2 Composing partial non-aligned argument-position map for similar functors — Let us elaborate on Example 5.1: in addition to the similar functors "$foo/5$" and "$bar/4$" at a given approximation degree $\alpha \in (0.0, 1.0]$ with *partial* argument maps $\mu^{\alpha}_{foo,bar} : \{3, 4\} \to \{2, 4\} = \{3 \mapsto$

$4, 4 \mapsto 2\ \}$ and $\mu^{\alpha}_{bar,foo} : \{\, 2, 4 \,\} \mapsto \{\, 3, 4 \,\} = \{\, 2 \mapsto 4, 4 \mapsto 3 \,\}$ so that $\mu^{\alpha}_{foo,bar} = (\mu^{\alpha}_{,bar,foo})^{-1}$, and $\mu^{\alpha}_{bar,foo} = (\mu^{\alpha}_{foo,bar})^{-1}$), there also exists a functor "$fuz/2$" that is similar at this approximation degree α to both $foo/5$ and $bar/4$ with *total* argument maps $\mu^{\alpha}_{fuz,foo} : \{1, 2\} \to \{1, 2, 3, 4, 5\} = \{\, 1 \mapsto 3, 2 \mapsto 4 \,\}$ and $\mu^{\alpha}_{fuz,bar} : \{1, 2\} \to \{1, 2, 3, 4\} = \{\, 1 \mapsto 4, 2 \mapsto 2 \,\}$, in such a way that $\mu^{\alpha}_{fuz,foo} = \mu^{\alpha}_{bar,foo} \circ \mu^{\alpha}_{fuz,bar}$ and $\mu^{\alpha}_{fuz,bar} = \mu^{\alpha}_{foo,bar} \circ \mu^{\alpha}_{fuz,foo}$. This, of course, requires that $\mathbf{ran}(\mu^{\alpha}_{fuz,foo}) = \mathbf{dom}(\mu^{\alpha}_{foo,bar})$ and also that $\mathbf{ran}(\mu^{\alpha}_{fuz,bar}) = \mathbf{dom}(\mu^{\alpha}_{bar,foo})$, as well as $\mathbf{ran}(\mu^{\alpha}_{fuz,foo}) = \mathbf{ran}(\mu^{\alpha}_{bar,foo})$ and $\mathbf{ran}(\mu^{\alpha}_{fuz,bar}) = \mathbf{ran}(\mu^{\alpha}_{foo,bar})$, as in the term similarity Expressions (5.2).



(5.2)

We show next how this can be accommodated in our formalization. In the following, the set denoted as $\{1, \ldots, n\}$ with $n = 0$ is always equal to the empty set $\emptyset$. In other words, for any $n \in \mathbb{N}$, $\{1, \ldots, n\} = \emptyset$ if and only if $n = 0$.

DEFINITION 5.1 (PARTIAL-MAP NON-ALIGNED SIMILAR FUNCTORS) *Let $m \geq 0$, $n \geq 0$; two functors $f \in \Sigma_m$ and $g \in \Sigma_n$ are said to be partial-map non-aligned similar functors at approximation degree $\alpha \in [0.0, 1.0]$ whenever:*

1. *there is a set $\mathcal{D}^{\alpha}_{fg} \subseteq \{1, \ldots, m\}$ of argument positions of f and a set $\mathcal{D}^{\alpha}_{gf} \subseteq \{1, \ldots, n\}$ of argument positions of g such that $|\mathcal{D}^{\alpha}_{fg}| = |\mathcal{D}^{\alpha}_{gf}|$; and,*

2. *there exist a pair of mutually inverse bijections $\mu^{\alpha}_{fg} : \mathcal{D}^{\alpha}_{fg} \to \{1, \ldots, n\}$ and $\mu^{\alpha}_{gf} : \mathcal{D}^{\alpha}_{gf} \to \{1, \ldots, m\}$ such that $\mathbf{ran}(\mu^{\alpha}_{fg}) = \mathcal{D}^{\alpha}_{gf} = \mathbf{dom}(\mu^{\alpha}_{gf})$ and $\mathbf{ran}(\mu^{\alpha}_{gf}) = \mathcal{D}^{\alpha}_{fg} = \mathbf{dom}(\mu^{\alpha}_{fg})$.*

Note that it is possible in the above definition that $\mathbf{dom}(\mu^{\alpha}_{fg}) = \emptyset$ or $\mathbf{ran}(\mu^{\alpha}_{fg}) = \emptyset$. The former means that no argument of f need be similar to any argument of g, and the latter means that no argument of g need be similar to any argument of f. That is, in both cases, the similarity of terms they construct reduces to that of the functors, regardless of any subterms.

We shall always require, for any approximation degree $\alpha \in [0.0, 1.0]$ and any functor f, that $\mathcal{D}^{\alpha}_{ff} = \{1, \ldots, \mathbf{arity}(f)\}$, $|\mathcal{D}^{\alpha}_{ff}| = \mathbf{arity}(f)$, and $\mu^{\alpha}_{ff} = \mathbb{1}_{\{1, \ldots, \mathbf{arity}(f)\}}$. This means that pairs of the form $\langle f, f \rangle$ (*i.e.*, the diagonal) always have as argument-position map the *total identity* on *all* the argument positions of f at any approximation degree.

The case where at least one of any two similar functors has a total injective map of its argument positions into the other functor's is a special case of this. When this is so, argument-position maps are composable because *all* the positions in the range of a map are always in the domain of any

map from this functor to another (of greater arity). With partial maps however, this may no longer be possible.

Example 5.3 Non-composable inconsistent partial-map non-aligned functors — In addition to the functors $foo/5$ and $bar/4$ of Example 5.2 where $\mu^{\alpha}_{foo,bar} = \{\, 3 \mapsto 4,\ 4 \mapsto 2 \,\}$ (and $\mu^{\alpha}_{bar,foo} = \{\, 2 \mapsto 4,\ 4 \mapsto 3 \,\}$), consider the functor $biz/4$ and the map $\mu^{\alpha}_{bar,biz} : \{1 \mapsto, 2,\ 3 \mapsto 4\}$. These maps will not be composable simply because $\mathbf{ran}(\mu^{\alpha}_{foo,bar}) = \{2, 4\}$ and $\mathbf{dom}(\mu^{\alpha}_{bar,biz}) = \{1, 3\}$ have no elements in common. That is, $\mathbf{ran}(\mu^{\alpha}_{foo,bar}) \cap \mathbf{dom}(\mu^{\alpha}_{bar,biz}) = \emptyset$.

And even if they had compatible domain and range, say if $\mu^{\alpha}_{bar,biz} : \{\, 1 \mapsto 2,\ 2 \mapsto 4 \,\}$, but $\mu^{\alpha}_{foo,biz} : \{\, 3 \mapsto 3,\ 4 \mapsto 4 \,\}$, this would mean that the composition $\mu^{\alpha}_{bar,biz} \circ \mu^{\alpha}_{foo,bar}$ and the map $\mu^{\alpha}_{foo,biz}$ also disagree as we have, on one hand:

$$\mu^{\alpha}_{foo,biz} \ = \ \{\, 3 \mapsto 3, 4 \mapsto 4 \}$$

and on the other hand:

$$\begin{aligned} \mu^{\alpha}_{bar,biz} \circ \mu^{\alpha}_{foo,bar} &= \{\, 3 \mapsto \mu^{\alpha}_{bar,biz}(\mu^{\alpha}_{foo,bar}(3)), 4 \mapsto \mu^{\alpha}_{bar,biz}(\mu^{\alpha}_{foo,bar}(4)) \,\} \\ &= \{\, 3 \mapsto \mu^{\alpha}_{bar,biz}(4), 4 \mapsto \mu^{\alpha}_{bar,biz}(2) \,\} \\ &= \{\, 3 \mapsto ?, 4 \mapsto 4 \,\} \end{aligned}$$

which is compositionally inconsistent, and thus $\mu^{\alpha}_{bar,biz} \circ \mu^{\alpha}_{foo,bar} \neq \mu^{\alpha}_{foo,biz}$.

And this is inconsistent also in the other direction as well, since $\mu^{\alpha}_{bar,foo} = \{\, 2 \mapsto 4,\ 4 \mapsto 3 \,\}$, $\mu^{\alpha}_{bar,biz} : \{\, 1 \mapsto 2,\ 2 \mapsto 4 \,\}$, and $\mu^{\alpha}_{foo,biz} : \{\, 3 \mapsto 3,\ 4 \mapsto 4 \,\}$, entail that the composition $\mu^{\alpha}_{foo,biz} \circ \mu^{\alpha}_{bar,foo}$ and the map $\mu^{\alpha}_{bar,biz}$ would also disagree. We have, on one hand:

$$\mu^{\alpha}_{bar,biz} \ = \ \{\, 1 \mapsto 2, 2 \mapsto 4 \}$$

and on the other hand:

$$\begin{aligned} \mu^{\alpha}_{foo,biz} \circ \mu^{\alpha}_{bar,foo} &= \{\, 2 \mapsto \mu^{\alpha}_{foo,biz}(\mu^{\alpha}_{bar,foo}(2)), 4 \mapsto \mu^{\alpha}_{foo,biz}(\mu^{\alpha}_{bar,foo}(4)) \,\} \\ &= \{\, 2 \mapsto \mu^{\alpha}_{foo,biz}(4), 4 \mapsto \mu^{\alpha}_{foo,biz}(3) \,\} \\ &= \{\, 1 \mapsto ?, 2 \mapsto 4, 4 \mapsto 3 \,\} \end{aligned}$$

which, again, entails $\mu^{\alpha}_{foo,biz} \circ \mu^{\alpha}_{bar,foo} \neq \mu^{\alpha}_{bar,biz}$, and thus is compositionally inconsistent as well.

We next define formally the conditions for similar functor partial-alignment consistency of a signature to make argument-position maps always be composable at any given approximation degree.

DEFINITION 5.2 (CONSISTENT PARTIAL SIMILARITY OF NON-ALIGNED SIGNATURE) *Let* $\Sigma \stackrel{def}{=} \cup_{k \geq 0} \Sigma_k$ *be a functor signature, and let* $\approx\!\!\approx : \Sigma \times \Sigma \to [0.0, 1.0]$ *be a similarity on* Σ. *It is said that signature* Σ *is non-aligned admitting* $\approx\!\!\approx$ *as a consistent partial similarity whenever all the following statements hold:*

1. *all argument-position mappings conditions* (3.15)–(3.18) *are satisfied;*[2]

[2] See Page 43.

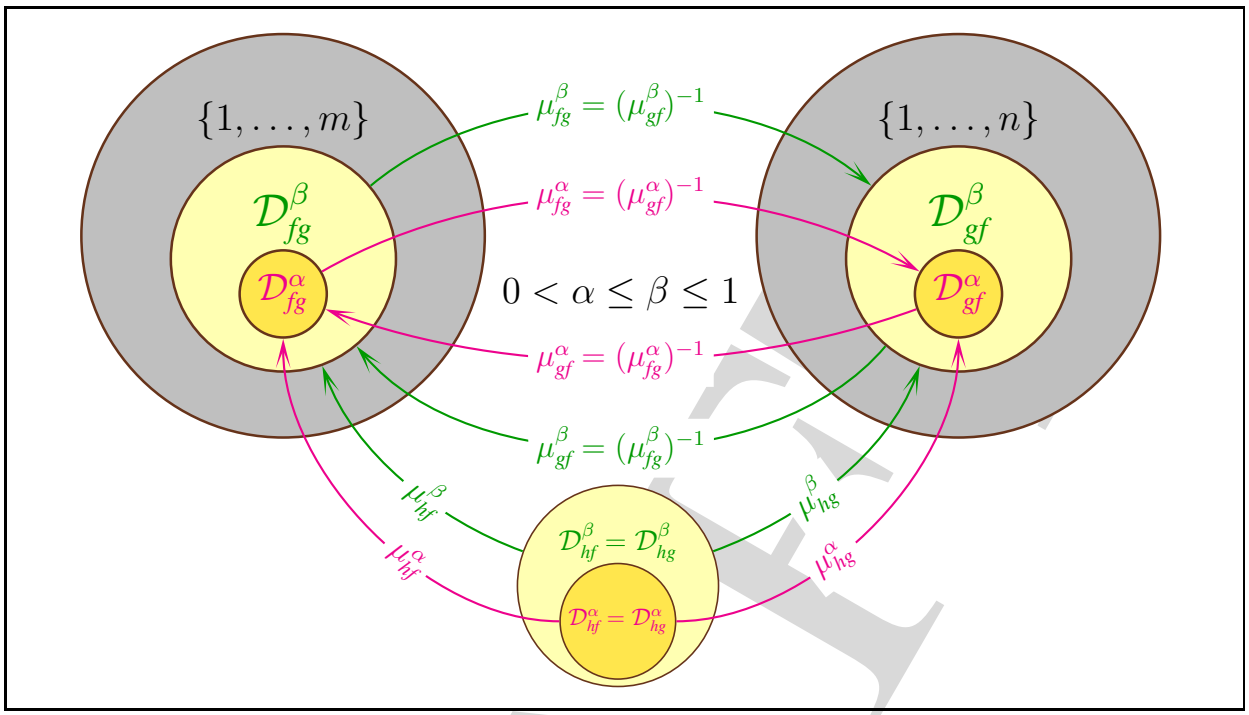


Figure 5.1: Partial-map non-aligned similar functors argument-map consistency diagram

2. *all pairs* $\langle f, g\rangle \in \Sigma_m \times \Sigma_n$ *of similar functors — i.e., when* $f \approxeq_\alpha g$ *with* $\alpha \in [0.0, 1.0]$ *— are partial-map non-aligned similar functors at approximation degree* α *as specified by Definition 5.1*;

3. *for any functors* $f \in \Sigma$ *and* $g \in \Sigma$, *and approximation degrees* $\alpha \in [0.0, 1.0]$ *and* $\beta \in [0.0, 1.0]$:

$$\alpha \leq \beta \Rightarrow \mathcal{D}^{\alpha}_{fg} \subseteq \mathcal{D}^{\beta}_{fg}\,; \tag{5.3}$$

4. *for all* $f \in \Sigma_m$, $g \in \Sigma_n$, $h \in \Sigma_\ell$, $m \geq 0$, $n \geq 0$, *and* $\ell \geq 0$:

$$\begin{cases} \textbf{\textit{ran}}(\mu^{\alpha}_{fg}) = \textbf{\textit{dom}}(\mu^{\alpha}_{gh}) \;\; (= \mathcal{D}^{\alpha}_{gh}) \\ \textbf{\textit{ran}}(\mu^{\alpha}_{fh}) = \textbf{\textit{ran}}(\mu^{\alpha}_{gh})\,; \end{cases} \tag{5.4}$$

and:

$$\begin{cases} \mu^{\alpha}_{hf} = \mu^{\alpha}_{gf} \circ \mu^{\alpha}_{hg} \\ \mu^{\alpha}_{hg} = \mu^{\alpha}_{fg} \circ \mu^{\alpha}_{hf}. \end{cases} \tag{5.5}$$

These conditions are concisely summarized as the commutative functional diagram of Figure 5.1.

COROLLARY 5.1 (COMPOSABILITY OF ARGUMENT-POSITION MAPS) *A non-aligned signature* Σ *with a consistent partial similarity* $\approxeq$ *satisfying all the conditions of Definition 5.2 are always consistently composable at any given approximation degree* $\alpha \in [0.0, 1.0]$.

PROOF We need to verify that, for any functors f, g, h and any $\alpha \in [0.0, 1.0]$:

$$\mathbf{ran}(\mu^{\alpha}_{fg}) \subseteq \mathbf{dom}(\mu^{\alpha}_{gh})$$

and,

$$\mathbf{dom}(\mu^{\alpha}_{fh}) = \mathbf{dom}(\mu^{\alpha}_{fg}) \text{ and } \mathbf{ran}(\mu^{\alpha}_{fh}) = \mathbf{ran}(\mu^{\alpha}_{gh})$$

and that, for all positions $i \in \mathbf{dom}(\mu^{\alpha}_{fg})$:

$$\mu^{\alpha}_{fh}(i) = \mu^{\alpha}_{gh}(\mu^{\alpha}_{fg}(i)).$$

All three properties can be verified to be immediate consequences of the conditions of Definition 5.2. □

The following is also a corollary of Definition 5.2 and Definition 5.1.

COROLLARY 5.2 (STABILITY OF PARTIAL SIMILARITY DOMAINS AND CLASSES) *Given any two functors f and g such that $f \approx_{\alpha} g$ at approximation degree $\alpha \in [0.0, 1.0]$, the size $|\mathcal{D}^{\alpha}_{fg}|$ of the set $\mathcal{D}^{\alpha}_{fg}$ is constant for fixed α; that is, $|[f]^{\alpha}_{\approx}| = |\mathcal{D}^{\alpha}_{fg}| = |\mathcal{D}^{\alpha}_{gf}| = |[g]^{\alpha}_{\approx}|$.*

PROOF This follows since, at any fixed approximation degree, all maps between functors in a similarity class are bijective and the fact that they all satisfy the properties specified in Definition 5.2 and Definition 5.1. □

DEFINITION 5.3 *The fuzzy relation $\approx^{\mathcal{T}}$ on $\mathcal{T}_{\Sigma,\mathcal{V}}$ is defined inductively as:*

1. $\forall X \in \mathcal{V}, \quad X \approx^{\mathcal{T}}_{1} X$;
2. $\forall X \in \mathcal{V}, \forall t \in \mathcal{T}$ *such that* $X \neq t$, $X \sim^{\mathcal{T}}_{0} t$ *and* $t \sim^{\mathcal{T}}_{0} X$;
3. *for* $m \in \mathbb{N}$, $n \in \mathbb{N}$, $f \approx_{\alpha} g$ *with* $\mu^{\alpha}_{fg} : \mathcal{D}^{\alpha}_{fg} \to \mathcal{D}^{\alpha}_{gf}$ *and* $s_i \approx^{\mathcal{T}}_{\alpha_i} t_{\mu^{\alpha}_{fg}(i)}$ *for all* $i \in \mathcal{D}^{\alpha}_{fg}$, *then:*

$$f(s_1, \ldots, s_m) \approx^{\mathcal{T}}_{(\alpha \wedge \bigwedge_{i \in \mathcal{D}^{\alpha}_{fg}} \alpha_i)} g(t_1, \ldots, t_n). \tag{5.6}$$

With this definition and the following theorem, we shall now have all the necessary formal tools to proceed as we did for the two previous $\mathcal{FOT}$ lattice structure constructions in the case where non-aligned Σ admits $\approx$ as a consistent partial similarity.

THEOREM 5.1 *The relation $\approx^{\mathcal{T}}$ defined by Definition 5.3 is a similarity relation on $\mathcal{T}$ the set of $\mathcal{FOT}$s.*

PARTIAL NON-ALIGNED SIMILAR TERM DECOMPOSITION

$$\left[f \approx^{\mu^{\beta}_{fg}}_{\beta} g;\ \ 0 \leq |\mathcal{D}^{\alpha\wedge\beta}_{fg}| = p,\ p \leq \min(m,n);\ \ \mathcal{D}^{\alpha\wedge\beta}_{fg} = \{d_1, \ldots d_p\} \right]$$

$$\frac{(\, E \cup \{\, f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n) \,\}\,)_{\alpha}}{(\, E \cup \{\, s_{d_1} \doteq t_{\mu^{\alpha\wedge\beta}_{fg}(d_1)}, \ldots, s_{d_p} \doteq t_{\mu^{\alpha\wedge\beta}_{fg}(d_m)} \,\}\,)_{\alpha\wedge\beta}}$$

Figure 5.2: Partial non-aligned similar $\mathcal{FOT}$ similar term decomposition rule

PARTIAL NON-ALIGNED FUNCTOR SIMILARITY

$$\left[f/m \approx_{\beta} g/n;\ \ \alpha_0 \stackrel{\text{def}}{=} \alpha \wedge \beta;\ \ h/p \in [f/m, g/n]_{\alpha_0};\ \ |\mathcal{D}^{\alpha_0}_{hf}| = |\mathcal{D}^{\alpha_0}_{hg}| = p \right]$$

$$\frac{\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha_0} \vdash \begin{pmatrix}s'_1\\ t'_1\end{pmatrix} u_1 \begin{pmatrix}\sigma^1_1\\ \sigma^1_2\end{pmatrix}_{\alpha_1} \ldots \begin{pmatrix}\sigma^{p-1}_1\\ \sigma^{p-1}_2\end{pmatrix}_{\alpha_{p-1}} \vdash \begin{pmatrix}s'_p\\ t'_p\end{pmatrix} u_p \begin{pmatrix}\sigma^p_1\\ \sigma^p_2\end{pmatrix}_{\alpha_p}}{\begin{pmatrix}\sigma_1\\ \sigma_2\end{pmatrix}_{\alpha} \vdash \begin{pmatrix}f(s_1, \ldots, s_m)\\ g(t_1, \ldots, t_n)\end{pmatrix} h(u_1, \ldots, u_p) \begin{pmatrix}\sigma^p_1\\ \sigma^p_2\end{pmatrix}_{\alpha_p}}$$

Figure 5.3: Partial non-aligned similar $\mathcal{FOT}$ generalization rule

From this, in the same manner as we did before, we shall derive a weaker subsumption preorder on $\mathcal{FOT}$s as well as adapt our previous sets of rules to specify the corresponding unification and generalization lattice operations for this preorder. This is what we present next.

The rules for unification of similar partial-map non-aligned $\mathcal{FOT}$s are those of Maria Sessa's weak unification (see Figure 3.7) where Rule **WEAK TERM DECOMPOSITION** is replaced with Rule **PARTIAL NON-ALIGNED TERM DECOMPOSITION** given in Figure 5.2. ***N.B.***: there is no need to re-orient a term equation as for total maps (see Figure 3.11). **Why?**

The rules for generalization of partial-map non-aligned similar $\mathcal{FOT}$s are those given in Figure 3.13 where Rule **SIMILAR FUNCTORS** is replaced with Rule **PARTIAL NON-ALIGNED FUNCTOR SIMILARITY** given in Figure 5.3, where, for $i = 1, \ldots, p$:

$$\begin{pmatrix}s'_i\\ t'_i\end{pmatrix}_{\beta_i} \stackrel{\text{def}}{=} \begin{pmatrix}s_{\mu^{\alpha_{i-1}}_{hf}(i)}\\ t_{\mu^{\alpha_{i-1}}_{hg}(i)}\end{pmatrix} \uparrow_{\alpha_{i-1}} \begin{pmatrix}\sigma^{i-1}_1\\ \sigma^{i-1}_2\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}\sigma^{i-1}_1\\ \sigma^{i-1}_2\end{pmatrix}_{\beta_i} \vdash \begin{pmatrix}s'_i\\ t'_i\end{pmatrix} u_i \begin{pmatrix}\sigma^i_1\\ \sigma^i_2\end{pmatrix}_{\alpha_i}.$$

N.B.: there is no differentiating left/right rules as for total maps (see Figure 3.14); only a single rule is needed. **Why?**

Automated signature completion Note that the unification and generalization rules above will work with non-aligned similar functors with partial argument-position maps as long as the conditions on the signature, the functor similarity, as well as all the corresponding partial argument alignment, satisfy the signature consistency conditions given in Definition 5.2 and illustrated in

Figure 5.1. These conditions are necessary to ensure *composability* of all partial argument maps at any approximation level $\alpha \in [0.0, 1.0]$. Indeed, it is composability of similar functor argument maps that ensures consistent transitivity of the functor similarity.

However, one may object to requiring signatures and their similarities to possess complete consistent partial maps as rather demanding. We now see how the process of verifying this to be true for a given signature and similarity can be automated. In fact, there are two questions that must be addressed:

1. Can a given signature with specified similarity and partial alignment maps *automatically* be either verified to meet these requirements, or proven inconsistent?
2. Can a consistent but incomplete signature be completed to a minimal consistent one containing it while respecting all its similaririties and argument alignment maps?

The good news is that the answer to both questions is "*yes.*"

1. There is a finite procedure to verify that for all pairs of transitive pairs of functors $\langle f, g\rangle$ and $\langle g, h\rangle$, all specified argument maps abide by necessary consistency conditions or to point out where this is violated, and why.
2. Should a signature be detected to be incomplete at a given approximation level $\alpha \in [0.0, 1.0]$ in the sense that some functor similarity class does not possess a least-arity representative with complete and consistent argument maps to all other functors in its class at level α, then either the signature can be completed with a new functor with this property with appropriate least (in terms of inclusion of sets of pairs) argument maps to all functors in its class, or an explicit counter-example can be given that shows why there is no consistent signature can complete this signature while respecting all argument maps.

This procedure may be performed at all approximation degrees $\alpha \in \textbf{DEGREES}^{\approxeq}$. Pseudocode specifying this completion procedure is given in Figure 5.4. It automatically completes an incomplete signature and a specified base set of similar functor pairs, together with some partial argument-position maps using a completion procedure on the signature and the similarity so that the signature may either be proven inconsistent, or completed with a similarity such that each similarity class at any approximation degree $\alpha \in \textbf{DEGREES}^{\approxeq}$ contains at least one representative functor with consistently aligned *total* argument-position maps to all members of the class.

Example 5.4 Non-aligned signature partial similarity completion — Consider a signature Σ in which the only pair of non-identical similar functors at a given similarity degree α are $f/4$ and $g/3$ such that $f \approxeq_\alpha^{\mu_{fg}^\alpha} g$ and $g \approxeq_\alpha^{\mu_{gf}^\alpha} f$ with mutually inverse injective partial argument-position maps $\mu_{fg}^\alpha = \{\langle 2,1\rangle, \langle 4,3\rangle\}$ and $\mu_{gf}^\alpha = \{\langle 1,2\rangle, \langle 3,4\rangle\}$, so that $\mathcal{D}_{fg}^\alpha \stackrel{\text{def}}{=} \{2,4\}$ and $\mathcal{D}_{fg}^\alpha \stackrel{\text{def}}{=} \{1,3\}$.

The similarity class $c = \{f, g\}$ does not have a least-arity functor class representative with total argument-position maps to all members of c. So, since $\{|\mathcal{D}_{fg}^\alpha| \mid f \in c,\ g \in c\} = 2$, the minimum value in this set is 2. So we add a new functor $h/2$ to Σ_2 with the *total* argument maps: $\mu_{hf}^\alpha = \{\langle 1, \mu_{gf}^\alpha(1)\rangle, \langle 2, \mu_{gf}^\alpha(3)\rangle\}$ and $\mu_{hg}^\alpha = \{\langle 1, \mu_{fg}^\alpha(2)\rangle, \langle 2, \mu_{fg}^\alpha(4)\rangle\}$; that is, $\mu_{hf}^\alpha = \{\langle 1,2\rangle, \langle 2,4\rangle\}$ and $\mu_{hg}^\alpha = \{\langle 1,1\rangle, \langle 2,3\rangle\}$.

And this is consistent by construction since $\mu_{hg}^\alpha = \mu_{fg}^\alpha \circ \mu_{hf}^\alpha$ and $\mu_{hf}^\alpha = \mu_{gf}^\alpha \circ \mu_{hg}^\alpha$ (as can be easily verified). So $h/2$ can be added to class c.

forall $\alpha \in$ **DEGREES**$^{\approxeq}$ and similarity class $c \in \Pi^{\approxeq}_{\alpha}$ **do**

if $\nexists$ a least-arity similarity class representative in c with *total* argument-position maps to *all* members of c

then

- add a *new* functor h/m to signature Σ_m such that:

$$m = \min\{|\mathcal{D}^{\alpha}_{fg}| \mid f \in c,\ g \in c\},$$

with least *total* consistent injective maps:

$$\mu^{\alpha}_{hf} : \{1, \ldots, m\} \to \{1, \ldots, n\}$$

for *all* $f/n \in c$; if not possible, Σ is inconsistent;

- add functor h/m to similarity class c;

Figure 5.4: Automated completion of partial-maps for non-aligned signature similarity

5.2 $\mathcal{OSF}$ Term Argument Alignment

The system we proposed in Chapter 4 fuzzified $\mathcal{OSF}$ subsumption and related fuzzy lattice operations on $\mathcal{OSF}$ terms always assume that similar sorts share indentically named features. We now consider the situation when this is not necessarily the case; *i.e.*, when sorts may also be similar modulo aligning the features of each pair of similar sorts.

§ FEATURE ARITY

We will assume that each sort $s \in \mathcal{S}$ has a "feature arity" $\mathbf{arity}(s) \in \mathbf{2}^{\mathcal{F}}$ that associates to the sort s a finite set of features. However, this must be consistent with feature inheritance. This is expressed formally by defining the mapping $\mathbf{arity} : \mathcal{S} \to \mathbf{2}^{\mathcal{F}}$ as the following lattice homomorphism from $\langle \mathcal{S}, \preceq, \curlywedge, \curlyvee \rangle$ to $\langle \mathbf{2}^{\mathcal{F}}, \supseteq, \cup, \cap \rangle$:

$$\mathbf{arity}(s \curlywedge t) = \mathbf{arity}(s) \cup \mathbf{arity}(t) \tag{5.7}$$

$$\mathbf{arity}(s \curlyvee t) = \mathbf{arity}(s) \cap \mathbf{arity}(t) \tag{5.8}$$

for all sorts s and t in $\mathcal{S}$. Note that the anti-monotonicity of sort subsumption and feature set inheritance, since Equation (5.7) and Equation (5.8) imply necessarily that:

$$s \preceq t \Rightarrow \mathbf{arity}(t) \subseteq \mathbf{arity}(s); \tag{5.9}$$

i.e., the more specific the sort, the larger its feature-arity set.

Again, note that the arity of a sort symbol is not a natural number as is the case of a $\mathcal{FOT}$'s functor where it stands for its number of arguments. The arity of a sort is a set of features that

denote the admissible field names of structure components rooted in a node of this sort. It is a set of symbols rather than a single natural number as in the case of $\mathcal{FOT}$s because order of a sort's features does not matter, and not all of them need occur attached to each occurrence of this sort. It may also be empty. Seeing a $\mathcal{FOT}$ as an $\mathcal{OSF}$ term, and thus seeing a function symbol $f/n \in \Sigma_n$ as a sort, $\textbf{arity}(f/n) \stackrel{\text{def}}{=} \{1, \ldots, n\}$, and the arity of a constant is the empty set.

§ Feature-alignment mappings

In the same manner as when comparing $\mathcal{FOT}$s using similar functors with misaligned argument positions, similar sorts may as well disagree on their feature sets as long as these correspond to one another's reciprocal feature mappings between the two sorts. By default (and up to now), when comparing two sorts s and t, every feature name is mapped to itself — *i.e.*, feature mapping from one sort to another is always the identity on $\mathcal{F}$ by default. But now, extending the same idea as for function symbols in $\mathcal{FOT}\!\int$, for any pair of sorts s and t in $\mathcal{S}$, we can assume defined a feature mapping $\pi_{st} : \mathcal{F} \to \mathcal{F}$ that satisfies the following properties.

- π_{st} is *injective* (*i.e.*, one-to-one); *i.e.*, for any pair of sorts s and t in $\mathcal{S}$:

  $$f \neq f' \to \pi_{st}(f) \neq \pi_{st}(f') \tag{5.10}$$

  for all *features* f and f' in $\mathcal{F}$;[3]

- π_{st} is the *identity almost everywhere on* $\mathcal{F}$ except on a finite (possibly empty) subset of features in $\textbf{arity}(s)$ and the non-identical image of this set is a (possibly empty) subset of features in $\textbf{arity}(t)$;

- it is *self-consistent*; *i.e.*,

  $$\pi_{ss} = \mathbb{1}_{\mathcal{F}} \tag{5.11}$$

  for all sorts s in $\mathcal{S}$;

- it is *inverse-consistent*; *i.e.*,

  $$\pi_{st} = \pi_{ts}^{-1} \tag{5.12}$$

  for all sorts s and t in $\mathcal{S}$;

- it is *composition-consistent*; *i.e.*,

  $$\pi_{tu} \circ \pi_{st} = \pi_{su} \tag{5.13}$$

  for all sorts s, t, u in $\mathcal{S}$ — that is, $\pi_{tu}(\pi_{st}(f)) = \pi_{su}(f)$, for all feature symbols $f \in \mathcal{F}$;

[3] Or, equivalently: $\pi_{st}(f) = \pi_{st}(f') \to f = f'$.

- it is *order-consistent*; *i.e.*, if any sorts s, s', t, and t' in $\mathcal{S}$ are such that $s \preceq s'$ and $t \preceq t'$, then this necessarily implies that,

$$\pi^{\neq}_{s't'} \subseteq \pi^{\neq}_{st} \tag{5.14}$$

where $\pi^{\neq}_{st} \stackrel{\text{def}}{=} \{\langle f, \pi_{st}(f)\rangle \,|\, \pi_{st}(f) \neq f\}$. This ensures that the same feature is always mapped to the same feature along a sort-order chain.

Authors' comment: *The algebraically-minded reader will probably flinch. In particular, isn't this π mapping the "missing link" [!] needed in order to complete a formal connection between our $\mathcal{OSF}$ formalism and Category Theory? We, the authors, agree that this does look like a functor of categories — or more generally any structure-preserving commutative diagram represented as order-sorted feature graph structures. Namely, sorted nodes can be seen as objects and their features can then be seen semantically as categorical arrows between objects when we close features by composition. This is because they denote functions that are composable — the feature of a feature is also a function (denoting the semantic composition of the features's denotations) — and all feature compositions out of a node that converge to a common node (by sharing a tag at the end) must commute (i.e., satisfy a feature-path equation). Then, the feature correspondence mappings π from one sort to another (subject to the coherence constraints above) can indeed be seen as a fuzzy categorical endofunctor for the (strict monoidal) category $\langle \mathcal{S}, \mathbf{2}^{\mathcal{F}} \mapsto \mathbf{2}^{\mathcal{F}}\rangle$.*

It might be worthwhile trying to make this connection formally more explicit in the style of [133] and [82], for example. However, we shall abstain here for fear of losing some readership despite our best efforts. For our present purposes, we will specify next this feature mapping more operationally by substituting the features yielding a sort t out of a tag X of sort s that occurs in an $\mathcal{OSF}$ constraint set using the $\mathbf{fmap}^{\pi_{st}}_{X}$ *construct we define below as Expression (5.15).*

§ $\mathcal{OSF}$ Unification Modulo Feature Alignment

In Rule Sort Intersection Modulo Feature Alignment of Figure 5.5, the function **fmap** is parameterized by (1) a variable X, and (2) a one-to-one feature mapping π_{st} associating to each feature f in $\mathcal{F}$ a unique feature $\pi_{st}(f)$ in $\mathcal{F}$. When sort s is compared with sort t, any feature

Sort Intersection Modulo Feature Alignment

$$\frac{\phi \;\&\; X:s \;\&\; X:t}{\mathbf{fmap}^{\pi_{st}}_{X}(\phi) \;\&\; X : s \curlywedge t}$$

Figure 5.5: Sort intersection rule for $\mathcal{OSF}$ unification modulo feature alignment

f for tag X of sort s is made to correspond with feature $\pi_{st}(f)$ of sort t. So parameterized, $\mathbf{fmap}^{\pi_{st}}_{X}$ transforms a conjunctive $\mathcal{OSF}$ constraint ϕ into another conjunctive $\mathcal{OSF}$ constraint

$\mathbf{fmap}_X^{\pi_{st}}(\phi)$ obtained from ϕ by replacing each constraint of the form $X.f \doteq Y$ by the constraint $X.\pi_{st}(f) \doteq Y$. Formally,

$$\begin{cases} \mathbf{fmap}_X^{\pi_{st}}(\phi) \stackrel{\text{def}}{=} \phi & \left[\begin{array}{l}\text{if } X.f \doteq Y \text{ is not part of } \phi \\ \text{for any } f \in \mathcal{F} \text{ and any } Y \in \mathcal{V}\end{array}\right]; \\ \mathbf{fmap}_X^{\pi_{st}}(\phi \ \& \ X.f \doteq Y) \stackrel{\text{def}}{=} \mathbf{fmap}_X^{\pi_{st}}(\phi) \ \& \ X.\pi_{st}(f) \doteq Y & [\text{otherwise}]. \end{cases} \tag{5.15}$$

A feature map $\mathbf{fmap}_X^{\pi_{st}}$ that applies to an $\mathcal{OSF}$ constraint transforming it into another $\mathcal{OSF}$ constraint is naturally extended to apply as well to a ψ-term to transform it into another ψ-term as follows:

$$\mathbf{fmap}_X^{\pi_{st}}(\psi) \stackrel{\text{def}}{=} \varphi^{-1}(\mathbf{fmap}_X^{\pi_{st}}(\varphi(\psi))); \tag{5.16}$$

that is, it is defined as the ψ-term whose dissolved form is the result of applying that same feature map to the dissolved form of the original ψ-term.[4]

When π_{st} is the identity for a pair of sorts s and t, Rule **Sort Intersection Modulo Feature Alignment** of Figure 5.5 becomes the conventional rule **Sort Intersection** of Figure 4.6 of $\mathcal{OSF}$ term unification. In practice, it is more likely to be the case for most similar pairs of sorts.

§ $\mathcal{OSF}$ Generalization Modulo Feature Alignment

The same observation can be made for $\mathcal{OSF}$ generalization modulo feature mapping as shown by Rule **Unequal Tags Modulo Feature Mapping** of Figure 5.6. Note that, unlike Rule **Unequal Tags** of Figure 4.16 which identifies features in the same order having the same index, this rule identifies which feature to use for the left ψ-term's subterm depending on the index of the corresponding right ψ-term's feature given by π_{st}. This feature is identified as f_i, and so $\mathbf{index}(f_i) = i$ for $i = 1, \ldots, m$. But by (5.12), inverse-consistency of the feature map π_{st}, this means that $f_i = \pi_{ts}(g_k)$, for some $k \in \{1, \ldots, p\}$. From this, it comes that, for any index $k \in \{1, \ldots, p\}$, there is a unique index $i \in \{1, \ldots, m\}$ such that $i = \mathbf{index}(\pi_{ts}(g_k))$.

Note that this rule could equivalently be replaced by its symmetric form; namely, Rule **Symmetric Fuzzy Unequal Tags** of Figure 5.7.

[4] See Equation 4.5 for the definition of ψ-term dissolution into an $\mathcal{OSF}$ constraint.

UNEQUAL TAGS MODULO FEATURE ALIGNMENT

$$\left[\begin{array}{l} X \neq Y; \\ m,n,p \geq 0 \text{ and } \{h_1,\ldots,h_p\} \stackrel{\text{def}}{=} \{\pi_{st}(f_1),\ldots,\pi_{st}(f_m)\} \cap \{g_1,\ldots,g_n\} \\ \text{s.t. } h_k \stackrel{\text{def}}{=} \pi_{st}(f_k) = g_k \text{ for all } k = 1,\ldots,p\,; \\ \psi'_k \stackrel{\text{def}}{=} \psi_{\mathbf{index}(\pi_{ts}(g_k))} \text{ for } k = 1,\ldots,p; \\ \gamma_1^0 \stackrel{\text{def}}{=} \gamma_1 \circ \{X/Z\} \text{ and } \gamma_2^0 \stackrel{\text{def}}{=} \gamma_2 \circ \{Y/Z\}\,, \text{ where } Z \text{ is a new tag name} \end{array}\right]$$

$$\frac{\begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \vdash \begin{pmatrix}\psi'_1\\ \xi_1\end{pmatrix} \!\uparrow\! \begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \chi_1 \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \quad \cdots \quad \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \vdash \begin{pmatrix}\psi'_p\\ \xi_p\end{pmatrix} \!\uparrow\! \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \chi_p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}}{\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash \begin{pmatrix}X : s\,(f_i \;\to\; \psi_i)_{i=1}^m\\ Y : t\,(g_j \;\to\; \xi_j)_{j=1}^n\end{pmatrix} U : s\vee t\,(h_k \;\to\; \chi_k)_{k=1}^p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}}$$

Figure 5.6: $\mathcal{OSF}$ generalization modulo feature alignment

SYMMETRIC UNEQUAL TAGS MODULO FEATURE ALIGNMENT

$$\left[\begin{array}{l} X \neq Y; \\ m,n,p \geq 0 \text{ and } \{h_1,\ldots,h_p\} \stackrel{\text{def}}{=} \{f_1,\ldots,f_m\} \cap \{\pi_{ts}(g_1),\ldots,\pi_{ts}(g_n)\} \\ \text{s.t. } h_k \stackrel{\text{def}}{=} f_k = \pi_{ts}(g_k) \text{ for all } k = 1,\ldots,p\,; \\ \xi'_k \stackrel{\text{def}}{=} \xi_{\mathbf{index}(\pi_{st}(f_k))} \text{ for } k = 1,\ldots,p; \\ \gamma_1^0 \stackrel{\text{def}}{=} \gamma_1 \circ \{X/Z\} \text{ and } \gamma_2^0 \stackrel{\text{def}}{=} \gamma_2 \circ \{Y/Z\}\,, \text{ where } Z \text{ is a new tag name} \end{array}\right]$$

$$\frac{\begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \vdash \begin{pmatrix}\psi_1\\ \xi'_1\end{pmatrix} \!\uparrow\! \begin{pmatrix}\gamma_1^0\\ \gamma_2^0\end{pmatrix} \chi_1 \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \quad \cdots \quad \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \vdash \begin{pmatrix}\psi_p\\ \xi'_p\end{pmatrix} \!\uparrow\! \begin{pmatrix}\gamma_1^{p-1}\\ \gamma_2^{p-1}\end{pmatrix} \chi_p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}}{\begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \vdash \begin{pmatrix}X : s\,(f_i \;\to\; \psi_i)_{i=1}^m\\ Y : t\,(g_j \;\to\; \xi_j)_{j=1}^n\end{pmatrix} Z : s\vee t\,(h_k \;\to\; \chi_k)_{k=1}^p \begin{pmatrix}\gamma_1^{p}\\ \gamma_2^{p}\end{pmatrix}}$$

Figure 5.7: Equivalent symmetric $\mathcal{OSF}$ generalization modulo feature alignment

Chapter 6

Version of April 10, 2020

Discussion

This chapter is a set of sections summarizing topics that we learned about in the course of doing our research trying to situate it with respect to others, as well as make it pragmatically usable for others. Section 6.1 reviews related work. Section 6.2 indicates further avenues to explore with this work. Section 6.3 reviews a few systems implementing some notion of fuzzy unification. Section 6.4 discusses the design of public libraries for fuzzy lattice operations on $\mathcal{FOT}$s and $\mathcal{OSF}$ terms. Section 6.5 looks at some potential applications and uses of fuzzy lattice operations on graph structures, especially how it may help enable approximate language understanding. Section 6.6 discusses a use case for fuzzy $\mathcal{OSF}$ lattice operations in Approximate Information Retrieval combining fuzzy and Bayesian reasoning.

6.1 Related work

This is non-exhaustive review and discussion of work dealing with other work in related topics. Section 6.1.1 reviews other work on fuzzy unification; Section 6.1.2 reviews graph-similarity measures; Section 6.1.3 reviews some known fuzzy data models (such as fuzzy object-oriented) and subtyping; Section 6.1.4 discusses other models of fuzzy knowledge representation; and, Section 6.1.5 looks at how our work could benefit soft-constraint solving;

6.1.1 Other fuzzy unification work

In the course of this work, we searched the literature for "*fuzzy unification*" and "*fuzzy logic programming*," and variations thereof. Our first observation in pursuing this interest has been that, unlike existing Prolog languages (and other technology that relies on standard $\mathcal{FOT}$ unification), not all the fuzzy $\mathcal{LP}$ languages that have been proposed and/or implemented (even if only as prototypes) agree on the same fuzzy $\mathcal{FOT}$ unification operation. We have looked at some of the most prominent among those existing in an attempt to characterize their fuzzy unification operations. We next summarize some essential points that we understood of these variations on fuzzy unification from our perspective. We will proceed succintly, as it would be presumptuous of us to give an exhaustive recap of research in Fuzzy $\mathcal{LP}$. Again, our perspective is not so much

Fuzzy $\mathcal{LP}$ as it is that of understanding its fuzzification of the lattice-theoretic operations on $\mathcal{FOT}$ s such as $\mathcal{FOT}$ unification, which is an essential part on any $\mathcal{LP}$ system.

Before we focused on M. Sessa's fuzzy $\mathcal{FOT}$ unification algorithm, which we present in Chapter 3,[1] we looked at other work which we review here. We also discuss our reasons for our choice to follow Sessa's approach as opposed to fuzzy Datalog or edit-distance $\mathcal{FOT}$ matching we describe next.

§ Fuzzy Datalog Unification

Among earlier frequently cited works related to fuzzy $\mathcal{FOT}$ unification is [37], where the title leads one to expect a fuzzy term unification in a fuzzy $\mathcal{LP}$ language. It is not quite that, however, as it restricts solving similarity equations over word symbols tolerating some imprecise matches (*i.e.*, a kind of fuzzy Datalog).[2] Such mismatches could come from (possibly accidental) syntactic proximity (*e.g.*, typos, misspellings).[3] The authors propose to accumulate mismatched symbols into what they dub "*clouds*," which represent in effect similarity classes of constant symbols (nullary functors). These clouds are given a measure of "similarity" computed as the meet of those of the components — which they propose to conceive as a "cost" of how much it deviates from a perfect match (which itself has zero cost, since perfect).

Note that when the only non-variable terms defined are constants, the rules of Figure 3.3 accumulate all constant mismatches as unresolved equations. Thus, what constitutes Arcelli *et al.*'s "clouds" are the sets of constants making up the equivalence classes of the reflexive-transitive closure of the relation containing these equations.

While this could be useful as a particular fuzzy extension of Datalog, it does not address issues concerning fuzzy database representation and evaluation issues such as expounded in [58] for (crisp) Datalog. In fact, such fuzzy extensions of Datalog had already been proposed, with a straightforward fix-point semantics extending that of classical Datalog (*e.g.*, [1]). Since it limits itself to fuzzification of a Datalog-like $\mathcal{LP}$, the semantics of Arcelli *et al.*'s fuzzy $\mathcal{LP}$ language only considers approximate equations between constant symbols, whose mutual fuzzy proximity is specified as fuzzy "proximity matrices". This early form of fuzzy unification was put to use in the fuzzy $\mathcal{LP}$ language Likelog [35], [36], [38], [39].

§ Edit-Distance Fuzzy Unification

Arcelli *et al.*'s fuzzy constant unification was later elaborated to work on full $\mathcal{FOT}$s as first exposed in [37] and used in [81], and then again in [154] and [155]. The authors use the same kind of fuzzy unification on constants (*i.e.*, names formalized as symbol strings), but instead of names deemed similar (*i.e.*, in a same "cloud" or similarity class), the more classical notion of *edit distance* is used to evaluate the similarity degree of a fuzzy name match (normalized over the symbol lengths). Edit distance between two strings is the minimal number of elementary edit actions (deleting a character, inserting a character, or replacing a character for another), in either

[1] Section 3.7.1.

[2] It is only acknowledged in the very last sentence of the paper, that the authors were yet "*aiming at extending this algorithm to the full-fledged algebra of first-order terms*" as future work.

[3] Or presumably, although they only mention it as a potential further work in their conclusion, from semantic proximity (such as could be specified as fuzzy knowledge in the form of fuzzy similarity matrices).

or both strings necessary to obtain a perfect match [116].[4] This fuzzy unification was put to use in biological genetics analysis with the fuzzy $\mathcal{LP}$ language FURY [81], [155].

In what follows, we use our own formal notation to summarize the essence of FURY-style fuzzy $\mathcal{FOT}$ unification [81], [154], [155]. We represent the empty string as ε, and a non-empty string as a dot-separated sequence of characters ending with the empty string (*e.g.*, `"this"` is represented as `t.h.i.s.`ε). Given a non-empty string $h.t$, we shall call its first character h its "*head*" and the substring t following it, its "*tail*." The length of a string s is its number of characters denoted $|s|$. That is, the monoid homomorphism:

$$|\varepsilon| \stackrel{\text{def}}{=} 0$$

$$|h.t| \stackrel{\text{def}}{=} 1 + |t|.$$

Thus, the edit distance $\delta(s_1, s_2)$ between two strings s_1 and s_2 is derived as:[5]

$$\left.\begin{array}{rcl} \delta(\varepsilon, s) & \stackrel{\text{def}}{=} & |s| \\ \delta(s, \varepsilon) & \stackrel{\text{def}}{=} & |s| \\ \delta(h.t_1, h.t_2) & \stackrel{\text{def}}{=} & \delta(t_1, t_2) \\ \delta(h_1.t_1, h_2.t_2) & \stackrel{\text{def}}{=} & 1 + \min\{\, \delta(t_1, h_2.t_2), \delta(h_1.t_1, t_2), \delta(t_1, t_2) \,\}, \quad \text{if } h_1 \neq h_2. \end{array}\right\} \tag{6.1}$$

These four defining equations express that the edit distance: (1) from any string to the empty string is the length of this string; (2) between two strings with equal first character, it is the distance between the remaining substrings; (3) otherwise, it is one plus the minimum of the three edit distances between one of the strings and the other string's rest, and between the two strings' rests. The latter is known as the "*Levenshtein distance*" between two strings.[6]

Because edit distance will increase with the lengths of strings, it is convenient to calibrate it over the size of the strings involved; hence the notion of "*normalized edit distance*" δ_N as in:

$$\delta_N(s_1, s_2) \stackrel{\text{def}}{=} \frac{\delta(s_1, s_2)}{\max(|s_1|, |s_2|)}. \tag{6.2}$$

In [81], this notion of (normalized) edit distance between constant symbol strings (including the empty string ε) is extended to an edit distance between $\mathcal{FOT}$ trees. It maps two terms t_1 and t_2 to a non-negative number $\delta(t_1, t_2) \stackrel{\text{def}}{=} m^{\sigma}_{n} \in \mathbb{N}$, which denotes the minimal total number m of mismatches (edit actions necessary to go from one to the other), along with two collateral pieces of information:

[4] It is used most crucially in Internet search keyword matches and DNA sequence alignment and matching.

[5] From which one can easily check that expected properties of a distance are satisfied by δ such as $\delta(s, s) = 0$, $\delta(s_1, s_2) = \delta(s_2, s_1)$, and $\delta(s_1, s_2) \leq \delta(s_1, s) + \delta(s, s_2)$, for any strings s, s_1, and s_2.

[6] The Levenshtein distance between two strings has the advantage to apply to strings of differring as well as of equal lengths. This is unlike the *Hamming distance* which is restricted to strings of equal lengths, and defined as the number of disagreeing character positions. This entails, in particular, that the Levenshtein distance between two equal-length strings is always less than or equal to their Hamming distance.

- σ — a most general variable substitution that may be necessary to resolve matches upon encountering variables in the process of computing this minimal edit distance between the two terms when assimilated to strings which are sequences of the non-punctuation characters that compose their their syntax; and,

- n — a normalization factor computed as the sum of the lengths of the longest of each of pair of symbols from each term as they are matched.

The normalization factor n is a function of the lengths of the terms when a term is seen as the string concatenation in the order they appear of the non-variable and non-punctuation symbols in it. In other words, parentheses, commas, variables, and ε are considered of length 0. Namely,

$$|t| \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } t \text{ is a variable} \\ |f| + \Sigma_{i=1}^{n}|t_i| & \text{if } t = f(t_1, \ldots, t_n),\ n \geq 0. \end{cases}$$

Compounding two $\mathcal{FOT}$ edit distances m_n^σ and p_q^θ consists in adding them up while composing their substitutions and adding their normalization factors:

$$m_n^\sigma + p_q^\theta \stackrel{\text{def}}{=} (m+p)_{n+q}^{\theta\sigma}. \tag{6.3}$$

In other words, as it sums the numbers of symbol mismatches, it also composes their associated variable substitutions and sums their normalization factors (which depend on the sizes of all the involved symbols). Note that this operation, while commutative in its numerical arguments (which are added), is not commutative in its substitution arguments (which are composed). It could also be defined by composing the substitutions in the other order if wished; but this is simpler.[7]

Given two $\mathcal{FOT}$s s and t, the edit distance between them $\delta(s,t)$ is defined as follows. If the first argument is the empty string, then:

$$\delta(\varepsilon, t) \stackrel{\text{def}}{=} |t|_{|t|}^{\emptyset} \tag{6.4}$$

meaning that the edit distance is the length of the second argument, which is also this distance's maximum known normalization factor, and the resulting substitution is the empty substitution (*i.e.*, the identity). If the first argument is a variable, then:[8]

$$\delta(X, t) \stackrel{\text{def}}{=} 0_0^{\{t/X\}} \tag{6.5}$$

meaning that it is zero, while binding its first argument to its second argument, with a zero normalization factor. If the second argument is a variable, then:

$$\delta(t, X) \stackrel{\text{def}}{=} \delta(X, t) \tag{6.6}$$

[7] The authors of [81] compose their substitutions the other way, which is why they need to write their recursive 'et' rule (the last one) with the first subterms as second arguments when collecting those of the subterms.

[8] In our (meta-)notation, in order to stress the syntactic nature of arguments that must be variables, we use Prolog's convention of identifying variables with symbols starting with a capital letter.

by symmetry, which then uses the previous case. Otherwise (neither argument is a variable), let $s = g(s_1, \ldots, s_m)$ and $t = f(t_1, \ldots, t_n)$ for some $m, n \geq 0$; then:

$$\left.\begin{array}{rcl} \delta(s,t) & \stackrel{\text{def}}{=} & \delta(f,g)^{\emptyset}_{\max(|f|,|g|)} \\ & & + \ \min\ \{\ \delta(\varepsilon, s_1) \ + \ \delta(\varepsilon(s_2, \ldots, s_m),\ \varepsilon(t_1, \ldots, t_n)) \\ & & \quad ,\ \delta(\varepsilon, t_1) \ + \ \delta(\varepsilon(s_1, \ldots, s_m),\ \varepsilon(t_2, \ldots, t_n)) \\ & & \quad ,\ \delta(s_1, t_1) + \ \delta(\varepsilon(s_2, \ldots, s_m)\sigma, \varepsilon(t_2, \ldots, t_n)\sigma)\ \} \\ & & \text{if } \delta(s_1, t_1) = p^{\sigma}_{q} \text{ for some } p, q \geq 0 \text{ and substitution } \sigma \end{array}\right\} \quad (6.7)$$

which defines a Levenshtein distance extended from strings to terms. It is equal to the edit distance between the functors plus the minimum of the three possible ways of aligning the respective sequences of subterms, composing substitutions and adding normalization factors, each set to the maximum functor length, while incrementally instantiating subterms remaining in the tails with the accumulated substitutions resulting from computing the heads' edit distance at each recursive calls.

The above rules given as Equations (6.4)–(6.7), with our own — and simpler — notation, are adapted from Definition 5 of the Gilbert-Schroeder paper [81]. However, while they agree on the first three rules, they do not on the last one. On that last one, they only agree on the first two cases of the three recursive patterns, they differ on the last: our own rule — Equation (6.7) — propagates to the rest of the arguments the substitution resulting from computing the edit distances between the first arguments of both terms. The two other cases need not do so as either term's first argument is only paired with the empty string ε, which simply returns the identity substitution $\emptyset$ — by Equation (6.4). Because this propagation is **not** done in their definition of the term edit distance '*et*' (Definition 5), this makes it incorrect. Take for instance the two terms $f(a, b)$ and $g(X, X)$. According to that definition, their term edit distance is $1^{\{a/X\}}_3$. However, taking that substitution into account, it should be $2^{\{a/X\}}_3$ (since there are 2 mismatches between $f(a, b)$ and either $g(a, a)$ or $g(b, b)$: $f \neq g$ and $a \neq b$). Indeed, the definition given in [81] means that the two occurrences of X are seen as two independent variables which then get independently bound (one time to a and the other time to b), then composing the substitutions will keep only the first one ($\{a/X\}$) and not account for the argument mismatch $a \neq b$. Whereas, propagating the substitution as done in Equation (6.7) makes it possible to account for the mismatch (since a will be have been substituted for X), therefore correctly returning $2^{\{a/X\}}_3$.

It could have been a typo or misprint in Definition 5 in [81], perhaps. But in other later papers using this unification (such as [154] and [155]), the same definition is again given. At any rate, to propagate substitutions from one matching argument to matching the rest is a simple option. Not doing it, although not optimal, does not invalidate their approach when the substitution propagation is done correctly (as done in Equation (6.7)); it just catches less mismatches in general than ought to be reported (since it has for effect to ignore potential mismatches that may come from any multiple-occurrence variable which are already bound to different symbols).

In this manner, this accounts for the fact it may be necessary to perform variable substitutions while establishing the normalized edit distance between two terms t_1 and t_2 such that the following

fuzzy equation holds whenever $\delta(t_1, t_2) = m_n^\sigma$:

$$t_1\sigma \sim t_2\sigma \left[\tfrac{n-m}{n}\right]. \tag{6.8}$$

Indeed, having m character mismatches over a maximum total length of n characters means that the rate of mismatch is $\frac{m}{n} \in [0.0, 1.0]$; or, equivalently, that the rate of correctly matched characters is $1 - \frac{m}{n}$; *i.e.*, $\frac{n-m}{n}$. It can then be used as a similarity degree fuzzifying the equation $t_1\sigma = t_2\sigma$. Indeed, if all the characters are mismatched, then $m = n$ and therefore the similarity degree of this equation is zero; whereas, if no characters are mismatched, then $m = 0$, and the truth value is one. As expected, the higher the number of mismatches, the fuzzier the solution.

This is a very interesting trick: "*stringifying*" the syntax of a $\mathcal{FOT}$ and then using fuzzy symbol matching while counting how many mismatches over how long symbols and substituting terms for variables as needed to resolve discrepancies in term structure. Thus, calculating the normalized edit distance between two terms with Equations (6.4)–(6.7) operates an implicit unification procedure (which we shall call Gilbert-Schroeder fuzzy unification). It has, in fact, the same recursive pattern as Robinson's procedural unification algorithm, relaxed to tolerate functor and arity inequalities [142].[9]

There are some important observations to be made at this point regarding Gilbert-Schroeder fuzzy $\mathcal{FOT}$ unification.

- It applies to conventional (crisp) Prolog terms: there is no need for "fuzzy $\mathcal{FOT}$s" whatever such may be (it is the unification that is fuzzy, not the terms).
- It is a purely lexical process: it relates strings as character sequences regardless of word meaning and/or context.
- It can always derive a minimal edit distance between two terms, however unrelated they may be — the more lexically unrelated, the larger this distance will be, although it will always be finite as it is bounded by a function of the size of the terms,[10] as well as the lengths of the functor symbols in them and the number of variable re-occurrences at leaves. Normalizing with respect to the length of concatenation of the longest of each pairs of symbols appearing in corresponding subterms gives a bounded measure in $[0.0, 1.0]$ of the character mismatch rate, therefrom a fuzzy matching measure may be derived.
- When fuzzy-unifying two non-variable non-ε terms, their arities (number of subterms) may differ as each subterm of one is unified with each subterm of the other, keeping only the minimal total number of mismatches (and collateral substitution and normalization factor) — which raises efficiency concerns. Such concerns have been addressed for tree edit distances in more recent works such as, *e.g.*, [71], although not for $\mathcal{FOT}$s which are rooted directed acyclic graphs (variables are shared nodes). Although the number of arguments of two fuzzy matching terms may differ, it must be noted that in computing edit-distance between two $\mathcal{FOT}$s, the order of argument-position is always preserved.

[9] *Op. cit.*, Section 5.8, Page 32.
[10] The number of functor nodes.

- It is not difficult to understand from Equations (6.1) and (6.7), that the complexity of a *naïve* implementation of this recursive scheme becomes quickly prohibitive for pragmatics. Thus, optimization methods, implementation techniques such as Dynamic Programming including specific-domain heuristics, have been the center of attention [44], [171], [168].[11]

- More worrisome is that computing this term edit distance is not only expensive, it is also *non-deterministic*. Indeed, there may be equal minimal number of mismatches with different and incomparable variable substitutions. For example, the minimal term edit distance between `f(X,X,a)` and `g(b,Y,Y)` is 2 with either substitutions $\{\mathtt{b}/\mathtt{X}, \mathtt{b}/\mathtt{Y}\}$ or $\{\mathtt{a}/\mathtt{X}, \mathtt{a}/\mathtt{Y}\}$, which are both most general although mutually incomparable (*i.e.*, they are not alphabetical variants "up to variable renaming" of one another).

6.1.2 Feature-graph similarity measures

Work using our $\mathcal{OSF}$ formalism has also elaborated a general lattice-theoretic approach to measuring similarity over $\mathcal{OSF}$ graphs [126]. We now review this work and discuss how our approach and theirs are in fact quite compatible, the latter providing a way to derive from the structure of $\mathcal{OSF}$ graphs a similarity distance which can be used as the fuzzy information presumed available by the former.

6.1.3 Fuzzy data models

Fuzzy object-oriented data model [52].

Fuzzy subtyping: [54].

6.1.4 Fuzzy ontologies

They were earlier attempts at fuzzifying Description Logic ($\mathcal{DL}$) (*e.g.*, [175, 159, 161]. This was done by attaching a similarity degree to $\mathcal{DL}$ assertions and interpreting constraints with fuzzy connectives: infimum ($\wedge$) is $\min$, supremum ($\vee$) is $\max$, and complement ($\phi \rightarrow \overline{\phi}$) is $w \rightarrow (1-w)$. Specifying minimal and/or maximal values is used to disregard all assertions and constraints with similarity degree outside a specified interval. In this regard (setting minimal/maximal bounds), the latter is similar to such fuzzy logics as [32].

There have been many others since then. Here is very short chronological list of the many, many, variations on more recent work on how to fuzzify $\mathcal{DL}$ ontologies.

- [2011] Fuzzy ontology representation using $\mathcal{OWL}$ 2 [49]
- [2014] LiFR: A Lightweight Fuzzy $\mathcal{DL}$ Reasoner [164]
- [2015] The fuzzy ontology reasoner *fuzzyDL* [50]
- [2017] Fuzzy ontology representation using $\mathcal{OWL}$ 2 [51]
- [2018] Dealing with uncertainty: Fuzzy (Description) Logics and Fuzzy Ontologies [143]

[11] See Section 6.2.3.

As one can see in this list, the quasi-totality of existing work in fuzzy knowldedge representation use $\mathcal{DL}$ as the medium for knowldedge representation and reasoning.[12] One of the main contributors is Umberto Fraccia, starting with his PhD work in the late 90s ([159], [160]). His more recent work is listed above along with other work using $\mathcal{DL}$ as the knowldedge representation formalism to fuzzify.

In summary, all the above work implicitly assumes that knowledge is represented as $\mathcal{DL}$, and specifically in one of its official $\mathcal{OWL}$ dialects. This makes fuzzy reasoning rely on fuzzifying $\mathcal{DL}$-kinds of rules. However, as explained in [9], reasoning in $\mathcal{DL}$ and $\mathcal{OSF}$ Logic is quite different. Therefore, how our approach compares formally with the above work in terms of expressiveness and implementation performance begs further study.

6.1.5 Soft constraints

There has been considerable work dealing with fuzzy Constraint Solving Problems (Fuzzy $\mathcal{CSP}$). One of the most thorough and comprehensive is that of Bistarelli *et al.* [48]. It solves Fuzzy $\mathcal{CSP}$s by combining Abstract Interpretation of constraints [67] with $\mathcal{CSP}$ [144].[13] Their approach to soft $\mathcal{CSP}$ extends to temporal constraints as well.

The kind of constraints we deal with in this book concerns similarity (a semantic-distance measure between words) among symbols in two major operations used in theorem proving (whether resolution-based, equational, temporal, or combinations). But $\mathcal{CSP}$ has been used with great success in boosting the performance of general-purpose theorem-proving by relieving the burden of search on specific patterns. This is possible because a constraint's semantics is that of a relation satisfied by its arguments and therefore inherits all the logical semantics for free while, operationally, a special-purpose algorithm can efficiently compute its solutions.

Therefore, it is conceivable to use our work in the same context as it provides a means of approximating term-based reasoning on a quotient set of terms rather than a set of terms. This begs for further study.

6.2 Further Work

This speculates on a few (and certainly non-exhaustive) possible avenues for further work. Section 6.2.1 indicates how our approach may be extended with the tools developed for the optimization of constraint-solving implementation by Abstract Interpretation in general; Section 6.2.2 makes a link with fuzzy automata for fuzzy string matching; Section 6.2.3 speculates about fuzzy dynamic programming; Section 6.2.4 tries to situate our work in the contextt of Fuzzy Quantum Logic; Section 6.2.5 looks a fuzzy Formal Concept Analysis; Section 6.2.6 considers fuzzifying the Generalized Distributive Law.

[12] See [9], [33], and [11] on how $\mathcal{OSF}$ Logic and $\mathcal{DL}$ are formaly and operationally related. See also this.

[13] A popular introduction to $\mathcal{CSP}$ is Chap. 6 [127] of [128] (was Chap. 5 in previous editions).

6.2.1 Fuzzy abstract interpretation

As shown in Section 6.1.5, Abstract Interpretation offers as mathematically elegant and operationally efficient formal analysis of computation, whatever the computational model may be [65]. When the latter is constraint solving, it applies as well ([66], [67], [64]).

In Chapter 3 and Chapter 4, we covered the lattice-theoretic properties of fuzzy operations on $\mathcal{FOT}$s $\mathcal{OSF}$ graphs. Such fuzzy operations may be put to use in Automated Reasoning and Knowledge Representation to perform approximate deduction and induction over composite objects and concepts in the same manner as Abstract Interpretation. In the specific case where information is represented as consistent $\mathcal{FOT}$s or $\mathcal{OSF}$ graphs and abstraction is provided by fuzzy lattice-theoretic approximation thereon — whether for consistent approximate deduction (by fuzzy unification), or for consistent approximate induction (by fuzzy generalization).

6.2.2 Fuzzy automata

An interesting avenue that neeed to be explored is the consequence of our results regarding a very close notion of labeled graphs; *viz.*, Finite-State Automata ($\mathcal{FSA}$) — whether deterministic ($\mathcal{DFA}$) non-deterministic ($\mathcal{NFA}$). Indeed, as initially explained in [4] and [6], $\mathcal{OSF}$ terms were formalized as labeling of what is essentially a $\mathcal{FSA}$: the alphabet is the set of features, the states are the sort nodes, the initial state is the root node.

A state is an equivalence class among words that are feature paths. There in only one reject state labeling the $\bot$ sort. The notion of accepting state is generalized to a sort-labeled state where subsorting in interpreted as compatibility (accept all feature paths ending in this state with most general compatible sort); a feature transition leading to no existing compatible sort, ends in the reject state ($\bot$).

Unification of two $\mathcal{FSA}$s computes the most general $\mathcal{FSA}$ (smallest in number of states) that is consistent with the two given $\mathcal{FSA}$s — the intersection of the regular feature-path languages of both $\mathcal{FSA}$s. Generalization of two $\mathcal{FSA}$s computes the most general $\mathcal{FSA}$ (smallest in number of states) that is consistent with either of the given $\mathcal{FSA}$s — the union of the two regular feature-path languages of each $\mathcal{FSA}$. A formal clarification of this connection can thus lead to a formal understanding of how our fuzzy $\mathcal{OSF}$ lattices can be also interpreted as fuzzy $\mathcal{FSA}$ lattices — and conversely.

It is easy to see that, by abstracting the algebraic operations it uses, Dijsktra's shortest path algorithm[14] is only one instance of a larger family of algorithms in an inf/sup lattice known as Warshall's algorithm working on any such algebraic structures [172].[15] This is of great benefit for software development: it is sufficient to encode only one algorithm with abstract `inf` and `sup` operations and vary the effect according to any specific instantiations of these operations. Many closing algorithms in dual algebras such as (semi-) rings, (semi-) lattices, *etc.*, are of this type (see, *e.g.*, Algorithm 1).

Formalization of fuzzy NFAs:

- [55]

[14] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
[15] http://www.cse.chalmers.se/~coquand/AUTOMATA/over7.pdf

- [117]

General references and links on fuzzy regular expressions (to connect with $\mathcal{OSF}$ unification extended to regular path expressions [42]):

- Fuzzy Regular Expressions
- Fuzzy Regular Expression Matching software

Approximate string matching: [105].
More readings on fuzzy automata:

- fuzzy automata [40]
- fuzzy and non-deterministic automata [124]
- Myhill-Nerode theory for fuzzy languages and automata [98]
- Lattice-ordered monoids and automata [162, 163]

6.2.3 Fuzzy dynamic programming

Bellman's Dynamic Programming [44] is a problem-solving paradigm first defined formally by Bellman for recursive optimization. Informally, it can be described as a technique that remembers previously established goals that can then be reused without being having to be re-established. It has been extended to several fuzzy dynamic programming type models in which various elements have been fuzzified, notably the goals and constraints, state and control, state transitions, termination time, *etc.*, ... [106].

Dynamic Programming techniques have been shown to be of great benefit in the implementation of Logic Programming along the lines of [104] and $\mathcal{XSB}$ Prolog. Hence, considering Fuzzy $\mathcal{LP}$ with similar techniques while using fuzzy unification as we define it is a potential avenue to explore.

6.2.4 Fuzzy quantum logic

Also to read the intriguing interesting connections of Fuzzy Logic with Quantum Logic. To cite just a few: [47],[16] [138], [139], [68], [152]. [89]. See also: this and this. In [89], a nice algebraic summary is given as:

Logic Property	Classical Logic	Birkhoff – Von Neumann Logic	Zadeh Fuzzy Logic	Giles – Łucasiewicz Fuzzy Logic
Binary	yes	yes	no	no
Commutative	yes	no	yes	yes
Distributive	yes	no	yes	no
Excluded Middle	yes	yes	no	yes
Non-Contradiction	yes	yes	no	yes

[16] This is generally credited to be the pioneering paper.

Attributed conceptual information is the space of observations (an infinite-dimensional (quasi-Hilbert) space), and the phase space comprises functions of probable neighbor states of points in the observation space according to the axioms of Quantum Logic $\mathcal{QL}$, which is a weakening of $\mathcal{FL}$.

Reasoning with Vectors: A Continuous Model for Fast Robust Inference.

6.2.5 Fuzzy formal concept analysis ($\mathcal{FCA}$)

See essentially Radim Bělohlávek's work and references in there.

6.2.6 Fuzzy generalized distributive law ($\mathcal{GDL}$)

Fuzzifying the Generalized Distributive Law ($\mathcal{GDL}$) [27, 28] (*i.e.*, lifting it to any fuzzy lattice structure).

> ***Authors' comment:*** *Need to* ***read*** *and comment the state of the art, and infuse whether this can connect productiveley (in either directions) with $\mathcal{OSF}$ graphs seen as order-sorted $\mathcal{FSA}$s, whether deterministic ($\mathcal{DFA}$) or non-deterministic ($\mathcal{NFA}$), on regular languages of feature composition words (see, e.g., [42]).*
>
> *Review part of the material in [6] that defines lattices on an algebra of (ψ, ϵ, complemented) ($\mathcal{OSF}$) terms as automata on access-path languages [86, 87], and clarify the link with the lattices of fuzzy $\mathcal{OSF}$ terms elaborated in Section 4.2.*

6.3 Fuzzy Implementations

A few fuzzy $\mathcal{LP}$ systems have been proposed and implemented using some of the fuzzy unification operations defined by the state of the art that we overviewed in Section 3.7.1, or variants thereof. The following are just some among the many one can look up, some of which may be downloadable.

- `1995` — Fuzzy extension of Datalog [1];[17]
- `1997` — Likelog (a fuzzy datalog) [35];[18]
- `2002` — LP with context-dependent fuzzy unification [32];[19]
- `2002` — Fuzzy Prolog using CLP($\mathbb{R}$); not a fuzzy CLP($\mathbb{R}$) programming language, but using CLP($\mathbb{R}$) to implement fuzzy operations on union of intervals of real numbers [167];[20]
- `2010` — Bousi~Prolog a fuzzy Prolog using a weak version of fuzzy unification [100];[21]

[17] http://people.inf.elte.hu/kiss/14abea/Achs_1995_ActaCybernetica.pdf
[18] http://www.programmazionelogica.it/[...]/uploads/1997/06/319_Fontana1.pdf
[19] http://repositori.udl.cat/bitstream/handle/10459.1/57984/001858.pdf
[20] https://cliplab.org/papers/fuzzy-lpar02_bitmap.pdf
[21] http://www.sciencedirect.com/science/article/pii/S1571066109002874

- 2015: Fasill [102];[22]
- 2018: Bousi∼Prolog using Aït-Kaci/Pasi unification [63].[23]

6.4 Proposed Proofs of Concept

The most important consequence of this work, it is hoped, is that it can provide several pragmatic operational improvements on inference and learning methods from purely declarative structure-oriented constraint specifications. This is meant to ease efficient implementation and the provision of software tools. In what follows we discuss a minimal set of necessary software development efforts needed to enable the expressive potential of fuzzy lattice operations on $\mathcal{FOT}$s and $\mathcal{OSF}$ graphs.

§ Interfaces (Java packages and libraries) for $\mathcal{FOT}$s and $\mathcal{OSF}$ terms

We have started an implementation in Java of the operational semantics derived from the axioms and rules that we presented and proven correct in Chapter 3, which has allowed us to confirm our results on concrete examples [12].[24] This was eased by the fact that the fuzzy lattice operations do not require altering these conventional first-order structures.

6.5 Applications

Example of applications and uses of fuzzy lattice operations on terms and order-sorted attributed graphs in knowledge and data processing (deduction and learning), linguistics (fuzzy order-sorted fuzzy HPSGs).[25]

> ***Authors' comment:*** *Take a look later at applications:*
>
> - *Zadeh's fuzzy interpretation of linguistic hedges [178]*
> - *Common Fuzzy Distributions for linguistic hedges [59]*
> - *Other sorts of application (fuzzy control in particular) [180].*

[22] https://arxiv.org/pdf/1501.02034.pdf

[23] https://[...]Towards_a_Full_Fuzzy_Unification_in_the_Bousi_Prolog_system

[24] See also [63], a recent extension of the Bousi-Prolog system based on our similarity-based unification tolerating functors of differing arities.

[25] *Op. cit.*, Section 3, pp. 387 ff. See HPSG. This is linguistics research that was developed independently of, but contemporaneously with, research on the more general $\mathcal{OSF}$ formalism ([18], [7], [17]). But it can be seen as an instance of $\mathcal{OSF}$ calculus for a specific purpose (making sense of written or spoken language in diverse contexts), only using its own jargon and biased toward linguistic analysis. So everything we formalize on fuzzy $\mathcal{OSF}$ could easily carry over to fuzzy HPSG with appropriate rewording using their specific jargon — although fuzzily.

6.6 Use Case — User-Fit Product Recommendation

As a realistic practical example using fuzzy $\mathcal{OSF}$ lattice operations, we discuss an application that could be part of a "smart" Information Retrieval system managing the interaction with a user for suggesting sets of available products fitting an interested user through a typical Internet recommandation service (*e.g.*, for videos, books, equipment, technical articles, *etc.*). Our objective with such an example is to illustrate how adding fuzzy information processing capabilities based on the formalism we have presented enables "smoother" control over the adequacy of the retrieved information enabling better user satisfaction with the proposed data and more flexible exploration of novel topics and areas less prone to inherent biases that constitute the principal pitfall of so-called Naïve Bayes models;[26] see Section 6.6.2. The same model would work for arbitrary categories of sorts of objects and users. It assumes that an ontology model of sorts of all objects in the form of a sort taxonomy typing structured objects according to contents along specific preference dimensions has been defined with attributes and values types and/or ranges. This is the crisp base to be fuzzified.

Although the crisp base system this extends is a generic model, the latter is probably close to the main idea underlying actual existing recommendation systems, and so our fuzzy $\mathcal{OSF}$ operations could be envisaged by such or similar systems to improve user satisfaction by learning to suggest more adequate recommendations. It can work as well when interacting with an unregistered user, in which case the accumulation of interaction data can be associated to an id with known attributes such as the connected agent's Internet address, local server location (city, country), language used, *etc.*, . . .

6.6.1 Basic model

The interaction scenario that our system is meant to improve is to suggest a set of specific products to a user "*lambda*" (referred to as "*User* **You Sir**"), based on one's past history. The representation formalism of data and knowledge in our model will be $\mathcal{OSF}$ graphs represented with a straightforward $\mathcal{RDF}$ notation, so it can be implemented using $\mathcal{HOOT}$, a language for expressing and querying hierarchical ontologies, objects, and types [10].

It is based on the following assumptions.

- A partially ordered taxonomy of the product sorts and all other concepts used is given, defining the model's $\mathcal{OSF}$ signature $\mathcal{S}$. A (sub)sort denotes a (sub)category of products, or sets of other data objects, and the partial order on sorts is set inclusion ($\subseteq$) for the sets they denote. This taxonomy always contains two specific sorts: a greatest sort $\top$ denoting everything, the set of all objects, and a least sort $\bot$ denoting nothing, the empty set of object.

- To each defined sort in the signature, an order-consistent set of feature symbol is associated defining the sort's arity. Requiring order-consistent sort arities, which are sets of features, means that they are also partially ordered as sets, but by set containment ($\supseteq$); *i.e.*, the set ordering dual to set inclusion. This means in particular that the more specific a sort, the larger its arity. In particular, $\mathbf{arity}(\top) = \emptyset$ and $\mathbf{arity}(\bot) = \mathcal{F}$ (the set of all features).

[26] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

- A subset of the sort signature $\mathcal{O} \subseteq \mathcal{S}$, we shall refer to as the *object data base*, contains specific atoms that are minimal sorts; *i.e.*, some of the least sorts that are supersorts of $\perp$, the sort denoting the empty set .
- An object is represented as a ψ-term, an attributed instance of a concept represented in the knowledge base. An object's attribute is necessarily part of this object's feature set inherited from all its supersorts, but also must point to a data object in $\mathcal{O}$; *i.e.*, an object whose root sort is minimal and all of whose attributes refer also to minimal data objects in $\mathcal{O}$.

6.6.2 Naïve Bayes biases

Being based on observed user's accumulated choice history (which is statistical in nature), a recommendation system using Bayes Law and assuming all choices are independent (as does Naïve Bayes) will be subject to bias. We found the following statement to give a good explanation of statistical bias from a frequentist perspective.

> Suppose there is a model for the data Y that depends on a parameter θ and, for a particular experiment, there is a true value of the parameter, θ_0. You develop an estimator $\hat{\theta} = \hat{\theta}(Y)$, *i.e.*, the estimator is a function of the data Y. Then the bias is:
>
> $$bias(\hat{\theta}) = E_{Y|\theta_0}[\hat{\theta}(Y) - \theta_0]$$
>
> where the expectation is taken with respect to the randomness of the data Y for the given true value of the parameter θ_0 (and the subscript on the expectation attempts to make this explicit). As we are talking about an expectation over possible realizations of data, this is a frequentist concept.
>
> In the description above, I have not mentioned how the estimator arises. This estimator could be a method of moments, maximum likelihood, Bayes, or something else estimator. Thus, the concept of bias of an estimator is frequentist, but the estimator itself could arise from a Bayesian analysis.
>
> jaradniemi
> Answer to *"Is bias a frequentist concept or a Bayesian concept?"*
> May 14, 2017 at 16:04

The method used by Bayesian-based learning methods is called *naïve* because is makes the assumption that all the attributes of the probabilistic objects are mutually independent. This is far from being the case in most actual situations, and the source of a lot of prediction inaccuracy. A good illustration of the influence of Bayes Law on accuracy of predictions is available in [158].

A Naïve Bayes method inevitably creates some biases in the choices offered to a user. These biases are inherited from the assumption that because some specific products were accessed, recommendations that are "naïvely" similar will be given higher preference. Which explains the increasingly selective choice of recommendations User **You Sir** will be receiving, this selectivity applying to object categories populated with objects, each of which is an instance representing a specific product's Uniform Resource Identifier (URI).[27]

[27] https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

While most such biases may be appreciated by User **You Sir** (automatic determination of one's tastes in music, humor, politics, *etc.*, ...), it also prevents one's access to an immense potential of other targets that might be of high interest to this user.

Using fuzziness to model semantic proximity of $\mathcal{OSF}$ concepts and data objects, and combining that to classical Bayesian probability evaluation system, gives an "intelligent" way to rebalance undue biases, while keeping some semantic correlations among concepts and subjects of interest to a user [126].

6.6.3 Fuzzy Bayesian reasoning

- Fuzzy Probabilities [91];
- Fuzzy Probability Theory [91];
- Fuzzification of crisp probabilistic domains [80];
- Bayesian Inference With Adaptive Fuzzy Priors and Likelihoods [131], Fuzzy Bayesian Inference [169];
- Bayesian Decisions and Fuzzy Logic [85];

Chapter 7

Version of April 10, 2020

Conclusion

We conclude this monograph with three synthetic statements. Section 7.1 summarizes the essential contribution of the work we presented. Section 7.2 makes a point concerning common misconceptions about the nature and utility of fuzzy concepts and reasoning. Section 7.3 explains on what task we need to focus our next effort.

7.1 What Have We Done?

We overviewed several ways to fuzzify $\mathcal{OSF}$ constraint logic. We hope to have provided enough evidence that what we describe in this book is to benefit Information Retrieval as well as Machine Learning when fuzzy-$\mathcal{OSF}$ can provide a precious initial focusing step prior to exploiting number-analytical techniques used in Inductive Logic Programming [147] or Bayesian Nets [92].

> ***Authors' comment:*** *This last point should perhaps be detailed somewhat talking about Yutaka Sasaki's work: [150], [148], [147], [149], [26]. (Or perhaps just mention and cite his work here, and then elaborate in a separate paper on fuzzy $\mathcal{OSF}$ learning with him as co-author?)*

We also discussed several fuzzy versions of related topics, from (Lattice) Algebra, to Automata Theory, to (Order-Sorted Feature) Logics, to Object-Oriented Graph Data Structures, to fuzzy approximation thereof.

7.2 Our Fuzzy Word for the Wise?

This is a short point-of-view section addressing the question:

> "*Why should any measure in the* $[0.0, 1.0]$ *real interval be interpreted exclusively as probability?*"

It summarizes the mind-opening *set-as-points* geometric arguments presented by USC's Bart Kosko is this mathematically immaculate article, published thirty-year ago [!], and entitled "Fuzziness *vs.* Probability" [110].

7.3 Where Do We Go from Here?

The most immediate avenue of research that the issues we discussed open up is the design (specification and implementation) of a $\mathcal{CLP}$ language that would be the "least upper bound" of LOGIN [19] (and later $\mathcal{LIFE}$ [7, 17]) with a fuzzy Logic Programming language; *e.g.*, [102] or [63]. Such a language, with access to distributed databases, would facilitate efficient approximate reasoning for query resolution as well as learning by approximate knowledge acquisition as fuzzy order-sorted feature structures. All the applications enabled by Fuzzy Logic Programming on one side and $\mathcal{OSF}$ Logic Programming on the other could then each benefit as each would thereby gain even more expressiveness and flexibility for the processing of approximate structured knowledge on massive data. While fuzzy $\mathcal{OSF}$ unification (the conjunctive connective) is the key to *deduction* (as used in logical or function rule invocation), fuzzy $\mathcal{OSF}$ generalization (the disjunctive connective) is the key to *induction* (as used in learning by abstraction when extrapolating knowledge from data).

The next step, of course, is the development of fuzzified applications: from Information Retrieval, to Natural Language, to Knowledge Processing. The potential is immense.

Concomittantly are all the pragmatic issues: efficient implementation (preprocessing, abstract machine compiling, interfacing to constraint-solving and fuzzy-set libraries, *etc.*).

Finally, as one can easily gather from the material reviewed in Chapter 6, the possibilities of extending and applying techniques we exposed in this book are legion.

Appendix A

Version of April 10, 2020

Background Material

This appendix is a summary of general formal notions, terminology, and notation we use or refer to in this monograph. Section A.1 gives a definitional ontology of monadic and dyadic algebraic structures in the form of conceptual "*is-a*" taxonomies. It may be useful to refer to as a terminological guide map giving a global picture of how the varied structures involved relate to one another. Section A.2 reviews basic definitions and properties of $\mathcal{FOT}$ substitutions represented, as in this work, as finitely non-identical variable-to-term mappings. Section A.3 contains the procedural $\mathcal{FOT}$ generalization algorithms as formulated in 1970, one by John Reynolds and the other by Gordon Plotkin.

A.1 A Definitional Ontology of Algebraic Structures

We shall be concerned with algebraic structures derived on a set with one internal binary operation (*monadic structures*) and those derived on a set with two internal binary operations (*dyadic structures*) defned on them.

A.1.1 Monadic structures

Figure A.1 shows an "*is-a*" taxonomy for the monadic algebraic structures that are defined below. This taxonomy means that each monadic structure (a node in this graph) inherits the characteristic algebraic properties of all its super-structures (*i.e.*, any node following the "*is-a*" arc paths).

DEFINITION A.1 (MONADIC STRUCTURE) *A monadic structure* $\langle D, \star \rangle$ *consists of a set* D *of elements — the domain — with an internal binary operation:*

$$\star : D \times D \to D. \tag{A.1}$$

In a monadic structure, the operation $\star$ has an associated *prefix relation* defined for all $x, y \in D$ as:

$$x \prec_\star y \text{ iff } \exists z \in D,\ x \star z = y. \tag{A.2}$$

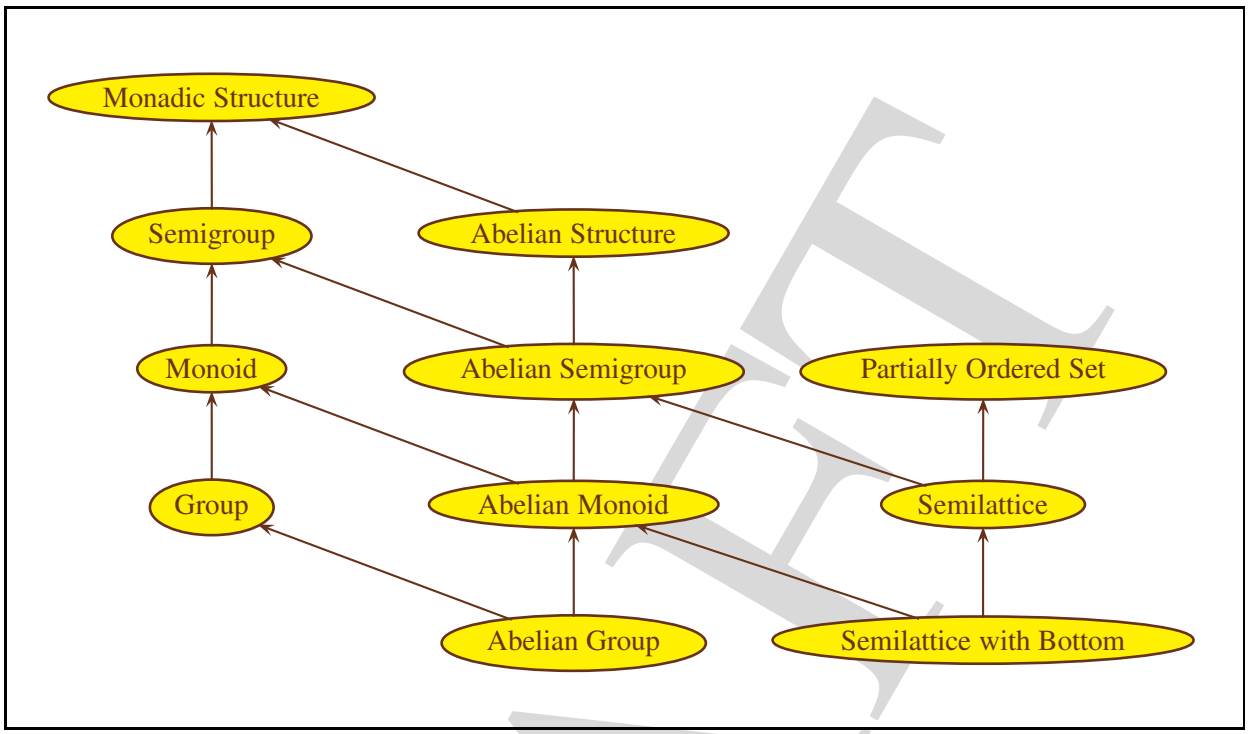


Figure A.1: Taxonomy of monadic algebraic structures

DEFINITION A.2 (SEMIGROUP) *A semigroup $\langle D, \star \rangle$ is a monadic structure with domain D whose operation $\star$ (A.1) is associative. That is, for all $x, y, z \in D$:*

$$x \star (y \star z) = (x \star y) \star z. \tag{A.3}$$

Note that in a semigroup $\langle D, \star \rangle$, the prefix relation $\prec_\star$ is always transitive (by virtue of associativity of $\star$). However, but it is not necessarily reflexive.

DEFINITION A.3 (MONOID) *A monoid $\langle D, \star, \epsilon \rangle$ is a semigroup $\langle D, \star \rangle$ with a special element $\epsilon \in D$, called a unit, such that, for all $x \in D$:*

$$x \star \epsilon = \epsilon \star x = x. \tag{A.4}$$

Note that in a monoid $\langle D, \star, \epsilon \rangle$, the prefix relation $\prec_\star$ is also reflexive (by virtue of the unit element). Therefore, it is a preorder, and is sometimes called the monoid's *prefix approximation*.

DEFINITION A.4 (GROUP) *A group $\langle D, \star, \epsilon \rangle$ is a monoid such that any element x has an inverse. That is, for any $x \in D$, there exists a (necessarily unique) $x^{-1} \in D$ such that:*

$$x \star x^{-1} = x^{-1} \star x = \epsilon. \tag{A.5}$$

DEFINITION A.5 (ABELIAN STRUCTURE) *An Abelian structure is any of the foregoing monadic structures whose operation $\star$ (A.1) is commutative. That is, for all $x, y \in D$:*

$$x \star y = y \star x. \tag{A.6}$$

Thus, we speak of an *Abelian* operation, an *Abelian* semigroup, an *Abelian* monoid, an *Abelian* group, *etc.*, ... Alternatively, the more suggestive adjective "commutative" is sometimes preferred to "Abelian."

DEFINITION A.6 (SEMILATTICE) *A semilattice* $\langle D, \star \rangle$ *is a commutative semigroup such that* $\star$ *is idempotent; i.e., for all* $x \in D$*:*

$$x \star x = x. \tag{A.7}$$

A natural partner to the $\star$ operation is the relation defined as $\leq_\star$ on D by:

$$\forall x, y \in D,\ x \leq_\star y \ \text{ iff } \ x \star y = y. \tag{A.8}$$

The relation $\leq_\star$ is called the *semilattice ordering* and indeed defines a partial order on D. Namely, $\leq_\star$ is reflexive (by idempotence of $\star$), anti-symmetric (by commutativity of $\star$) and transitive (by associativity of $\star$).

In a semilattice $\langle D, \star \rangle$, the prefix relation $\prec_\star$ is also an ordering and furthermore it coincides with the semilattice ordering. Namely:

THEOREM A.1 (ALGEBRAIC APPROXIMATION ORDERING) $\forall x, y \in D,\ x \prec_\star y$ *iff* $x \leq_\star y$.

PROOF Assume that $x \leq_\star y$. By definition, this means that $x \star y = y$. Thus, it is clear that $\exists z,\ x \star z = y$ (taking $z = y$). Therefore, $x \prec_\star y$.

Now assume that $x \prec_\star y$. Then, by definition, $x \star z_{xy} = y$ for some $z_{xy} \in D$. Hence,

$$\begin{aligned} x \star y &= x \star (x \star z_{xy}) && \text{(replacing } y \text{ by its value)} \\ &= (x \star x) \star z_{xy} && \text{(associativity)} \\ &= x \star z_{xy} && \text{(idempotence)} \\ &= y \end{aligned}$$

and so, $x \leq_\star y$. □

Note that $\star$ is automatically a *supremum* operation for its semilattice ordering; namely:

THEOREM A.2 (ALGEBRAIC APPROXIMATION SUPREMUM) *For all* $x, y, z \in D$*:*

$$\textit{if } \ y \leq_\star x \ \textit{ and } \ z \leq_\star x \ \textit{ then } \ y \star z \leq_\star x. \tag{A.9}$$

PROOF Assume that $y \leq_\star x$ and $z \leq_\star x$; then,

$y \star x = x$	by (A.8)	(a)
$z \star x = x$	by (A.8)	(b)
$(y \star x) \star (z \star x) = x \star x$	by (a) and (b)	
$(y \star x) \star (z \star x) = x$	by (A.7)	
$(y \star z) \star (x \star x) = x$	by (A.3) and (A.6)	
$(y \star z) \star x = x$	by (A.7)	
$y \star z \leq_\star x$	by (A.8).	

□

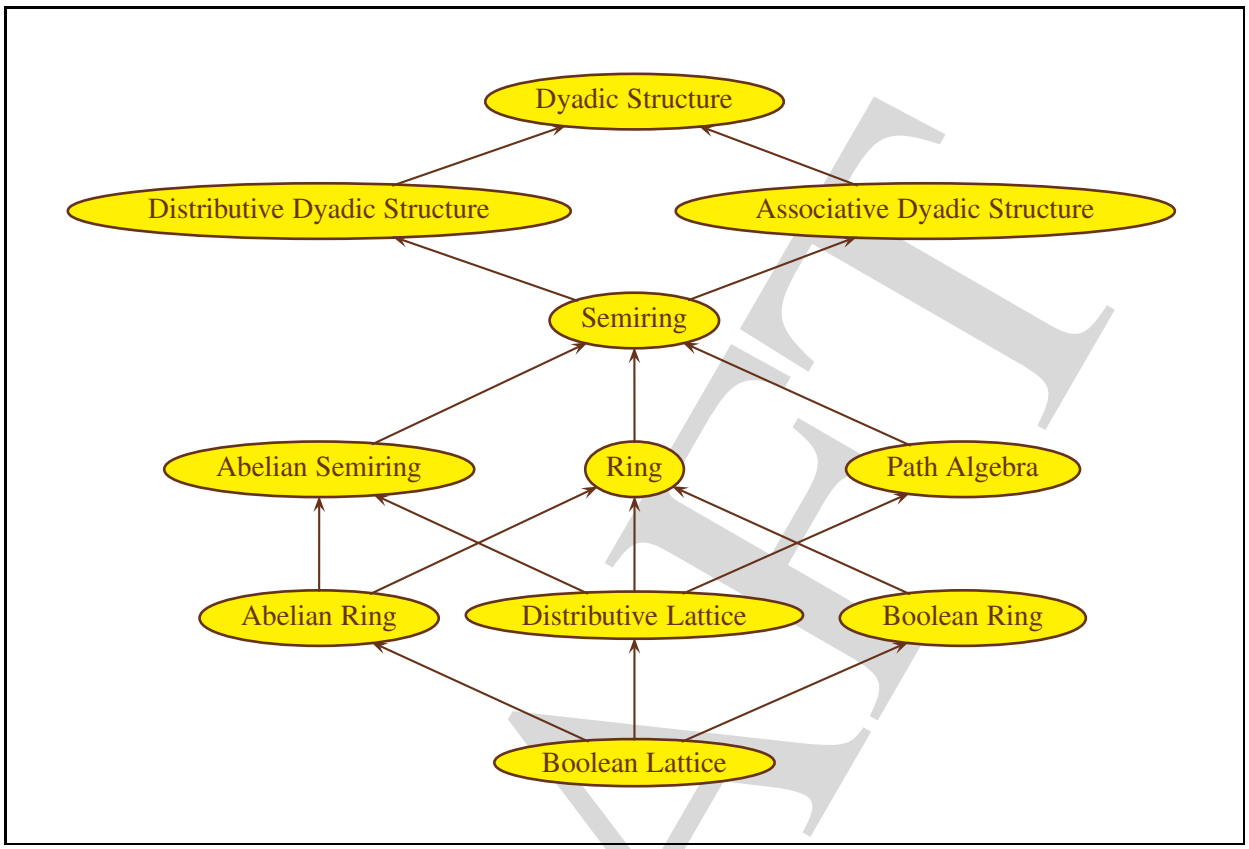


Figure A.2: Taxonomy of dyadic algebraic structures

Finally, note that if a semilattice $\langle D, \star \rangle$ is also a monoid $\langle D, \star, \epsilon \rangle$, Equation (A.8) entails that ϵ is the (necessarily unique) *least element* of D for $\leq_\star$. Then, it is sometimes written as $\bot$ (and called *bottom*). Thus, a semilattice with bottom can also be described as an idempotent Abelian monoid.

A.1.2 Dyadic structures

The dyadic algebraic structures shown in Figure A.2 are defined below. As for monadic algebras, this taxonomy means that each dyadic structure inherits the characteristic algebraic properties of its super-structure.

DEFINITION A.7 (DYADIC STRUCTURE) *A dyadic structure $\langle D, \star, * \rangle$ is a pair of monadic structures $\langle D, \star \rangle$ and $\langle D, * \rangle$ sharing the same domain D.*

In the notation used for an abstract structure, the particular symbols that denote the operation and unit element (if it is a monoid), are, of course, generic. Thus, in our definitions so far, we have used $\star$ for the operation, and ϵ for the unit element. Clearly, however, other symbols could be used instead — what matters is that the chosen symbols substituted for $\star$ and ϵ obey the appropriate equations. This being said, the familiar arithmetic operation symbols $+$ and $\times$, with associated unit symbols $\emptyset$ and $\mathtt{1}$, respectively, are sometimes used as generic symbols, despite their conventional

arithmetic meaning. Generally, this is to suggest that the structure at hand will behave as, or mostly as, in familiar arithmetic. The adjective *additive* (resp., *multiplicative*) is then used to designate properties of a structure whose operation is written $+$ (resp., $\times$).

Many common dyadic structures combine they additive and multiplicative operations using *distributivity* of the multiplication over the addition — this is a characteristic of *semirings*.

DEFINITION A.8 (SEMIRING) *A semiring $\langle D, +, \emptyset, \times, \mathtt{1}\rangle$ is a dyadic (additive and multiplicative) structure on a single set D such that:*

- *$\langle D, +, \emptyset\rangle$ is a commutative monoid;*
- *$\langle D, \times, \mathtt{1}\rangle$ is a monoid;*
- *$\times$ is distributive over $+$; that is, for all $x, y, z \in D$:*

$$x \times (y + z) = (x \times y) + (x \times z) \tag{A.10}$$

and

$$(x + y) \times z = (x \times z) + (y \times z). \tag{A.11}$$

To distinguish between the two operations's unit elements in a semiring, the additive unit $\emptyset$ is referred to as the *zero* element, and the multiplicative unit $\mathtt{1}$ as the *unit* element.

A semiring is an *Abelian* (or *commutative*) semiring if its multiplicative operation $\times$ is commutative (*i.e.*, if $\langle D, \times, \mathtt{1}\rangle$ is a commutative monoid).

A *path algebra* is *the* appropriate algebraic structure to handle path problems in graphs and networks [57].

DEFINITION A.9 (PATH ALGEBRA) *A path algebra $\langle D, +, \emptyset, \times, \mathtt{1}\rangle$ is a semiring such that:*

- *$+$ is idempotent; i.e., for all $x \in D$:*

$$x + x = x; \tag{A.12}$$

- *$\emptyset$ is absorptive for $\times$; i.e., for all $x \in D$:*

$$x \times \emptyset = \emptyset \times x = \emptyset; \tag{A.13}$$

In other words, a path algebra is a semiring that is also an additive semilattice as well as a $\emptyset$-absorptive multiplicative semigroup.

We define a (possibly empty) *path* of a graph $G = (V, A)$, where V is a finite set of vertices and $A \subseteq V \times V$ is a set of arcs, as a (possibly empty) ordered sequence of n arcs ($n \geq 0$) $\langle v_1, v_2\rangle, \ldots, \langle v_n, v_{n+1}\rangle$, such that $\forall i \in \{2, \ldots, n-1\},\ v_i = v_{i+1}$. A *cycle* is such a path where $v_1 = v_{n+1}$. A *simple path* has *no arc* occurring more than once. Hence, if a path is *not simple* then it must contain a cycle. An *elementary path* has *no vertex* occurring more than *twice*. Therefore, any elementary path is a simple path that contains no cycle.

Path Problem	Path Algebra Name	Domain	Sum	Zero	Product	One
Determination of accessible sets	$\mathbf{P}_1$	$\{0,1\}$	$\max(x,y)$	0	$\min(x,y)$	1
Determination of shortest paths	$\mathbf{P}_2$	$\mathbb{R}$	$\min(x,y)$	$+\infty$	$x+y$	0.0
Critical (longest) paths	$\mathbf{P}_3$	$\mathbb{R}$	$\max(x,y)$	$-\infty$	$x+y$	0.0
Most reliable paths	$\mathbf{P}_4$	$[0.0,1.0]$	$\max(x,y)$	0.0	$x\times y$	1.0
Paths with greatest capacity	$\mathbf{P}_5$	$\mathbb{R}^+$	$\max(x,y)$	0.0	$\min(x,y)$	$+\infty$
Listing of all paths	$\mathbf{P}_6$	$\mathcal{P}(\Sigma^*)$	$X\cup Y$	$\emptyset$	$X\cdot Y$	$\{\epsilon\}$
Listing of simple paths	$\mathbf{P}_7$	$\mathcal{P}(\mathcal{S}(\Sigma^*))$	$X\cup Y$	$\emptyset$	$X\cdot Y$	$\{\epsilon\}$
Listing of elementary paths	$\mathbf{P}_8$	B	$\underline{\mathrm{b}}(X\cup Y)$	$\emptyset$	$X\cdot Y$	$\{\epsilon\}$

Table A.1: Some network path problems formulated as equation-solving in specific path algebras

A path algebra is so-named because, as listed in Table A.1, many graph-theoretic path problems in networks consisting of (arc-)labeled graphs with labels coming from a set having a path-algebra structure can be formulated as solving systems of simultaneous fix-point linear equations where the unknowns are the vertices and the coefficients are the labels on the arcs.[1]

In Table A.1, $[0.0,1.0]$ denotes the closed unit interval in $\mathbb{R}$; $\mathbb{R}$ denotes the set of real numbers, including both $\pm\infty$; and, $\mathbb{R}^+$ denotes the set of non negative elements of $\mathbb{R}$. The powerset $\mathcal{P}(S)$ of a set S is the set of all the subsets of S. For a set S of strings on a finite alphabet Σ, the notation $\mathcal{S}(S)$ denotes the set of all the *simple strings* of S; *i.e.*, in which no symbol of Σ occurs more than once. The *set-concatenation* of two sets of strings S_1 and S_2, is the set of strings $S_1\cdot S_2 \stackrel{\text{def}}{=} \{\, s_1\cdot s_2 \,|\, s_1\in S_1 \text{ and } s_2\in S_2 \,\}$; *i.e.*, the set of all strings that are the concatenation of a string in S_1 and a string in S_2. Given a language $B\subseteq\Sigma^*$, a string $s\in\Sigma^*$ is *basic* for B iff B does not contain any substring of s (including ϵ). Given any set of strings S, we use the expression $\underline{\mathrm{b}}(S)$ to denote the subset of S of strings that are basic for S; *viz.*, $\underline{\mathrm{b}}(S) \stackrel{\text{def}}{=} \{\, s\in S \,|\, s \text{ is basic for } S \,\}$. In other words, $\underline{\mathrm{b}}(S)$ is the subset of S obtained from S after removing any string that is a substring of another string in S (including ϵ). Note in particular that both the empty language $\emptyset$ and the one-element language $\{\epsilon\}$ are elements of any basic language B.

Therefore, all the graph-path problems in Table A.1 can be formulated as solving a system of fix-point linear equations in a path algebra. This is computationally possible in a path algebra because quasi-inverses exist and may be computed since all iterated self-composition involved in the computation of a $*$-closure converges to a stable limit in a finite number of iterations [13].

[1] This table is adapted from [57] (*op. cit.*, Table 3.1, Page 86).

DEFINITION A.10 (RING) *A ring is a special case of a semiring. In fact, a ring structure is to a group what a semiring structure is to a monoid. Indeed, a ring* $\langle D, +, \emptyset, \times, \mathtt{1}\rangle$ *is a dyadic (additive and multiplicative) structure on a set* D *such that:*

- $\langle D, +, \emptyset\rangle$ *is a commutative group;*
- $\langle D, \times, \mathtt{1}\rangle$ *is a group;*
- *the multiplicative operation* $\times$ *is distributive over the additive operation* $+$*; that is, Equations* (A.10) *and* (A.11) *hold for all* $x, y, z \in D$*.*

A ring is an Abelian (or commutative) ring if its multiplicative operation $\times$ is commutative (*i.e.*, if $\langle D, \times, \mathtt{1}\rangle$ is a commutative group).

DEFINITION A.11 (LATTICE) *A lattice* $\langle D, +, \times\rangle$ *is a dyadic structure such that:*

- $\langle D, +\rangle$ *is a semilattice (called its additive semilattice);*
- $\langle D, \times\rangle$ *is a semilattice (called its multiplicative semilattice);*
- *its two operations are mutually absorptive; i.e., for all* $x, y \in D$*:*

$$x + (x \times y) = x = x \times (x + y). \tag{A.14}$$

Thus, the structure of a lattice is symmetric with respect to its two operations in the sense that $\langle D, +, \times\rangle$ is a lattice iff $\langle D, \times, +\rangle$ is a lattice. This important property is called *duality*. It makes a valid statement equally valid when changing every additive part into its multiplicative counterpart, and *vice versa*.

Note that a lattice is partially ordered both as an additive semilattice and as a mutiplicative semilattice. In fact, it is easy to see that the two partial orders are mutual inverses. That is,

$$\leq_+ \; = \; \leq_\times^{-1}, \tag{A.15}$$

and thus also, by duality:

$$\leq_\times \; = \; \leq_+^{-1} . \tag{A.16}$$

By convention, because the additive and multiplicative orderings of a lattice are mutual inverses, we write simply $\leq$ for $\leq_+$ and $\geq$ for $\leq_\times$. Thus, if a lattice is an additive (resp., multiplicative) monoid, $\emptyset$ is the least (resp., $\mathtt{1}$ is the greatest) element for $\leq$ and is often referred to as "bottom" (resp., "top") and sometimes written "$\bot$" (resp., "$\top$").

Note also that if a lattice $\langle D, +, \times\rangle$ is an additive monoid $\langle D, +, \emptyset\rangle$, then $\emptyset$ is necessarily absorptive for $\times$; *i.e.*, Equation (A.13) holds for all $x \in D$. Dually, if a lattice $\langle D, +, \times\rangle$ is a multiplicative monoid $\langle D, \times, \mathtt{1}\rangle$, then $\mathtt{1}$ is necessarily absorptive for $+$; *i.e.*, Equation (A.17) holds for all $x \in D$:

$$x + \mathtt{1} = \mathtt{1} + x = \mathtt{1}. \tag{A.17}$$

It is important to realize that a lattice is neither an instance of, nor is it more general than, a semiring (it lacks distributivity). However, it is easy to show that the following "sub-distributive" inequality holds in a lattice:

THEOREM A.3 (SUBDISTRIBUTIVITY) *Let $\mathcal{L} = \langle D, +, \times \rangle$ be a lattice. Then, for all x, y and z in D:*

$$x \times (y + z) \geq (x \times y) + (x \times z) \tag{A.18}$$

and, dually:

$$x + (y \times z) \leq (x + y) \times (x + z). \tag{A.19}$$

PROOF We need only establish Inequality (A.19); the proof of Inequality (A.18) is the same up to duality.

Let x, y, and z be arbitrary elements of a lattice $\langle D, +, \times \rangle$. Clearly, we have:

$$x \leq x + y \tag{A.20}$$

(since $x + (x + y) = x + y$). Similarly,

$$x \leq x + z. \tag{A.21}$$

Since $\times$ is an infimum operation for $\leq$, it follows from Inequalities (A.20) and (A.21) that:

$$x \leq (x + y) \times (x + z). \tag{A.22}$$

On the other hand, we also have:

$$y \times z \leq y \leq x + y \tag{A.23}$$

and

$$y \times z \leq z \leq x + z. \tag{A.24}$$

Again, because $\times$ is an infimum operation for $\leq$, Inequalities (A.23) and (A.24) imply that:

$$y \times z \leq (x + y) \times (x + z). \tag{A.25}$$

Finally, because $+$ is a supremum operation for $\leq$, Inequalities (A.22) and (A.25) imply Inequality (A.19). □

A *distributive lattice* is a lattice in which equality, rather than $\leq$, holds in (A.18) for all x, y and z (or equivalently, by duality, if equality, rather than $\geq$, holds in Inequality (A.19)). Thus, a distributive lattice with top and bottom is both an additive and a multiplicative commutative semiring. That is, Equation (A.10) holds for all $x, y, z \in D$ (and so does (A.11), by commutativity of $\times$).[2]

Finally, note that a distributive lattice with top and bottom is simultaneously an additive and a multiplicative path algebra.

[2] This is equivalent, by duality, to the additive operation $+$ being also distributive over the multiplicative operation $\times$; that is:

$$x + (y \times z) = (x + y) \times (x + z) \tag{A.26}$$

or, by commutativity of $+$:

$$(x \times y) + z = (x + z) \times (y + z). \tag{A.27}$$

DEFINITION A.12 (BOOLEAN RING) *A boolean ring is a ring in which any element admits a (necessarily unique) complement with respect to the additive and multiplicative operations. That is, for any $x \in D$, there exists a unique $\bar{x} \in D$ such that:*

$$x + \bar{x} = \bar{x} + x = \mathbb{1}, \tag{A.28}$$

and

$$x \times \bar{x} = \bar{x} \times x = \emptyset. \tag{A.29}$$

A.2 First-Order Term Substitutions

This section gives basic terminology and properties of $\mathcal{FOT}$ substitutions as defined in Chapter 3, Section 3.3, where the set-theoretic definition of substitutions as finitely non-identical variable-to-term mappings is given as Expression (3.2);[3] that is:

$$\begin{aligned} \sigma\theta \quad \stackrel{\text{def}}{=} \quad & \big(\{\, t\theta/X \mid t/X \in \sigma \,\} \setminus \{\, X/X \mid X \in \mathbf{dom}(\sigma) \,\}\big) \\ & \cup \\ & \big(\theta \setminus \{\, u/Y \mid Y \in \mathbf{dom}(\sigma) \,\}\big). \end{aligned}$$

LEMMA A.1 *Given two substitutions σ and θ in* **SUBST**$_{\mathcal{T}}$*, the operation defined by Expression (3.2) always results in a substitution in* **SUBST**$_{\mathcal{T}}$*.*

> **PROOF** It must be verified that, given σ and θ two finitely non-identical mappings from $\mathcal{V}$ to $\mathcal{T}$, the notation $\sigma\theta$ defined in set-theoretic terms from the set structure of σ and θ by Expression (3.2) always results in a finitely non-identical mapping from $\mathcal{V}$ to $\mathcal{T}$. This is an elementary exercise from the very set-theoretic definition of substitution composition given as Expression (3.2). □

LEMMA A.2 *For any term t in $\mathcal{T}$ and any substitutions σ and θ in* **SUBST**$_{\mathcal{T}}$*, the expression $\sigma\theta$ defined by Expression (3.2) is a substitution that has the same effect as first applying σ to t, and then applying θ to the result; that is, $\forall\, t \in \mathcal{T}, \forall\, \sigma \in$* **SUBST**$_{\mathcal{T}}, \forall\, \theta \in$* **SUBST**$_{\mathcal{T}}$*, $t(\sigma\theta) = (t\sigma)\theta$.*

> **PROOF** Expression (3.2) consists of two parts of a (disjoint) set union. The first part of this union consists in the set of pairs t/X in σ transformed into the set of pairs $t\theta/X$ for each each pair t/X in σ. This has for effect to "capture" any potential variables in $\mathbf{var}(t\sigma) \cap \mathbf{dom}(\theta)$ by mapping directly to $t\sigma\theta$ any variable mapped to t by σ. This corresponds to precomputing the necessary "shortcut" of instantiating X directly into to $t\sigma\theta$ for all such concerned variables in $\mathbf{dom}(\theta)$. Note that since this may possibly introduce identical pairs X/X, which must then be eliminated.

[3] See Page 22.

The second part of the union in Expression (3.2) simply completes the resulting substitution with pairs t/Y in θ concerning those variables Y which are not affected by σ (*i.e.*, all $Y \in \mathbf{dom}(\theta)$ such that $Y \notin \mathbf{dom}(\sigma)$). Indeed, these variables are taken care of in the first part in the terms mapping the variables in $\mathbf{dom}(X)$ by further instantiating by θ as need be.

These two cases clearly cover the only possibilties for variable mapping by σ and θ, and by construction in each case, this results in a finite set of term/variable pairs, thus completely specified by Expression (3.2) on all $\mathcal{V}$, when applied to any term t, has the same effect of first applying σ to t and then applying θ to the result. □

COROLLARY A.1 *Substitution composition as defined by Expression* (3.2) *is an associative operation; i.e., for all* σ, θ, *and* δ *in* **SUBST** $_{\mathcal{T}}$, $\sigma(\theta\delta) = (\sigma\theta)\delta$.

PROOF Let t be any term in $\mathcal{T}$, and σ, θ, and δ be three substitutions in **SUBST** $_{\mathcal{T}}$. Applying Lemma A.2 successively, we have $t(\sigma(\theta\delta)) = (t\sigma)(\theta\delta) = ((t\sigma)\theta)\delta = (t(\sigma\theta))\delta = t((\sigma\theta)\delta)$. Since both sides applied to any term are equal, this means that $\sigma(\theta\delta) = (\sigma\theta)\delta$. □

Note that, as a set of term/variable pairs, the substitution which is the identity everywhere on $\mathcal{V}$ is the empty set of pairs — which is why it is called the empty substitution and denoted as the empty set $\emptyset$. It is easy to verify that this empty substitution is also the unique identity element on **SUBST** $_{\mathcal{T}}$. Namely, for all substitution $\sigma \in$ **SUBST** $_{\mathcal{T}}$, $\sigma\emptyset = \emptyset\sigma = \sigma$ and if $\sigma\theta = \theta\sigma = \sigma$ for some $\theta \in$ **SUBST** $_{\mathcal{T}}$, then $\theta = \emptyset$. Therefore, **SUBST** $_{\mathcal{T}}$ with composition and $\emptyset$ is a monoid. Note finally that substitution composition is not commutative since in general $\sigma\theta \neq \theta\sigma$.[4] Therefore, the set **SUBST** $_{\mathcal{T}}$ with substitution composition is a non-commutative monoid.

Like all monoids, the set **SUBST** $_{\mathcal{T}}$ of substitutions inherits a relation $\preceq$ defined as follows.

DEFINITION A.13 $\sigma \preceq \theta$ *iff* $\exists\, \delta \in$ **SUBST** $_{\mathcal{T}}$ *s.t.* $\sigma = \theta\delta$.

The expression "$\sigma \preceq \theta$" is read "σ *refines* θ" or "θ *is more general than* σ."

LEMMA A.3 *The relation* $\preceq$ *is a preorder on the set of first-order term substitutions* **SUBST** $_{\mathcal{T}}$.

PROOF We must show that $\preceq$ is reflexive and transitive. **Reflexivity:** For any $\sigma \in$ **SUBST** $_{\mathcal{T}}$, there exists $\delta = \emptyset$ such that $\sigma = \sigma\delta$, which means by definition of $\preceq$ that $\sigma \preceq \sigma$. **Transitivity:** Assume $\sigma_1 \preceq \sigma_2$ and $\sigma_2 \preceq \sigma_3$; this means that there exist δ_1 and δ_2 such that $\sigma_1 = \sigma_2\delta_1$ and $\sigma_2 = \sigma_3\delta_2$. Replacing σ_2 by its value in the expression of σ_1, it comes as a result that $\sigma_1 = \sigma_3\delta_2\delta_1$. And so, there exists $\delta_3 = \delta_2\delta_1$ such that $\sigma_1 = \sigma_3\delta_3$; which means that $\sigma_1 \preceq \sigma_3$. □

Note that $\preceq$ is not an order relation because it is not anti-symmetric. Indeed, if we have both $\sigma \preceq \theta$ and $\theta \preceq \sigma$, this does not necessarily imply that $\sigma = \theta$. However, this defines an equivalence relation on substitutions.

[4] Take for example $\sigma = \{\, a/X \,\}$ and $\theta = \{\, b/X \,\}$, for which $\sigma\theta = \{\, a/X\}$ and $\theta\sigma = \{\, b/X\}$.

LEMMA A.4 *The relation* $\simeq \stackrel{def}{=} \preceq \cap \preceq^{-1}$ *is an equivalence on the set of substitutions* $\textbf{SUBST}_{\mathcal{T}}$.

PROOF Let us verify that $\simeq$ has three properties of an equivalence. **Reflexivity:** Clearly, for any $\sigma \in \textbf{SUBST}_{\mathcal{T}}$, $\sigma \simeq \sigma$ since this is equivalent to $\sigma \preceq \sigma$ and $\sigma \preceq \sigma$, which is always true since $\preceq$ is reflexive because it is a preorder. **Symmetry:** Also, for any $\sigma \in \textbf{SUBST}_{\mathcal{T}}$ and $\theta \in \textbf{SUBST}_{\mathcal{T}}$, if $\sigma \simeq \theta$, this is equivalent by definition to $\sigma \preceq \theta$ and $\theta \preceq \sigma$; which is also equivalent to $\theta \simeq \sigma$. Therefore, $\simeq$ is symmetric. **Transitivity:** Let us now assume that (1) $\sigma \simeq \theta$ and (2) $\theta \simeq \delta$. This implies in particular, by definition of $\simeq$ and $\simeq$: (1) ($\sigma \preceq \theta$ and $\theta \preceq \delta$), which by transitivity of $\preceq$ implies $\sigma \preceq \delta$; and (2) ($\delta \preceq \theta$ and $\theta \preceq \sigma$); which by transitivity of $\preceq$ implies $\delta \preceq \sigma$. Hence, we have both $\sigma \preceq \delta$ and $\delta \preceq \sigma$, which is equivalent to $\sigma \simeq \delta$. Therefore, $\simeq$ is transitive. □

DEFINITION A.14 *A variable renaming* ρ *is a substitution in* $\textbf{SUBST}_{\mathcal{T}} \cap (\mathcal{V} \to \mathcal{V})$ *that is injective. That is,*

- $\rho = \{X'_i/X_i\}_{i=1}^{n}$ *with* $X_i \in \mathcal{V}$ *and* $X'_i \in \mathcal{V}$*; and,*
- *if* $X_i \neq X_j$ *then* $X'_i \neq X'_j$*, for any* $i, j = 1, \ldots, n$ *such that* $i \neq j$.

COROLLARY A.2 *If both* $\sigma \preceq \theta$ *and* $\theta \preceq \sigma$*, this entails that* σ *and* θ *are equal up to a renaming of their variables. Namely,* $\exists\, \rho : \mathcal{V} \to \mathcal{V}$ *bijective such that* $\theta = \rho\sigma$ *and* $\sigma = \rho^{-1}\theta$.

PROOF If $\sigma \preceq \theta$ and $\theta \preceq \sigma$ then, by definition, there exist two substitutions ρ and ρ' such that $\sigma = \theta\rho$ and $\theta = \sigma\rho'$. In other words:

$$\begin{cases} \sigma = \sigma\,\rho\,\rho', \\ \theta = \theta\,\rho'\rho\,; \end{cases} \quad \text{which is equivalent to:} \quad \begin{cases} \rho\,\rho' = \emptyset, \\ \rho'\rho = \emptyset; \end{cases} \quad \text{and therefore to:} \quad \begin{cases} \rho = \rho'^{-1}, \\ \rho' = \rho^{-1}. \end{cases}$$

Note also that since ρ and ρ' are mutual inverses on $\mathcal{V}$, it must be that ρ and ρ' are injective. This follows from the axiom of functionality for ρ and ρ', which states that for every pair of variables X and X' in $\mathcal{V}$, if $X = X'$ then necessarily $X\rho = X'\rho$ and $X\rho' = X'\rho'$. But since ρ are mutual inverses on $\mathcal{V}$, this means that whenever $Y\rho' = Y'\rho'$ for any pair of variables Y and Y' in $\mathcal{V}$, then necessarily $Y\rho'\rho = Y'\rho'\rho$; *i.e.*, $Y\emptyset = Y'\emptyset$, and thus $Y = Y'$, which means that ρ' must be injective. The same reasoning in the other direction will entail that ρ must be injective as well. Note finally that ρ is also surjective on $\mathcal{V}$, since any variable $X \in \mathcal{V}$ is such that $X\rho'\rho = X$, therefore there exists $Y = X\rho'$ such that $Y\rho = X$. The same applies to ρ' in the other direction. Therefore, ρ and ρ' are bijective inverses. □

A.3 Reynolds-Plotkin $\mathcal{FOT}$ Generalization

The two essentially identical algorithms (up to notation) for the generalization of two $\mathcal{FOT}$s given by Reynolds [141] and Plotkin [135] in the same volume are reproduced verbatim in Figure A.3 and Figure A.4. As can be seen in these figures, each describes a procedural method computing the most specific $\mathcal{FOT}$ subsuming two given $\mathcal{FOT}$s in finitely many steps by comparing them simultaneously, and generating a pair of generalizing substitutions from a fresh variable wherever they disagree being scanned from left to right, each time replacing the disagreeing terms by the new variable everywhere they both occur in each term.

(4) If A and B are CAFs beginning with the same predicate symbol, then let $Z_1, Z_2, \ldots$ be a sequence of variables which do not occur in A or B, and obtain $A \sqcup B$ by the following *Anti-unification Algorithm*:*

(a) Set the variables $\bar{A}$ to A, $\bar{B}$ to B, ζ and η to the empty substitution, and i to zero.

(b) If $\bar{A}=\bar{B}$, exit with $A \sqcup B=\bar{A}=\bar{B}$.

(c) Let k be the index of the first symbol position at which $\bar{A}$ and $\bar{B}$ differ, and let S and T be the terms which occur, beginning in the kth position, in $\bar{A}$ and $\bar{B}$ respectively.

(d) If, for some j such that $1 \leqslant j \leqslant i$, $Z_j\zeta=S$ and $Z_j\eta=T$, then alter $\bar{A}$ by replacing the occurrence of S beginning in the kth position by Z_j, alter $\bar{B}$ by replacing the occurrence of T beginning in the kth position by Z_j, and go to step (b).

(e) Otherwise, increase i by one, alter $\bar{A}$ by replacing the occurrence of S beginning in the kth position by Z_i, alter $\bar{B}$ by replacing the occurrence of T beginning in the kth position by Z_i, replace ζ by $\zeta \cup \{S/Z_i\}$, replace η by $\eta \cup \{T/Z_i\}$, and go to step (b).

* This algorithm has been discovered independently by Mr Gordon Plotkin of the University of Edinburgh.

Figure A.3: Reynolds's $\mathcal{FOT}$ "anti-unification" algorithm ([141], pages 138–139)

Let W_1, W_2 be any two compatible words. The following algorithm terminates at stage *3*, and the assertion made there is then correct.
1. Set V_i to $W_i (i=1,2)$. Set ε_i to $\varepsilon (i=1,2)$. ε is the empty substitution.
2. Try to find terms t_1, t_2 which have the same place in V_1, V_2 respectively and such that $t_1 \neq t_2$ and either t_1 and t_2 begin with different function letters or else at least one of them is a variable.
3. If there are no such t_1, t_2 then halt. V_1 is a least generalization of $\{W_1, W_2\}$ and $V_1 = V_2$, $V_i \varepsilon_i = W_i (i=1,2)$.
4. Choose a variable x distinct from any in V_1 or V_2 and wherever t_1 and t_2 occur in the same place in V_1 and V_2, replace each by x.
5. Change ε_i to $\{t_i | x\} \varepsilon_i (i=1,2)$.
6. Go to *2*.

Figure A.4: Plotkin's $\mathcal{FOT}$ "least generalization" algorithm ([135], page 155)

DRAFT—PLEASE DO NOT DISTRIBUTE

Appendix B

Version of April 10, 2020

$\mathcal{OSF}$ Extensions and Examples

- Appendix B.1 gives three useful additional decidable $\mathcal{OSF}$ constraints: partial features, extensional (*i.e.*, element-denoting) sort symbols, and aggregation. The first and second of these constraints convey implicit additional axioms that $\mathcal{FOT}$s verify as $\mathcal{OSF}$ terms.
- Appendix B.2 gives detailed examples of $\mathcal{OSF}$ lattice operations.

B.1 Other Decidable $\mathcal{OSF}$ Constraints

The set of $\mathcal{OSF}$-constraints normalization rules presented thus far may be extended with useful additional axioms that enable other constraints commonly enforced in object/class-based systems; *viz.*, *partial features*, *element constructors*, and *aggregation*. We next describe additional rules that enforce such additional axioms.

§ Partial Features

Let $\mathbf{dom} : \mathcal{F} \mapsto \mathbf{2}^{\mathcal{S}}$ associate to a feature f its *domain* $\mathbf{dom}(f)$, the set of maximal sorts in $\mathcal{S}$ for which f is defined. A feature f is said to be:

- *total* when $\mathbf{dom}(f) = \{\top\}$;
- *undefined* when $\mathbf{dom}(f) = \{\bot\}$;
- *partial* when it is neither undefined nor total.

Given a feature $f \in \mathcal{F}$, for each sort $s \in \mathbf{dom}(f)$, the *range* of f over s, denoted as $\mathbf{ran}_s(f) \in \mathcal{S}$, is the set of all maximal sorts of the possible values that feature f can take on sort s.[1] A possible $\mathcal{OSF}$-constraint normalization rule correctly enforcing such partial features is shown as Rule **Partial Feature** in Figure B.1.

Note however that Rule **Partial Feature** is non-deterministic, since there may be several incomparable maximal sorts making up the domain (or, for a sort in its domain, several ranges)

[1] Computational linguists, who have borrowed heavily from the $\mathcal{OSF}$ formalism to express HPSG grammars for natural-language processing, call this category of axioms "*feature appropriateness*" axioms (see, *e.g.* [56]).

PARTIAL FEATURE

$$\frac{[s \in \mathbf{dom}(f);\ \ s' \in \mathbf{ran}_s(f)] \qquad \phi\ \ \&\ \ X.f \doteq X'}{\phi\ \ \&\ \ X.f \doteq X'\ \ \&\ \ X : s\ \ \&\ \ X' : s'}$$

Figure B.1: Partial feature

for a partial feature when it has multiple domains and corresponding ranges (even for a single domain sort). One way to ease this issue is to introduce a new supersort (or use one if there is one) of all the domain sorts of a feature, and a new supersort (or use one if there is one) for all its range sorts. However, note that, semantically, this is not equivalent since there may be a loss of information since $f : \bigvee_{k\geq 2} d_k \to \bigvee_{k\geq 2} r_k$ is algebraically coarser than $f : \bigvee_{k\geq 2}(d_k \to r_k)$.[2] This way of enforcing, even if only partially, known or declared domains and ranges for features is to generate constraints to that effect systematically when dissolving an $\mathcal{OSF}$-term. Indeed, we can also redefine the $\mathcal{OSF}$-dissolution function φ defined by Equation (4.5) on Page 82, with:

$$\begin{array}{lcccccccc} \varphi(t) & \stackrel{\text{def}}{=} & X : s & \& & X.f_1 \doteq X_1 & \& & \ldots & \& & X.f_n \doteq X_n \\ & & & \& & X : s_1 & \& & \ldots & \& & X : s_n \\ & & & \& & X_1 : s'_1 & \& & \ldots & \& & X_n : s'_n \\ & & & \& & \varphi(t_1) & \& & \ldots & \& & \varphi(t_n) \end{array} \tag{B.1}$$

where $s_i \stackrel{\text{def}}{=} \bigvee \mathbf{dom}(f_i)$ (*i.e.*, the **lub** of all the known or declared maximal domains of f_i), and $s'_i \stackrel{\text{def}}{=} \bigvee_{s_i} \mathbf{ran}_{s_i}(f_i)$ (*i.e.*, the **lub** of all the known or declared ranges of f_i on domain s_i), for $i = 1, \ldots, n$.[3] This will include explicitly the most general feature domain/range constraints. Then, normalizing with the rules of Figure B.2 **PARTIAL FEATURE DOMAIN NARROWING** and **PARTIAL FEATURE RANGE NARROWING** will "finish the job," as each rule uses sort information on a feature's domain (resp., range) to further constrain its range (resp., domain) whenever this information is available. Importantly, the third side condition of each rule is to ensure finite termination of normalization by preventing repeated introductions of useless sort constraints, as is the case for example for Rule **PARTIAL FEATURE** — which, while being correct, is also non-deterministic as well as non-terminating!

All this special handling of partial features in the $\mathcal{OSF}$ machinery is done perhaps with the necessity of introducing many new symbols for missing unique **lub**s, but it can be done automatically and incurs no penalty for execution nor space in sort ordering checking nor lattice operations thanks to bit-encoding techniques [15]. On the other hand, in practice, this narrows features to their appropriate domain and range sorts, including eliminating spurious feature applications, even trivial ones such as, *e.g.*, typos.

[2] Can you see why? (*Hint:* a range sort depends on its feature's domain — try for $k = 2$.)

[3] Again, if an actual sort symbol does not exist in $\mathcal{S}$ for either **lub** expressions, a new one is introduced for s_i or s'_i.

PARTIAL FEATURE DOMAIN NARROWING

$$\frac{[\mathbf{dom}(f) = \{s\};\ \mathbf{ran}_s(f) = \{s'\};\ X : s'' \notin \phi \text{ with } s'' \preceq s] \quad \phi\ \&\ X.f \doteq X'\ \&\ X' : s'}{\phi\ \&\ X : s\ \&\ X.f \doteq X'\ \&\ X' : s'}$$

PARTIAL FEATURE RANGE NARROWING

$$\frac{[\mathbf{dom}(f) = \{s\};\ \mathbf{ran}_s(f) = \{s'\};\ X' : s'' \notin \phi \text{ with } s'' \preceq s'] \quad \phi\ \&\ X : s\ \&\ X.f \doteq X'}{\phi\ \&\ X : s\ \&\ X.f \doteq X'\ \&\ X' : s'}$$

Figure B.2: Partial-feature narrowing

§ ELEMENT CONSTRUCTORS

A sort denotes a set of values of the domain of interpretation. When this set is a singleton, the sort is assimilated to the value contained in the denoted singleton (*e.g.*, a number, a string, *etc.*). However, such data may also have structure; then, it is assimilated to a data constructor. Such a structure denotes an individual element only when all its subterms under a set of specific features do as well. For example, a pair constructor "**pair**" with features "**left**" and "**right**" will denote a single individual **pair** object only when both subterms under these two features denote each a single individual; otherwise, it denotes a set. Thus, the ψ-term $\mathtt{pair}(\mathtt{left} \to \mathtt{1}, \mathtt{right} \to \mathtt{2})$ denotes the *individual* object $\langle 1, 2\rangle$, whereas if the sort **nat** denotes the set of natural numbers $\mathbb{N}$, then the term $\mathtt{pair}(\mathtt{left} \to \mathtt{nat}, \mathtt{right} \to \mathtt{nat})$ denotes the *set* of all pairs whose left and right subterms are natural numbers (*viz.*, $\{\, \langle m, n\rangle \mid m \in \mathbb{N},\ n \in \mathbb{N} \,\}$). For such sorts, we must then ensure that only individual-denoting terms are uniquely represented. Let $\mathcal{E}$ (for "*element*," or "*extensional*," sorts) be the set of sorts in $\mathcal{S}$ that are element constructors. Define the *arity* $\mathbf{arity}(e)$ of such an element sort e giving its *feature arity* as a set of features — *i.e.*, $\mathbf{arity} : \mathcal{E} \mapsto \mathbf{2}^{\mathcal{F}}$. The set $\mathbf{arity}(e)$ is the set of features that completely determine the unique element of sort e. In other words, whenever all features of $\mathbf{arity}(e)$ denote singletons, then so does e. All such values ought to be uniquely identified. Note in passing that all atomic constants in $\mathcal{E}$ always have empty arity. For example, for any number n, $\mathbf{arity}(n) = \emptyset$. The $\mathcal{OSF}$-constraint normalization rule that enforces this uniqueness axiom on element sorts is called Rule **WEAK EXTENSIONALITY** as shown in Figure B.3.

With this rule, for example, if $\mathcal{S} = \{\top, \bot, \mathtt{nil}, \mathtt{cons}, \mathtt{list}, \mathtt{nat}, 0, 1, 2, \ldots\}$ such that $\mathtt{nil} < \mathtt{list}$, $\mathtt{cons} < \mathtt{list}$, $n < \mathtt{nat}$ for $n \in \mathbb{N}$ (where $<$ is the subsort ordering). Let $\mathcal{E} = \{\mathtt{nil}, \mathtt{cons}, n\},\ (n \in \mathbb{N})$, such that $\mathbf{arity}(\mathtt{nil}) = \emptyset$, $\mathbf{arity}(\mathtt{cons}) = \{\mathtt{head}, \mathtt{tail}\}$, and $\mathbf{arity}(n) = \emptyset$ for $n \in \mathbb{N}$. Then, the ψ-term:

$$X : \mathtt{cons}(\mathtt{head} \to \mathtt{1}, \mathtt{tail} \to \mathtt{nil})\ \ \&\ \ Y : \mathtt{cons}(\mathtt{head} \to \mathtt{1}, \mathtt{tail} \to \mathtt{nil})$$

WEAK EXTENSIONALITY

$$\frac{[s \in \mathcal{E};\ \forall f \in \mathbf{arity}(s), \{X.f \doteq Y, X'.f \doteq Y\} \subseteq \phi] \quad \phi\ \&\ X : s\ \&\ X' : s}{\phi\ \&\ X : s\ \&\ X \doteq X'}$$

Figure B.3: Weak extensionality

is normalized into:

$$X : \mathtt{cons}(\mathtt{head} \to \mathtt{1}, \mathtt{tail} \to \mathtt{nil})\ \ \&\ \ X \doteq Y$$

This rule is called *"weak"* because it can only enforce uniqueness of *acyclic* elements. Rules with a stronger condition working for cyclic terms are given next.

§ STRONG EXTENSIONALITY

Basically, the reason why Rule WEAK EXTENSIONALITY of Figure B.3 does not recognize singletons that are cyclic terms is that it works *inductively*. Doing so, it is well-founded only because it proceeds from leaves to their roots. However, for cyclic terms, there may be no leaf to proceed from. Consider, for example, an extensional sort $s \in \mathcal{E}$ such that $\mathbf{arity}(s) = \{f\}$, and the conjunction of cyclic terms:

$$X : s(f \to X)\ \ \&\ \ X' : s(f \to X') \tag{B.2}$$

or, even better, that of the mutually cyclic terms:

$$X : s(f \to X')\ \ \&\ \ X' : s(f \to X). \tag{B.3}$$

Now, $\mathbf{arity}(s) = \{f\}$ means that *"s denotes a singleton sort whenever its f feature denotes one as well."* Semantically, in both examples, variables X and X' denote therefore the same element (due to *all* the features in $\mathbf{arity}(s)$ being consistently sorted as singletons). However, the Rule WEAK EXTENSIONALITY does not transform either term (B.2) or term (B.3) into one where X and X' are equal as they should be as per the semantics of arity and extensionality. Therefore, this inductive manner of proceeding will not work for cyclic extensional terms such as these. The alternative is to proceed *coinductively*, from roots to leaves or previously processed nodes, while keeping a record of which extensional sorts appear with which variables, since such sorts denote single element. This is done by carrying an *occurrence context* as a set Γ of elements of the form $s : \{X_1, \ldots, X_n\}$, where $X_i \in \mathcal{V}$, for $i = 1, \ldots, n, (n \geq 0)$, where $s \in \mathcal{E}$ is extensional, and such that each such s may not occur more than once in any such occurrence context Γ. A *contexted rule* is one of the form:

(Rule Number) **RULE NAME** :

$$\frac{\mathit{Prior\ Context} \vdash \mathit{Prior\ Form}}{\mathit{Posterior\ Context} \vdash \mathit{Posterior\ Form}}\ [\,\mathit{Condition}\,]$$

Appropriate extensional sort occurrences record-keeping is thus achieved using contexted Rule *Extensional Variable* of Figure B.4. The "real" work is then done by contexted Rule **STRONG EXTENSIONALITY**. Using these two rules on weak normal forms will work as expected; *viz.*, it will merge any remaining potential cyclic extensional elements that denote the same individual.

EXTENSIONAL VARIABLE

$$\frac{\begin{bmatrix} X \notin V;\ \ s \in \mathcal{E};\ \ s' \in \mathcal{E}; \\ \forall f \in \mathbf{arity}(s),\ \{X.f \doteq X', X' : s'\} \subseteq \phi \end{bmatrix} \qquad \Gamma \uplus \{s : V, \ldots,\} \vdash \phi \ \&\ X : s}{\Gamma \uplus \{s : V \cup \{X\}, \ldots\} \vdash \phi \ \&\ X : s}$$

STRONG EXTENSIONALITY

$$\frac{[s \in \mathcal{E}] \qquad \Gamma \uplus \{s : \{X, X', \ldots\} \vdash \phi}{\Gamma \uplus \{s : \{X, \ldots\} \vdash \phi \ \&\ X \doteq X'}$$

Figure B.4: Strong extensionality

§ RELATIONAL FEATURES AND AGGREGATION

The $\mathcal{OSF}$ formalism deals with functional features. However, relational features may also come handy. A relational feature is a binary relation or, equivalently, a set-valued function. In other words, a multi-valued functional attribute may be aggregated into a set. Indeed, combining Rule **SORT INTERSECTION** with Rule **FEATURE FUNCTIONALITY** of Figure 4.6 enforces that a variable's sort, and hence value, may only be computed by intersection of consistent sorts. On the other hand, a relational feature denotes a set-valued function, and normalization must thus provide a means for aggregating mutually distinct values of a sort. This semantics can be accommodated with the following value aggregation rule, which generalizes Rule **SORT INTERSECTION**. The notation for the atomic constraint "$X : s$" is generalized to carry an optional value $e \in \mathcal{E}$ (*i.e.*, e is an extensional sort): "$X = e : s$" means "X has value e of sort s," where $X \in \mathcal{V}$, $e \in \mathcal{E}$, $s \in \mathcal{S}$. The shorthand "$X = e$" means "$X = e : \top$." When the sort $s \in \mathcal{S}$ is a commutative monoid $\langle \star, \mathbf{1}_\star \rangle$, the shorthand "$X : s$" means "$X = \mathbf{1}_\star : s$." The conditions (4.4) are then extended with:

$$\mathcal{A}, \alpha \ \models\ X = e : s \ \text{ iff } \ e^{\mathcal{A}} \in s^{\mathcal{A}} \ \text{ and } \ \alpha(X) = e^{\mathcal{A}}. \tag{B.4}$$

Thus, element values of a sort that denotes a commutative monoid $M = \langle \star, \mathbf{1}_\star \rangle$ may be composed using this monoid's operation. In particular, such a monoid operation may be that of a *set* constructor; *i.e.*, one that is associative, commutative, and idempotent. This is what Rule **VALUE AGGREGATION** of Figure B.5 accommodates.

VALUE AGGREGATION

$$\frac{[s, s' \text{ both subsorts of commutative monoid } \langle \star, \mathbf{1}_\star \rangle] \quad \phi \ \& \ X = e : s \ \& \ X = e' : s'}{\phi \ \& \ X = e \star e' : s \wedge s'}$$

Figure B.5: Aggregation

Note that Rule **VALUE AGGREGATION** is more general than need be for just accommodating aggregating a set (*i.e.*, a commutative idempotent free monoid) or a multiset (commutative but non-idempotent free monoid). Indeed, it also accommodates any other commutative monoids using aggregation operations such as min, max, sum, product, *etc.*, ... Thus, one may use this rule by using $\textbf{AGGREGATE}(f, s, m, \star, \mathbf{1}_\star)$ to declare the fact that when feature f is applied on (domain) sort s, it takes values in (range) sort m denoting a specific commutative monoid $\langle \star, \mathbf{1}_\star \rangle$ (*i.e.*, $s \in \mathbf{dom}(f)$ and $\mathbf{ran}_s(f) = m$). In other words,

$$X : s \ \& \ X.f \doteq Y \ \& \ Y = \mathbf{1}_\star : m. \tag{B.5}$$

Then, the rules **PARTIAL FEATURE DOMAIN/RANGE NARROWING** used in conjunction with Rule **VALUE AGGREGATION** of Figure B.5 will use such a declaration to proceed with the correct aggregation for so-declared features. For example, declaring:

$$\textbf{AGGREGATE}(\texttt{activities}, \texttt{person}, \texttt{activity}, \cup, \emptyset)$$

would ensure that whenever applied on sort **person** feature **activities** aggregates values of sort **activity** using set union ($\cup$). The default value is the empty set ($\emptyset$). If the feature **activities** of a **person** object is an individual element a of sort **activity**, it is assimilated to the singleton set $\{a\}$. Note however that we require a *commutative* monoid to ensure confluence of this rule with the other $\mathcal{OSF}$-constraint normalization rules in a non-deterministic normalization setting. In other words, the order in which the rules are applied does not matter on the outcome of the aggregation only when the monoid operation is commutative. This rule can also be used on non-commutative collection structures such as lists (free monoid), although the order of application may then result in different structures. Decidability results concerning the differences between attributive concepts using functional features *vs.* relation roles are reviewed in [151]. Aggregation has also been considered in the same setting in [41] with similar decidability results. This last work offers intriguing potential connections with the paradigm of declarative aggregation as described in [79] or [90] where a versatile computable algebraic theory of monoid comprehensions is defined in terms of monoid homomorphisms effective declarative aggregates where the aggregation can be any associative binary operation. As defined in [78], a Monoid Comprehension Calculus can be defined as a conservative extension of the λ-calculus with aggregates to specify an object-relational model enjoying algebraic properties that greatly facilitate query optimization.

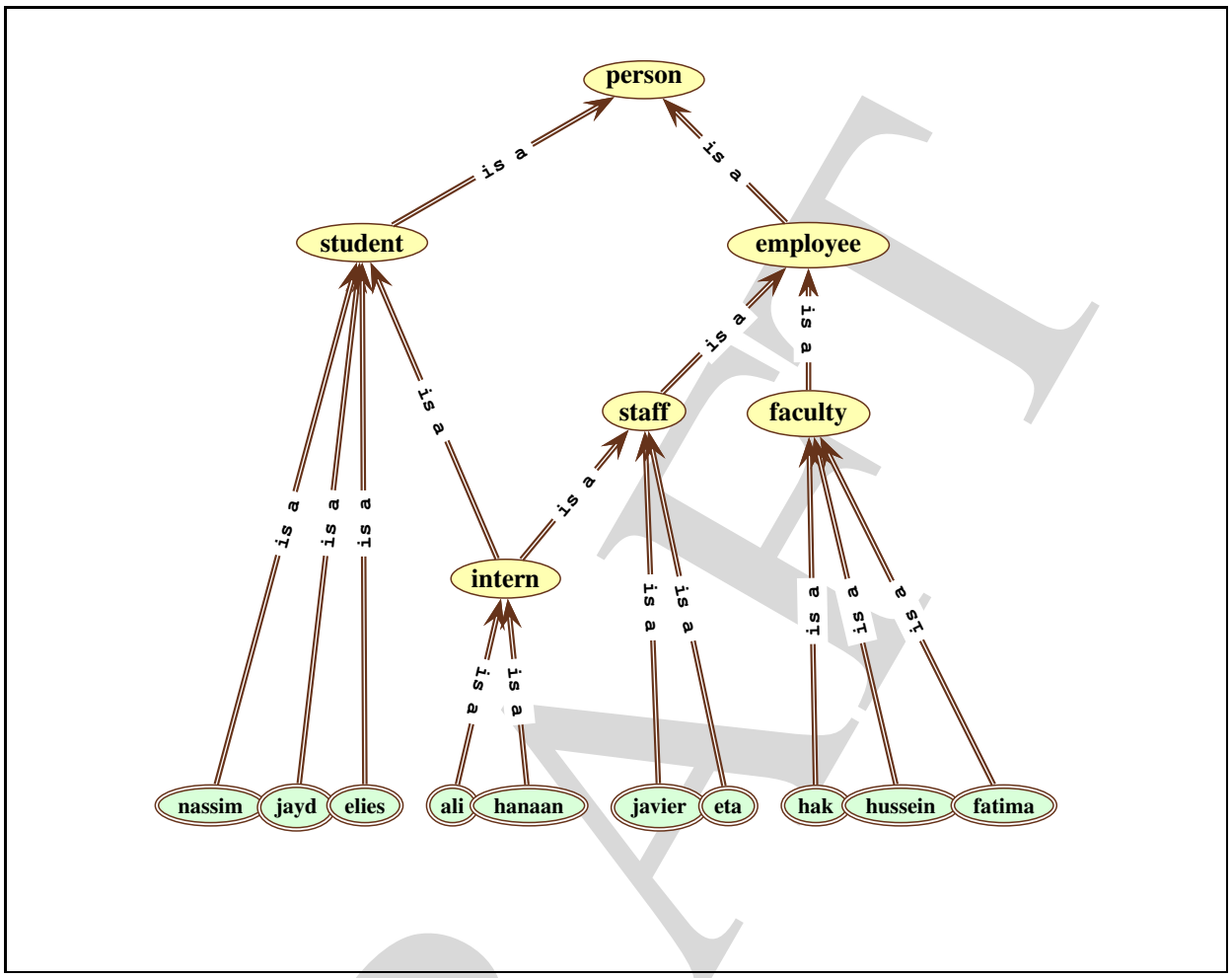


Figure B.6: "School example" concept taxonomy

B.2 Examples of $\mathcal{OSF}$-Term Lattice Operations

In what follows, we give detailed examples illustrating $\mathcal{OSF}$ unification and generalization using the sort taxonomy in Figure B.6, which corresponds to one using meaningful symbols as opposed to those used in Example 4.2 and Example 4.4 given in Chapter 4, and more complex term structure.

B.2.1 The taxonomy

The following could be used as a possible description of a partial school population with the following subconcept and instance declarations — the one pictured in Figure B.6.

- signature of set-denoting concepts:
  - a `student` is a `person`

  - an employee is a person

  - a staff is an employee
  - a faculty is an employee

  - an intern is a student
  - an intern is a staff

- individual-denoting subconcepts:

  - nassim is a student
  - jayd is a student
  - elies is a student

  - ali is an intern
  - hanaan is an intern

  - javier is a staff
  - eta is a staff

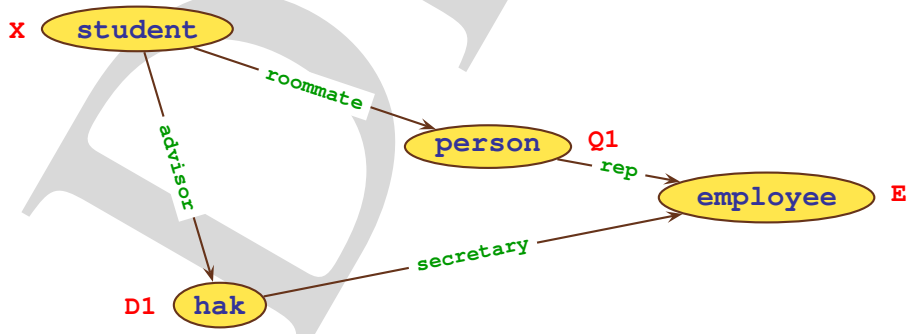
  - hak is a faculty
  - hussein is a faculty
  - fatima is a faculty

B.2.2 The ψ-terms

Example B.1 $\mathcal{OSF}$ **lattice operations** — Consider for example the ψ-term $\mathbf{t}_1$:

$$\begin{aligned}\mathbf{t}_1 \;=\; & \texttt{X:student} \\ & (\,\texttt{roommate} \rightarrow \texttt{person}(\texttt{rep} \rightarrow \texttt{E:employee}) \\ & \,,\,\texttt{advisor} \rightarrow \texttt{hak}(\texttt{secretary} \rightarrow \texttt{E}))\end{aligned}$$

corresponding to the $\mathcal{OSF}$-graph:[4]

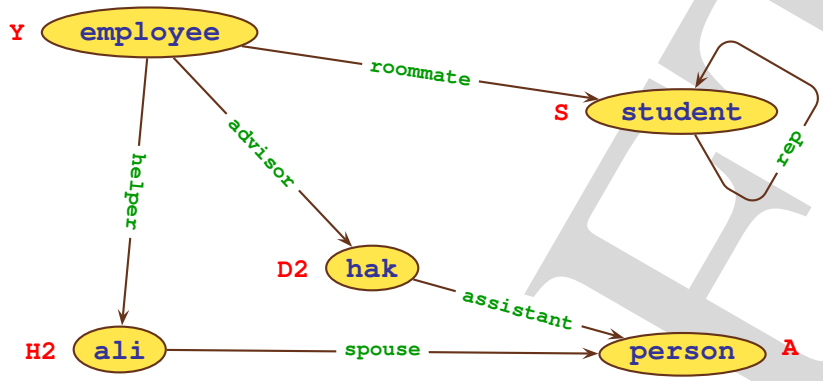


and the ψ-term $\mathbf{t}_2$:

[4] In this and all similar examples, we shall generate new unique tag names that do not occur in the ψ-terms equivalent to $\mathcal{OSF}$-graphs for root nodes of tagless subterms — *e.g.*, Q1 and D1 in this example.

$$\begin{aligned}\mathbf{t}_2 \;=\; & \mathtt{Y{:}employee} \\ & (\,\mathtt{advisor} \rightarrow \mathtt{hak}(\mathtt{assistant} \rightarrow \mathtt{A}) \\ & ,\,\mathtt{roommate} \rightarrow \mathtt{S{:}student}(\mathtt{rep} \rightarrow \mathtt{S}) \\ & ,\,\mathtt{helper} \rightarrow \mathtt{ali}(\mathtt{spouse} \rightarrow \mathtt{A}))\end{aligned}$$

corresponding to the $\mathcal{OSF}$-graph:

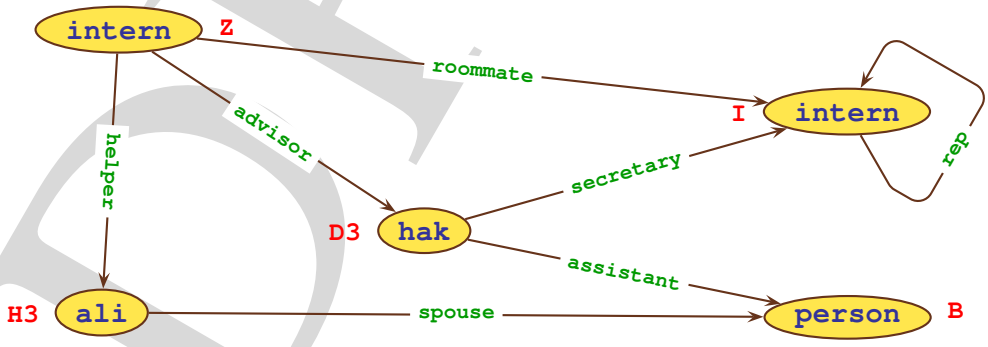


in the context of the sort partial order ("concept taxonomy") shown in Figure B.6.

Endomorphic mappings γ, γ_1, and γ_2, can be computed to exhibit the lattice structure of ψ-terms. Given the two terms $\mathbf{t}_1$ and $\mathbf{t}_2$ shown above, their greatest lower bound is the ψ-term $\underline{\mathbf{t}}$:

$$\begin{aligned}\underline{\mathbf{t}} \;=\; & \mathtt{Z{:}intern} \\ & (\,\mathtt{advisor} \rightarrow \mathtt{hak}(\mathtt{assistant} \rightarrow \mathtt{B}, \\ & \qquad\qquad\qquad\quad \mathtt{secretary} \rightarrow \mathtt{I}) \\ & ,\,\mathtt{helper} \rightarrow \mathtt{ali}(\mathtt{spouse} \rightarrow \mathtt{B}) \\ & ,\,\mathtt{roommate} \rightarrow \mathtt{I{:}intern}(\mathtt{rep} \rightarrow \mathtt{I}))\end{aligned}$$

corresponding to the graph:



given by their $\mathcal{OSF}$ unification realized by the endomorphic mapping γ such that:

$$\gamma(\mathbf{t}_1) = \gamma(\mathbf{t}_2) = \underline{\mathbf{t}}.$$

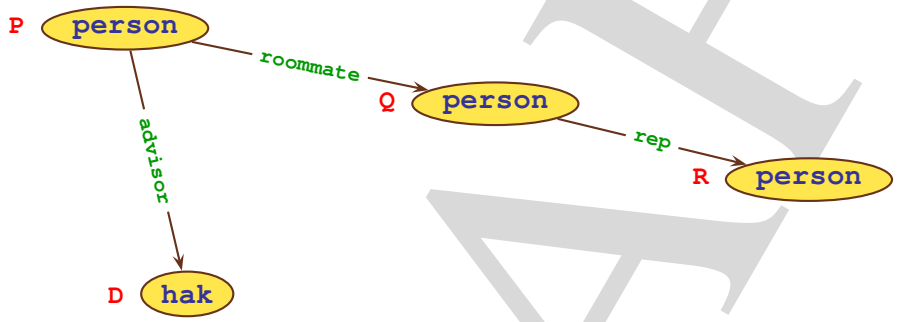
It corresponds to the tag mapping:

$$\begin{array}{llll} \gamma(\mathtt{X}) & = \mathtt{Z}, & \gamma(\mathtt{Y}) & = \mathtt{Z}, \\ \gamma(\mathtt{Q1}) & = \mathtt{I}, & \gamma(\mathtt{S}) & = \mathtt{I}, \\ \gamma(\mathtt{E}) & = \mathtt{I}, & \gamma(\mathtt{D2}) & = \mathtt{D3}, \\ \gamma(\mathtt{D1}) & = \mathtt{D3}, & \gamma(\mathtt{H2}) & = \mathtt{H3}, \\ & & \gamma(\mathtt{A}) & = \mathtt{B}. \end{array} \tag{B.6}$$

Dually, their least upper bound is the ψ-term $\mathbf{t}$:

$$\begin{array}{l} \mathbf{t} \;=\; \mathtt{P}\!:\!\mathtt{person} \\ \qquad (\,\mathtt{roommate} \rightarrow \mathtt{person}(\mathtt{rep} \rightarrow \mathtt{person}) \\ \qquad ,\,\mathtt{advisor} \rightarrow \mathtt{hak}) \end{array}$$

corresponding to the $\mathcal{OSF}$-graph:



given by their $\mathcal{OSF}$ generalization realized by the two endomorphic mappings $\langle\gamma_1, \gamma_2\rangle$ whereby:

$$\begin{array}{l} \gamma_1(\mathbf{t}) = \mathbf{t}_1, \\ \gamma_2(\mathbf{t}) = \mathbf{t}_2. \end{array}$$

They correspond to the tag mappings γ_1 and γ_2 defined as:

$$\begin{array}{llll} \gamma_1(\mathtt{P}) & = \mathtt{X}, & \gamma_2(\mathtt{P}) & = \mathtt{Y}, \\ \gamma_1(\mathtt{Q}) & = \mathtt{Q1}, & \gamma_2(\mathtt{Q}) & = \mathtt{S}, \\ \gamma_1(\mathtt{R}) & = \mathtt{E}, & \gamma_2(\mathtt{R}) & = \mathtt{S}, \\ \gamma_1(\mathtt{D}) & = \mathtt{D1}; & \gamma_2(\mathtt{D}) & = \mathtt{D2}. \end{array}$$

We next give the detailed rule application traces for unification (Section B.2.3, Example B.2) and generalization (Section B.2.4, Example B.3) of ψ-terms.

B.2.3 $\mathcal{OSF}$ unification

Here is a step-by-step trace of constraint normalization computing the unification $\mathbf{glb}(\mathbf{t}_1, \mathbf{t}_2)$ of the ψ-terms $\mathbf{t}_1$ and $\mathbf{t}_2$ defined in Example B.1.

Example B.2 $\mathcal{OSF}$ **unification** — The ψ-term $\underline{\mathbf{t}} = \mathbf{glb}(\mathbf{t}_1, \mathbf{t}_2)$ together with the endomorphism $\gamma : \mathit{Tags}(\mathbf{t}_1) \cup \mathit{Tags}(\mathbf{t}_2) \mapsto \mathit{Tags}(\underline{\mathbf{t}})$ are computed using the $\mathcal{OSF}$ unification rules of Figure 4.6.

We start by dissolving $\mathbf{t}_1$ and $\mathbf{t}_2$:

- $\varphi(\mathtt{t}_1) = \mathtt{X} : \mathtt{student} \;\&\; \mathtt{X}.\mathtt{roommate} \doteq \mathtt{Q1} \;\&\; \mathtt{X}.\mathtt{advisor} \doteq \mathtt{D1} \;\&\; \mathtt{Q1} : \mathtt{person} \;\&\; \mathtt{Q1}.\mathtt{rep} \doteq \mathtt{E} \;\&\; \mathtt{D1} : \mathtt{hak} \;\&\; \mathtt{D1}.\mathtt{secretary} \doteq \mathtt{E} \;\&\; \mathtt{E} : \mathtt{employee}$
- $\varphi(\mathtt{t}_2) = \mathtt{Y} : \mathtt{employee} \;\&\; \mathtt{Y}.\mathtt{roommate} \doteq \mathtt{S} \;\&\; \mathtt{Y}.\mathtt{advisor} \doteq \mathtt{D2} \;\&\; \mathtt{Y}.\mathtt{helper} \doteq \mathtt{H2} \;\&\; \mathtt{S} : \mathtt{student} \;\&\; \mathtt{S}.\mathtt{rep} \doteq \mathtt{S} \;\&\; \mathtt{D2} : \mathtt{hak} \;\&\; \mathtt{D2}.\mathtt{assistant} \doteq \mathtt{A} \;\&\; \mathtt{H2} : \mathtt{ali} \;\&\; \mathtt{H2}.\mathtt{spouse} \doteq \mathtt{A} \;\&\; \mathtt{A} : \mathtt{person}$

Then, we keep applying any applicable $\mathcal{OSF}$ unification rule of Figure 4.6 in any order until none applies starting with the initial constraint $\mathtt{X} \doteq \mathtt{Y} \;\&\; \varphi(\mathtt{t}_1) \;\&\; \varphi(\mathtt{t}_2)$.[5]

1. Start with:

   $\underline{\mathtt{X} \doteq \mathtt{Y}}$

   $\&$
   $\mathtt{X} : \mathtt{student} \;\&\; \mathtt{X}.\mathtt{roommate} \doteq \mathtt{Q1} \;\&\; \mathtt{X}.\mathtt{advisor} \doteq \mathtt{D1} \;\&\; \mathtt{Q1} : \mathtt{person}$
   $\&\; \mathtt{Q1}.\mathtt{rep} \doteq \mathtt{E} \;\&\; \mathtt{D1} : \mathtt{hak} \;\&\; \mathtt{D1}.\mathtt{secretary} \doteq \mathtt{E} \;\&\; \mathtt{E} : \mathtt{employee}$
   $\&$
   $\mathtt{Y} : \mathtt{employee} \;\&\; \mathtt{Y}.\mathtt{roommate} \doteq \mathtt{S} \;\&\; \mathtt{Y}.\mathtt{advisor} \doteq \mathtt{D2} \;\&\; \mathtt{Y}.\mathtt{helper} \doteq \mathtt{H2} \;\&\; \mathtt{S} : \mathtt{student} \;\&\; \mathtt{S}.\mathtt{rep} \doteq \mathtt{S} \;\&\; \mathtt{D2} : \mathtt{hak} \;\&\; \mathtt{D2}.\mathtt{assistant} \doteq \mathtt{A} \;\&\; \mathtt{H2} : \mathtt{ali} \;\&\; \mathtt{H2}.\mathtt{spouse} \doteq \mathtt{A} \;\&\; \mathtt{A} : \mathtt{person}$

2. apply Rule **TAG ELIMINATION** ($\mathtt{X}/\mathtt{Y}$):

   $\underline{\mathtt{X} : \mathtt{student}} \;\&\; \mathtt{X}.\mathtt{roommate} \doteq \mathtt{Q1} \;\&\; \mathtt{X}.\mathtt{advisor} \doteq \mathtt{D1} \;\&\; \mathtt{Q1} : \mathtt{person}$
   $\&\; \mathtt{Q1}.\mathtt{rep} \doteq \mathtt{E} \;\&\; \mathtt{D1} : \mathtt{hak} \;\&\; \mathtt{D1}.\mathtt{secretary} \doteq \mathtt{E} \;\&\; \mathtt{E} : \mathtt{employee}$
   $\&$
   $\underline{\mathtt{X} : \mathtt{employee}} \;\&\; \mathtt{X}.\mathtt{roommate} \doteq \mathtt{S} \;\&\; \mathtt{X}.\mathtt{advisor} \doteq \mathtt{D2} \;\&\; \mathtt{X}.\mathtt{helper} \doteq \mathtt{H2} \;\&\; \mathtt{S} : \mathtt{student} \;\&\; \mathtt{S}.\mathtt{rep} \doteq \mathtt{S} \;\&\; \mathtt{D2} : \mathtt{hak} \;\&\; \mathtt{D2}.\mathtt{assistant} \doteq \mathtt{A} \;\&\; \mathtt{H2} : \mathtt{ali} \;\&\; \mathtt{H2}.\mathtt{spouse} \doteq \mathtt{A} \;\&\; \mathtt{A} : \mathtt{person}$
   $\&$
   $\mathtt{X} \doteq \mathtt{Y}$

3. apply Rule **SORT INTERSECTION** ($\mathtt{X} : \mathtt{student} \wedge \mathtt{employee}$):

   $\mathtt{X} : \mathtt{intern}$
   $\&$
   $\underline{\mathtt{X}.\mathtt{roommate} \doteq \mathtt{Q1}} \;\&\; \mathtt{X}.\mathtt{advisor} \doteq \mathtt{D1} \;\&\; \mathtt{Q1} : \mathtt{person} \;\&\; \mathtt{Q1}.\mathtt{rep} \doteq \mathtt{E} \;\&\; \mathtt{D1} : \mathtt{hak} \;\&\; \mathtt{D1}.\mathtt{secretary} \doteq \mathtt{E} \;\&\; \mathtt{E} : \mathtt{employee}$
   $\&$
   $\underline{\mathtt{X}.\mathtt{roommate} \doteq \mathtt{S}} \;\&\; \mathtt{X}.\mathtt{advisor} \doteq \mathtt{D2} \;\&\; \mathtt{X}.\mathtt{helper} \doteq \mathtt{H2} \;\&\; \mathtt{S} : \mathtt{student} \;\&\; \mathtt{S}.\mathtt{rep} \doteq \mathtt{S} \;\&\; \mathtt{D2} : \mathtt{hak} \;\&\; \mathtt{D2}.\mathtt{assistant} \doteq \mathtt{A} \;\&\; \mathtt{H2} : \mathtt{ali} \;\&\; \mathtt{H2}.\mathtt{spouse} \doteq \mathtt{A} \;\&\; \mathtt{A} : \mathtt{person}$
   $\&$
   $\mathtt{X} \doteq \mathtt{Y}$

4. apply Rule **FEATURE FUNCTIONALITY** ($\mathtt{X}.\mathtt{roommate}$):

[5] We shall underline the parts of a constraint matching a unification rule prior constraint pattern, which rule is then applied next in the application trace.

```
X : intern
&
X . roommate ≐ Q1 & X . advisor ≐ D1 & Q1 : person & Q1 . rep ≐
E & D1 : hak & D1 . secretary ≐ E & E : employee
&
Q1 ≐ S & X . advisor ≐ D2 & X . helper ≐ H2 & S : student & S . rep ≐
S & D2 : hak & D2 . assistant ≐ A & H2 : ali & H2 . spouse ≐ A & A :
person
&
X ≐ Y
```

5. apply Rule TAG ELIMINATION (Q1/S):

```
X : intern
&
X . roommate ≐ Q1 & X . advisor ≐ D1 & Q1 : person & Q1 . rep ≐
E & D1 : hak & D1 . secretary ≐ E & E : employee
&
X . advisor ≐ D2 & X . helper ≐ H2 & Q1 : student & Q1 . rep ≐
Q1 & D2 : hak & D2 . assistant ≐ A & H2 : ali & H2 . spouse ≐ A & A :
person
&
X ≐ Y & Q1 ≐ S
```

6. apply Rule FEATURE FUNCTIONALITY (X.advisor):

```
X : intern
&
X . roommate ≐ Q1 & X . advisor ≐ D1 & Q1 : person & Q1 . rep ≐
E & D1 : hak & D1 . secretary ≐ E & E : employee
&
D1 ≐ D2 & X . helper ≐ H2 & Q1 : student & Q1 . rep ≐ Q1 & D2 :
hak & D2 . assistant ≐ A & H2 : ali & H2 . spouse ≐ A & A : person
&
X ≐ Y & Q1 ≐ S
```

7. apply Rule TAG ELIMINATION (D1/D2) and use logical simplification $\phi \;\&\; \phi \rightarrow \phi$ with $\phi =$ `D1 : hak`:

```
X : intern
&
X . roommate ≐ Q1 & X . advisor ≐ D1 & D1 : hak & Q1 : person & Q1 . rep ≐
E & D1 . secretary ≐ E & E : employee
&
X . helper ≐ H2 & Q1 : student & Q1 . rep ≐ Q1 & D1 . assistant ≐
A & H2 : ali & H2 . spouse ≐ A & A : person
&
X ≐ Y & Q1 ≐ S & D1 ≐ D2
```

8. apply Rule SORT INTERSECTION (Q1 : person ∧ student):

   X : intern
   &
   & X . roommate ≐ Q1 & X . advisor ≐ D1 & D1 : hak & Q1 : student
   & Q1 . rep ≐ E & D1 . secretary ≐ E & E : employee
   &
   X . helper ≐ H2 & Q1 . rep ≐ Q1 & D1 . assistant ≐ A & H2 : ali
   & H2 . spouse ≐ A & A : person
   &
   X ≐ Y & Q1 ≐ S & D1 ≐ D2

9. apply Rule FEATURE FUNCTIONALITY (Q1.rep):

   X : intern
   &
   X . roommate ≐ Q1 & X . advisor ≐ D1 & D1 : hak & Q1 : student & Q1 . rep ≐ E & D1 . secretary ≐ E & E : employee
   &
   X . helper ≐ H2 & E ≐ Q1 & D1 . assistant ≐ A & H2 : ali & H2 . spouse ≐ A & A : person
   &
   X ≐ Y & Q1 ≐ S & D1 ≐ D2

10. apply Rule TAG ELIMINATION (E/Q1):

    X : intern
    &
    X . roommate ≐ E & X . advisor ≐ D1 & D1 : hak & E : student & E . rep ≐ E & D1 . secretary ≐ E & E : employee
    &
    X . helper ≐ H2 & D1 . assistant ≐ A & H2 : ali & H2 . spouse ≐ A & A : person
    &
    X ≐ Y & Q1 ≐ S & D1 ≐ D2 & E ≐ Q1

11. apply Rule SORT INTERSECTION (E : student ∧ employee):

    X : intern
    &
    X . roommate ≐ E & X . advisor ≐ D1 & D1 : hak & E : intern & E . rep ≐ E & D1 . secretary ≐ E
    &
    X . helper ≐ H2 & H2 . spouse ≐ A & A : person
    &
    X ≐ Y & Q1 ≐ S & D1 ≐ D2 & E ≐ Q1

12. normal form (after commutative reordering of conjuncts):

    X : intern

```
&
X . roommate ≐ E & E : intern & E . rep ≐ E
&
X . advisor ≐ D1 & D1 : hak
&
D1 . secretary ≐ E
&
D1 . assistant ≐ A & A : person
&
X . helper ≐ H2 & H2 . spouse ≐ A & H2 : ali
&
X ≐ Y & Q1 ≐ S & D1 ≐ D2 & E ≐ Q1.
```

A scoped endomorphism $\gamma : \mathit{Tags}(\mathbf{t}_1) \cup \mathit{Tags}(\mathbf{t}_2) \rightarrow \mathit{Tags}(\underline{\mathbf{t}})$ is obtained from this normal form as follows. Since $\mathit{Tags}(\mathbf{t}_1) = \{\mathtt{X}, \mathtt{D1}, \mathtt{Q1}, \mathtt{E}\}$ and $\mathit{Tags}(\mathbf{t}_2) = \{\mathtt{Y}, \mathtt{H2}, \mathtt{D2}, \mathtt{A}, \mathtt{S}\}$, the set $\mathit{Tags}(\underline{\mathbf{t}}) = \{\mathtt{X}, \mathtt{D1}, \mathtt{Q1}, \mathtt{E}, \mathtt{Y}, \mathtt{H2}, \mathtt{D2}, \mathtt{A}, \mathtt{S}\}$ is partitioned into the following tag coreference classes corresponding to least tag equivalence (*i.e.*, reflexive, symmetric, and transitive) closure of the final tag renaming constraints remaining in the normal form; *viz.*, $\mathtt{X} \doteq \mathtt{Y}$ & $\mathtt{Q1} \doteq \mathtt{S}$ & $\mathtt{D1} \doteq \mathtt{D2}$ & $\mathtt{E} \doteq \mathtt{Q1}$:

$$\{\mathtt{X}, \mathtt{Y}\}\{\mathtt{Q1}, \mathtt{E}, \mathtt{S}\}\{\mathtt{D1}, \mathtt{D2}\}\{\mathtt{H2}\}\{\mathtt{A}\}$$

which we can rename with a new unique tag name per tag-coreference class as a new tag representative to ensure a well-scoped tag endomorphism as follows:

$$\begin{aligned}
\mathtt{Z} &\stackrel{\text{def}}{=} \{\mathtt{X}, \mathtt{Y}\},\\
\mathtt{I} &\stackrel{\text{def}}{=} \{\mathtt{Q1}, \mathtt{S}, \mathtt{E}\},\\
\mathtt{D3} &\stackrel{\text{def}}{=} \{\mathtt{D1}, \mathtt{D2}\},\\
\mathtt{H3} &\stackrel{\text{def}}{=} \{\mathtt{H2}\},\\
\mathtt{B} &\stackrel{\text{def}}{=} \{\mathtt{A}\}.
\end{aligned}$$

The renamed normal form is that of $\underline{\mathbf{t}} = \mathbf{glb}(\mathbf{t}_1, \mathbf{t}_2)$ shown in Figure B.7, with the corresponding endomorphic mapping γ in Equation (B.6) such that $\underline{\mathbf{t}} = \gamma(\mathbf{t}_1) = \gamma(\mathbf{t}_2)$.

B.2.4 $\mathcal{OSF}$ generalization

Here is a step-by-step trace of constraint normalization computing the generalization $\mathbf{lub}(\mathbf{t}_1, \mathbf{t}_2)$ of the ψ-terms $\mathbf{t}_1$ and $\mathbf{t}_2$ defined in Example B.1. It computes their **lub**, along with the corresponding endomorphic mappings $\gamma_i : \mathit{Tags}(\mathbf{t}) \mapsto \mathit{Tags}(\mathbf{t}_i)$, for $i = 1, 2$, using the $\mathcal{OSF}$ generalization rules of Figure 4.8 as follows.

Example B.3 $\mathcal{OSF}$ **generalization** — Let start with the following $\mathcal{OSF}$ generalization judgment constraint to resolve:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}\mathbf{t}_1\\ \mathbf{t}_2\end{pmatrix} \mathbf{t}_1 \vee \mathbf{t}_2 \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix}$$

in which the ψ-term $\mathbf{t}_1 \vee \mathbf{t}_2$ and the tag maps γ_1 and γ_2 are to be determined by normalizing this judgment according to the axiom and rule of Figure 4.8.

Since the ψ-terms $\mathbf{t}_1$ and $\mathbf{t}_2$ defined in Example B.1 are such that $\textsc{root}(\mathbf{t}_1) = \mathtt{X}$ and $\textsc{root}(\mathbf{t}_2) = \mathtt{Y}$, we note that:

- since $\mathtt{X} \neq \mathtt{Y}$,
- because $\mathtt{student} \vee \mathtt{employee} = \mathtt{person}$,
- if ψ_1 and ψ_2 are two subterms to be determined, and
- introducing a new tag name $\mathtt{P}$,

it becomes evident that initial Judgment (B.7) corresponds to the denominator of Rule **Unequal Tags**:

$$\begin{pmatrix}\emptyset\\ \emptyset\end{pmatrix} \vdash \begin{pmatrix}\mathbf{t}_1\\ \mathbf{t}_2\end{pmatrix} \mathtt{P} : \mathtt{person}(\mathtt{roommate} \to \psi_1, \mathtt{advisor} \to \psi_2) \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \tag{B.7}$$

and it becomes the following sequence of two judgments indicated by the rule's numerator defining the two endomorphic mappings $\gamma_1^0(\mathtt{P}) \stackrel{\text{def}}{=} \mathtt{X}$ and $\gamma_2^0(\mathtt{P}) \stackrel{\text{def}}{=} \mathtt{Y}$:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{P}\}\\ \{\mathtt{Y}/\mathtt{P}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{Q1} : \mathtt{person}(\mathtt{rep} \to \mathtt{E} : \mathtt{employee})\\ \mathtt{S} : \mathtt{student}(\mathtt{rep} \to \mathtt{S})\end{pmatrix} \uparrow \begin{pmatrix}\{\mathtt{X}/\mathtt{P}\}\\ \{\mathtt{Y}/\mathtt{P}\}\end{pmatrix} \psi_1 \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \tag{B.8}$$

$$\begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \vdash \begin{pmatrix}\mathtt{D1} : \mathtt{hak}(\mathtt{secretary} \to \mathtt{E})\\ \mathtt{D2} : \mathtt{hak}(\mathtt{assistant} \to \mathtt{A})\end{pmatrix} \uparrow \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \psi_2 \begin{pmatrix}\gamma_1\\ \gamma_2\end{pmatrix} \tag{B.9}$$

and then evaluating the unapplication in Judgment (B.8), it becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{P}\}\\ \{\mathtt{Y}/\mathtt{P}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{Q1} : \mathtt{person}(\mathtt{rep} \to \mathtt{E} : \mathtt{employee})\\ \mathtt{S} : \mathtt{student}(\mathtt{rep} \to \mathtt{S})\end{pmatrix} \psi_1 \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \tag{B.10}$$

then, by Rule **Unequal Tags**:

- because $\mathtt{Q1} \neq \mathtt{S}$,
- since $\mathtt{person} \vee \mathtt{student} = \mathtt{person}$,
- defining $\psi_1 \stackrel{\text{def}}{=} \mathtt{Q} : \mathtt{person}(\mathtt{rep} \to \psi_1')$,
- where $\mathtt{Q}$ is a new tag name,
- ψ_1' is a subterm to determine,

Judgment (B.10) becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{P}, \mathtt{Q1}/\mathtt{Q}\}\\ \{\mathtt{Y}/\mathtt{P}, \mathtt{S}/\mathtt{Q}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{Q1} : \mathtt{person}(\mathtt{rep} \to \mathtt{E} : \mathtt{employee})\\ \mathtt{S} : \mathtt{student}(\mathtt{rep} \to \mathtt{S})\end{pmatrix} \mathtt{Q} : \mathtt{person}(\mathtt{rep} \to \psi_1') \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \tag{B.11}$$

then, Judgment (B.11) corresponds again to the pattern in the denominator of Rule **Unequal Tags** and since the two terms to generalize in this judgment have only one common feature (*viz.*, $\mathtt{rep}$), Judgment (B.11) begets the following single-subterm judgment:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{P}, \mathtt{Q1}/\mathtt{Q}\}\\ \{\mathtt{Y}/\mathtt{P}, \mathtt{S}/\mathtt{Q}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{E} : \mathtt{employee}\\ \mathtt{S} : \mathtt{student}(\mathtt{rep} \to \mathtt{S})\end{pmatrix} \uparrow \begin{pmatrix}\{\mathtt{X}/\mathtt{P}, \mathtt{Q1}/\mathtt{Q}\}\\ \{\mathtt{Y}/\mathtt{P}, \mathtt{S}/\mathtt{Q}\}\end{pmatrix} \psi_1' \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \tag{B.12}$$

which after evaluating the unapplication becomes:

$$\begin{pmatrix}\{\mathtt{X}/\mathtt{P}, \mathtt{Q1}/\mathtt{Q}\}\\ \{\mathtt{Y}/\mathtt{P}, \mathtt{S}/\mathtt{Q}\}\end{pmatrix} \vdash \begin{pmatrix}\mathtt{E} : \mathtt{employee}\\ \mathtt{S} : \mathtt{student}(\mathtt{rep} \to \mathtt{S})\end{pmatrix} \psi_1' \begin{pmatrix}\gamma_1^1\\ \gamma_2^1\end{pmatrix} \tag{B.13}$$

and:

- since the two terms in Judgment (B.13) have no common feature,
- because $\texttt{employee} \vee \texttt{student} = \texttt{person}$,
- using a new tag name $\texttt{R}$,
- defining $\psi_1' \stackrel{\text{def}}{=} \texttt{R} : \texttt{person}$,

this becomes:

$$\begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}\} \end{pmatrix} \vdash \begin{pmatrix} \texttt{E} : \texttt{employee} \\ \texttt{S} : \texttt{student}(\texttt{rep} \to \texttt{S}) \end{pmatrix} \texttt{R} : \texttt{person} \begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}\} \end{pmatrix} \tag{B.14}$$

and terminates the proof of Judgment (B.8) with $\gamma_1^1 = \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}\}$ and $\gamma_2^1 = \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}\}$.

We may now proceed to proving remaining Judgment (B.9) which becomes:

$$\begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}\} \end{pmatrix} \vdash \begin{pmatrix} \texttt{D1} : \texttt{hak}(\texttt{secretary} \to \texttt{E}) \\ \texttt{D2} : \texttt{hak}(\texttt{assistant} \to \texttt{A}) \end{pmatrix} \uparrow \begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}\} \end{pmatrix} \psi_2 \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} \tag{B.15}$$

and after evaluating the unapplication, this becomes:

$$\begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}\} \end{pmatrix} \vdash \begin{pmatrix} \texttt{D1} : \texttt{hak}(\texttt{secretary} \to \texttt{E}) \\ \texttt{D2} : \texttt{hak}(\texttt{assistant} \to \texttt{A}) \end{pmatrix} \psi_2 \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} \tag{B.16}$$

and:

- because $\texttt{hak} \vee \texttt{hak} = \texttt{hak}$,
- defining $\psi_2 \stackrel{\text{def}}{=} \texttt{D} : \texttt{hak}$,
- where $\texttt{D}$ is a new tag name,

this terminates the complete proof of Judgment (B.7) with the following final judgment:

$$\begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}, \texttt{D1}/\texttt{D}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}, \texttt{D2}/\texttt{D}\} \end{pmatrix} \vdash \begin{pmatrix} \texttt{D1} : \texttt{hak}(\texttt{secretary} \to \texttt{E}) \\ \texttt{D2} : \texttt{hak}(\texttt{assistant} \to \texttt{A}) \end{pmatrix} \texttt{D} : \texttt{hak} \begin{pmatrix} \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}, \texttt{D1}/\texttt{D}\} \\ \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}, \texttt{D2}/\texttt{D}\} \end{pmatrix} \tag{B.17}$$

i.e., with the final tag endomorphisms $\gamma_1 = \{\texttt{X}/\texttt{P}, \texttt{Q1}/\texttt{Q}, \texttt{E}/\texttt{R}, \texttt{D1}/\texttt{D}\}$ and $\gamma_2 = \{\texttt{Y}/\texttt{P}, \texttt{S}/\texttt{Q}, \texttt{S}/\texttt{R}, \texttt{D2}/\texttt{D}\}$ and least upper bound:

$$\texttt{t}_1 \vee \texttt{t}_2 = \texttt{P} : \texttt{person}(\texttt{roommate} \to \texttt{Q} : \texttt{person}(\texttt{rep} \to \texttt{R} : \texttt{person}), \texttt{advisor} \to \texttt{D} : \texttt{hak}).$$

The result of all the above together with the result of the trace of the $\mathcal{OSF}$ unification operation on the same terms $\texttt{t}_1$ and $\texttt{t}_2$ in Section B.2.3 can be summarized as illustrated by the lattice diagram shown in Figure B.7.

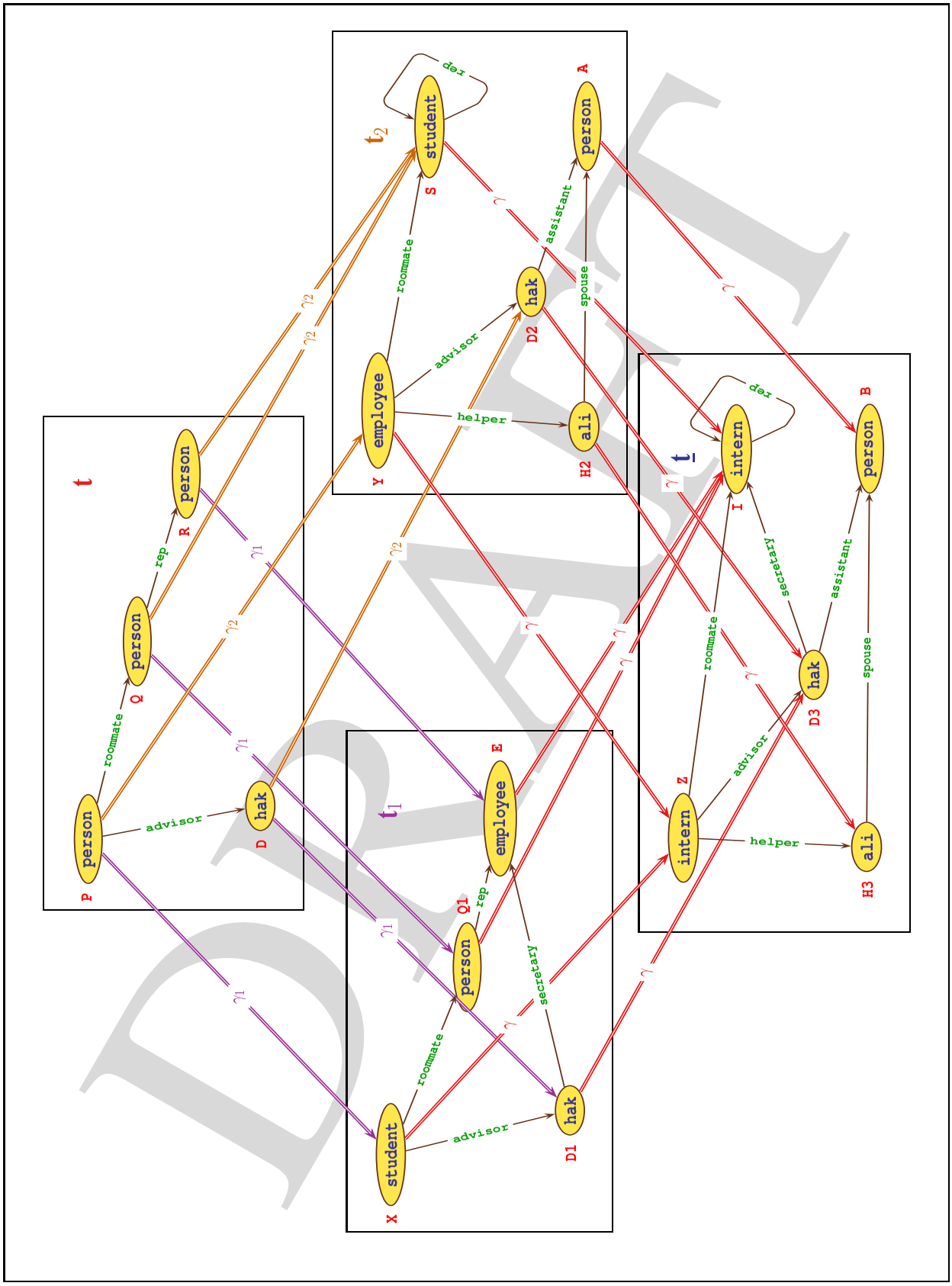


Figure B.7: Example of $\mathcal{OSF}$ subsumption lattice operations

Appendix C

Version of April 10, 2020

Probabilistic Background

In this appendix chapter, we give a quick tutorial on Aji and McEliece's *Generalized Distributive Law* ($\mathcal{GDL}$) [28] aiming at simplifying the task of implementing a generic $\mathcal{GDL}$ library (such as [146], [145]) for use in constraint-based probabilistic reasoning [70]. The $\mathcal{GDL}$'s essence is that it factors multiplications out of additions in algebraic expressions using commutative distributive sum/product operations. Doing so, it optimizes this class of computation using a Dynamic-Programming "memo-ing" technique [44].[1] Thus, providing a $\mathcal{GDL}$ library toolset makes sense for supporting practical, generic, efficient, and versatile tools for analysis, inference, and learning with Bayesian probabilities, in a wide family of algebraic structures in many diverse contexts.

C.1 Bayesian Nets

DEFINITION C.1 (INDEPENDENT EVENTS) *Two events A and B are said to be independent whenever $p(A,B) = p(A)\times p(B)$.*

DEFINITION C.2 (CONDITIONAL PROBABILITY) *If A and B are events, we write "$p(A|B)$" to denote the conditional probability of A knowing that B has occurred. It is defined as:*

$$p(A|B) \stackrel{def}{=} \frac{p(A,B)}{p(B)}$$

whenever $p(B) \neq 0$.

This definition makes intuitive sense since it the probability that both A and B occur, tempered by the probability that B occurs at all (whether A does or not). Then, clearly, two events A and B are independent *iff* $p(A|B) = p(A)$, as well as *iff* $p(B|A) = p(B)$.

Bayes Law is a universal property that constrains the mutual conditional probabilities of two events A and B always to obey the following symmetrical equation:

$$p(A|B)\times p(B) = P(B|A)\times p(A) \tag{C.1}$$

[1] I recommend https://cs.uwaterloo.ca/~gweddell/cs234/lect-Dynamic.pdf for a clear, short, and simple lecture on this concept and why it often works very well in the situations where it applies — as is the case for the $\mathcal{GDL}$.

since in this case, both sides are equal to $p(A, B)$ by commutativity of conjunction. In words, the probability of A knowing B times the probability of B must always be equal to the probability of B knowing A times the probability of A. This property is quite useful since, as illustrated below, it means we can use what we have observed to narrow possibilities to derive the probability of what we have not.

A *Bayesian Net* is a graph-theoretic encoding of observed causality among events [132, 70]. As as example, let consider the four following event: (1) cloudy weather; (2) rainy weather; (3) the garden sprinkler is on; and, (4) the grass is wet. Our experience has gathered data that has made us observe four possible causality relations among these four events. Namely, that: (1) the sprinkler is on when the weather is cloudy; (2) weather is rainy when it also is cloudy; (3) the grass is wet when the sprinkler is on; and, (4) the grass is wet when the weather is rainy. These observed relationships are what make up a Bayesian Net as a causal graph, as illustrated in Figure C.1 for the given example.

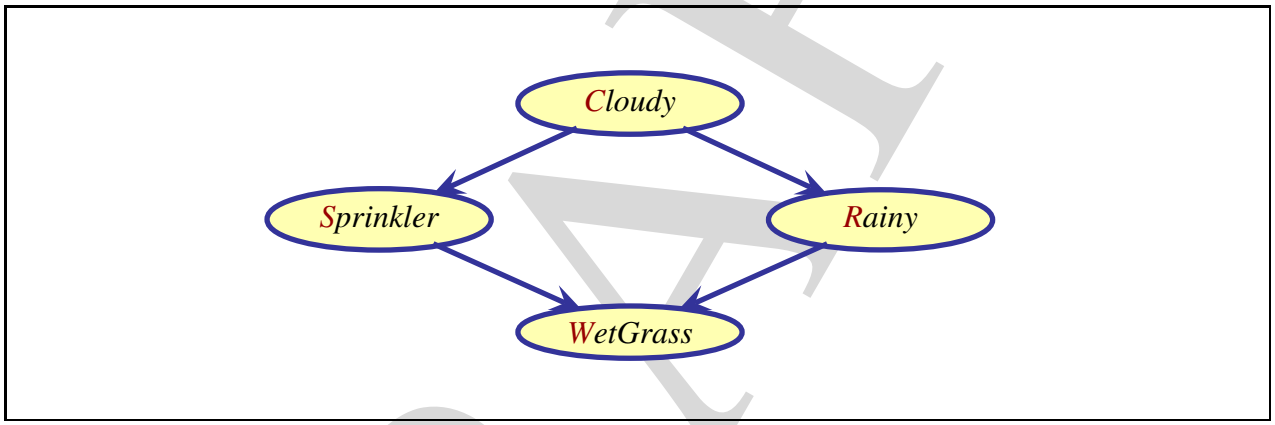


Figure C.1: Example of Bayesian net

Such a causal graph expresses implicit information concerning the observable conditional independence of the events represented as nodes. This graph tells us that (1) *R* and *S* are independent given *C*, and (2) *W* and *C* are independent given *R* and *S*. Figure C.2 shows an example of causal conditional probability tables for the Bayesian Net of Figure C.1.

C.2 Inference in Bayesian Nets

Bayes rule allows computing a joint probability as a product of dependent probabilities. This is made usable for more than just two events, for any set of $n \geq 2$ events $\{E_1, \ldots, E_n\}$ since for any permutation π of the set $\{1, \ldots, n\}$, Equation (C.1) implies:

$$p(E_{\pi(1)}, \ldots, E_{\pi(n)}) = p(E_{\pi(1)}) \times p(E_{\pi(2)}|E_{\pi(1)}) \times \ldots \times p(E_{\pi(n)}|E_{\pi(1)}, \ldots, E_{\pi(n-1)}). \quad \text{(C.2)}$$

For example, using C for *Cloudy*, S for *Sprinkler*, R for *Rainy*, and W for *WetGrass*, this allows stating that:

$$p(C, S, R, W) = p(C) \times p(S|C) \times p(R|C, S) \times p(W|R, C, S).$$

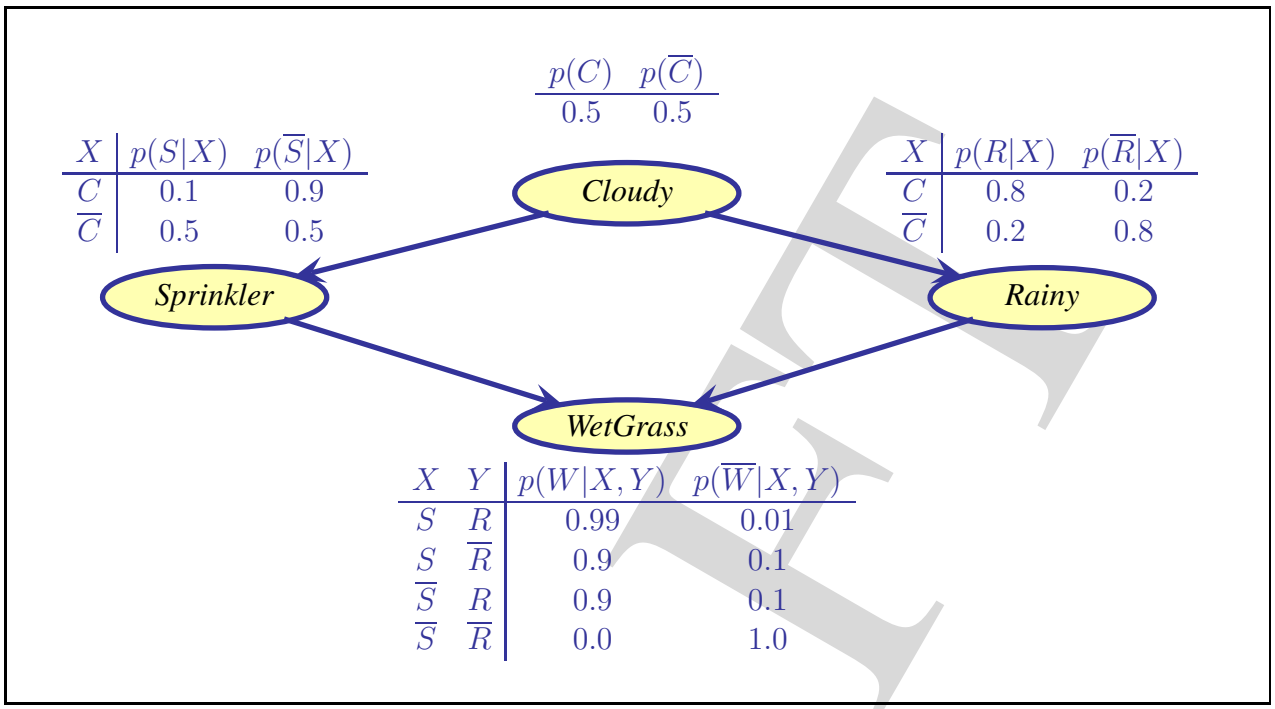


Figure C.2: Example of a Bayesian net's causal conditional probabilities

Using the independence inferred from the net topology, this simplifies to:

$$p(C,S,R,W) = p(C)\times p(S|C)\times p(R|C)\times p(W|R,S).$$

Probability *"marginalization"* is readjusting the probability of still unknown events taking into account known events. For example, the probability that the sprinkler is on knowing that the grass is wet is given by the ratio obtained from Bayes Law on the two concerned events; that is:

$$p(S|W) = \frac{p(S,W)}{p(W)} = \frac{\sum_{c,r} p(C=c,S,R=r,W)}{\sum_{c,s,r} p(C=c,S=s,R=r,W)} = \frac{0.2781}{0.6471} = 0.4298,$$

and the probability that it is rainy knowing that the grass is wet is:

$$p(R|W) = \frac{p(R,W)}{p(W)} = \frac{\sum_{c,s} p(C=c,S=s,R,W)}{\sum_{c,s,r} p(C=c,S=s,R=r,W)} = \frac{0.4581}{0.6471} = 0.7079.$$

Markov Blanket

As can be seen from our "wet grass" example, intuition may be easily fooled trying to determine *what* is independent of *what* given *what*—even in such a trivial causal graph! Fortunately, the wealth of formal research on the subject has made it possible to reduce this analysis to a very simple criterion. Indeed, conditional independence can be easily determined from the connectivity of a causal graph by computing each node's so-called *Markov blanket*. The Markov blanket of a node is defined as the set of nodes comprising the node's parents, its children, and its children's

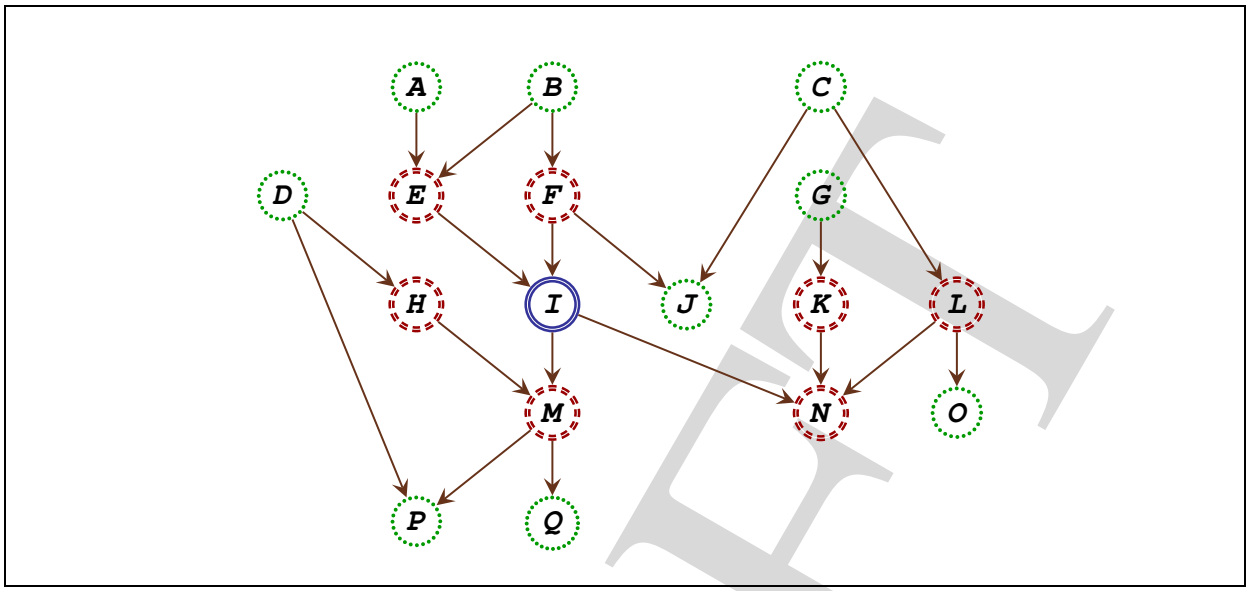


Figure C.3: Example of a Markov blanket

other parents.[2] That is, given a node X, its Markov blanket is defined as the set:

$$\partial_X \stackrel{\text{def}}{=} \textsc{Parents}(X) \cup \textsc{Children}(X) \cup \textsc{Parents}(\textsc{Children}(X)) \setminus \{X\}.$$

For example, referring to the causal graph of the example in Figure C.3, we obtain:

$$\textsc{Parents}(\boldsymbol{I}) = \{\boldsymbol{E}, \boldsymbol{F}\},$$

$$\textsc{Children}(\boldsymbol{I}) = \{\boldsymbol{M}, \boldsymbol{N}\},$$

and:

$$\textsc{Parents}(\textsc{Children}(\boldsymbol{I})) \setminus \{\boldsymbol{I}\} = \{\boldsymbol{H}, \boldsymbol{K}, \boldsymbol{L}\}.$$

Hence, the Markov blanket of the node $\boldsymbol{I}$ is:

$$\partial_{\boldsymbol{I}} = \{\boldsymbol{E}, \boldsymbol{F}, \boldsymbol{H}, \boldsymbol{K}, \boldsymbol{L}, \boldsymbol{M}, \boldsymbol{N}\}.$$

The Markov blanket splits each node X in the set $\mathcal{N}$ of nodes of a Bayesian network's causal graph partition $\mathcal{N}$ into three mutually exclusive components; namely, $\mathcal{N} = \partial_X \uplus \{X\} \uplus \partial_X^c$. The key result is that any node is independent of nodes outside its Markov blanket: $X \perp \partial_X^c \mid \partial_X$ [132, 93]. Using Bayes's rule, this allows the following simplification by conditional independence: $p(X \mid \partial_X, \partial_X^c) = p(X \mid \partial_X)$.

[2] http://en.wikipedia.org/wiki/Markov_blanket

Belief revision One of the most powerful capabilities offered by a Bayesian network is that it can adapt its knowledge according to accumulated evidence. This is known as *"explaining away"* since it is a form if *plausible reasoning* such that whenever several events are plausible causes of another one, say X, posterior evidence changes the likelihood of explanations for X. To see that with our example, let us suppose that it is observed that (W) the grass is wet and that (R) it is raining. Then, this indicates that the posterior likelihood that (S) the sprinkler is on goes down as follows:

$$p(S \mid W, R) = \frac{p(S, W, R)}{p(W, R)} = \frac{\sum_c p(C = c, S, W, R)}{\sum_{c,s} p(C = c, S = s, W, R)} = 0.1945.$$

Causal learning Yet another benefit of Bayesian networks is that they can be learned from data, thus circumventing the "expert belief assessment" problem [45]. Indeed, recent research in Data Mining has made great progress for learning Bayesian network (parameters *and* structure) from data [125].[3,4]

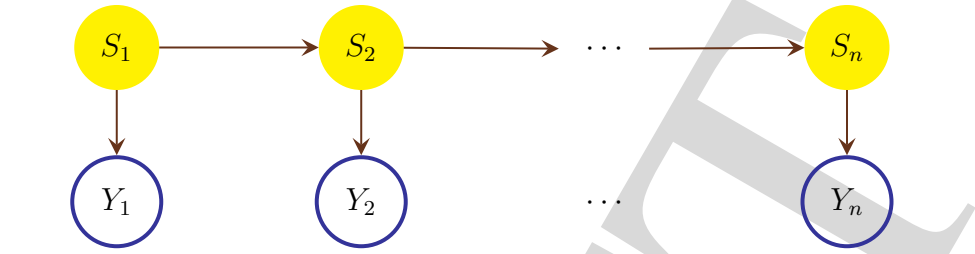
As illustrated in [28], several important algorithms that were independently conceived, such as Judea's Pearl Belief Propagation [132], the Expectation-Maximization algorithm [122], Viterbi's algorithm [170], and many others, can all be cast as specific instances of the $\mathcal{GDL}$ algorithm.

The Expectation-Maximization algorithm was introduced in 1977 in a paper by Dempster, Laird, and Rubin [72]. This algorithm is used to find the maximum-likelihood parameters of a statistical model when the equation of the model itself cannot be solved directly. This algorithm is often used in computational biology applications since it allows drawing conclusions with incomplete sets of data. The Expectation-Maximization algorithm is indeed quite useful in domains where there is no guarantee that the data collection will produce complete sets of data. The algorithm is quite complex but it is possible to expose a commutative semiring underpinning the operations performed by the algorithm. Therefore, it becomes possible to use the $\mathcal{GDL}$ to solve this kind of problem.

Viterbi's algorithm is a Dynamic-Programming algorithm for finding the most likely sequence of hidden states in a Hidden Markov Model. It is broadly used for decoding convolutional codes—such as, for example, using Viterbi's Algorithm in order to decode bit-stream [170]. These algorithms are well-known in probabilistic deduction. They are widely used in causal learning [125], Sequential Data Analysis [140, 181, 115, 129], belief revision [108, 132], probabilistic-logic programming [69], and many more application areas [136], *etc.*, ...

Dynamic Bayesian Networks Bayesian networks also appear in particular specific instances (*e.g.*, Hidden Markov Models, Linear Dynamic Systems) that are very successful for pattern recognition of sequential data (*e.g.*, speech recognition [181], time series data [115]). Figure C.4 shows examples Hidden Markov Models ($\mathcal{HMM}$), a particular instance of a dynamic Bayesian network where the S_i's are time-indexed "hidden" (*i.e.*, unobservable) states of a Markov process, and each Y_i is an observable random function of the corresponding hidden state S_i.

[3] http://www.cs.cmu.edu/~awm/10701/slides/Param_Struct_Learning05v1.pdf
[4] http://www.autonlab.org/tutorials/



Examples	Mobile robots	Speech recognition	Biological sequencing
S_n	location	phonemes	protein structure
Y_n	sensor input	acoustic signal	amino acids

Figure C.4: Example of $\mathcal{HMM}$'s hidden states and their observations

Hidden Markov Models A Hidden Markov Model ($\mathcal{HMM}$) is a particular instance of a Bayesian Net where S_i is a "hidden" (*i.e.*, unobservable) state of a Markov process [93]; and, Y_i is an observable random function of S_i.

In addition, $\mathcal{HMM}$s can be "trained" on data in the manner of neural networks to enable forecasting. They have been used extensively and applied with great success in diverse fields such as signal decoding [118], speech recognition [181], geological exploration [136], stock trading [129], and genome analysis.[5]

C.3 The Generalized Distributive Law

Inference and learning using Bayesian Nets has proved effective as several notable graph algorithms such as Judea Pearl's belief propagation algorithm [132], Expectation maximization algorithm [140], Baum-Welch "forward-backward" algorithm [140], Viterbi algorithm [170], Discrete Kalman filtering [28], to cite a few, are all instances of the $\mathcal{GDL}$ algorithm in specific domains of objects comprising the domain of a commutative semiring structure with specific addition and multiplication operations, and zero and unity — see, *e.g.*, [28] for more details. All of these (and many other such) algorithms being instances a single generic algorithm it makes sens to make this a tool for deriving other similar applications of the $\mathcal{GDL}$. The $\mathcal{GDL}$ is an algebraic algorithm (a method) that computes sums of products:

$$\sum_{x_i \in S_i}^{1 \le i \le m} \prod_{y_j \in T_j}^{1 \le j \le n} \varphi(x_1, \ldots, x_m, y_1, \ldots, y_n)$$

where $\sum$ and $\prod$ correspond to additive and multiplicative laws of a commutative semiring.

[5] http://genomics10.bu.edu/bioinformatics/kasif/bayes-net.html

Commutative Semiring Algebraic structure $\langle \boldsymbol{K}, +, 0, \times, 1\rangle$ s.t. $+ : K \times K \to K$, $0 \in K$, $\times : K \times K \to K$, $1 \in K$, where $\langle \boldsymbol{K}, +, 0\rangle$ and $\langle \boldsymbol{K}, \times, 1\rangle$ are commutative monoids and $\times$ distributes over $+$. Figure C.5 shows examples of commutative semirings.

	$\boldsymbol{K}$	$+$	0	$\times$	1
(1)	$\boldsymbol{R}$	$+$	0	$\times$	1
(2)	$\boldsymbol{R}[x]$	$+$	0	$\times$	1
(3)	$\boldsymbol{R}[x, y, \ldots]$	$+$	0	$\times$	1
(4)	$[0, +\infty)$	$+$	0	$\times$	1
(5)	$(0, +\infty]$	min	$+\infty$	$\times$	1
(6)	$[0, +\infty)$	max	0	$\times$	1

	$\boldsymbol{K}$	$+$	0	$\times$	1
(7)	$(-\infty, +\infty]$	min	$+\infty$	$+$	0
(8)	$[-\infty, +\infty)$	max	$-\infty$	$+$	0
(9)	$\{\mathfrak{false}, \mathfrak{true}\}$	$\mathfrak{or}$	$\mathfrak{false}$	$\mathfrak{and}$	$\mathfrak{true}$
(10)	$\mathbf{2}^S$	$\cup$	$\emptyset$	$\cap$	S
(11)	$\boldsymbol{L}$	$\wedge$	$\bot$	$\vee$	$\top$
(12)	$\boldsymbol{L}$	$\vee$	$\top$	$\wedge$	$\bot$

$\boldsymbol{R}$: arbitrary commutative ring; $\boldsymbol{R}[x]$: polynomials with variable x with coefficients in $\boldsymbol{R}$; $\boldsymbol{R}[x, y, \ldots]$: multinomials with variables x, y, ..., with coefficients in $\boldsymbol{R}$; S: arbitrary set; $\boldsymbol{L}$: arbitrary complete distributive lattice.

Figure C.5: Examples of commutative semirings

As clearly illustrated by the several distinct instances listed above, various algorithms have been independently designed in domains such as Information Theory, Digital Communications, Statistics, Artificial Intelligence, *etc.*., with specific domains and commutative semiring operations.

There are a few notions and notations used by the algorithm that need defining. We summarize them next. For more details, examples, and computation methods, the reader is referred to [146].[6]

- **Domains**: given finite discrete sets D_i, $i =, 1, \ldots, n$.
- **Variables**: finite set $\{x_1, \ldots, x_n\}$, where each variable x_i takes values in domain D_i.
- **Local Indices**: $\boldsymbol{I} \stackrel{\text{def}}{=} \{I_1, \ldots, I_m\}$, m subsets of the set of the n first natural number; *i.e.*, $I_j \subseteq \{1, \ldots, n\}$, for $j = 1, \ldots, m$.
- **Local Domain**: given the local index $I = \{i_1, \ldots, i_r\}$, its *local domain* $x_I \in D_I$ where $D_I \stackrel{\text{def}}{=} D_{i_1} \times \ldots \times D_{i_r}$.
- **Local Kernel**: $\alpha_i : D_{I_i} \to \boldsymbol{K}$ ($\boldsymbol{K}$ commutative semiring)
- **Global Kernel**: $\beta : D_1 \times \ldots \times D_n \to \boldsymbol{K}$ defined by $\beta(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \prod_{i=1}^{m} \alpha_i(x_{I_i})$.

[6] *Op. cit.*, Section 3: The Generalized Distributive Law Algorithm.

- **Marginalization** I_i-*marginalisation* of $\beta_i : D_{I_i} \to \boldsymbol{K}$:

$$\beta_i(x_{I_i}) \stackrel{\text{def}}{=} \sum_{x_{I_i^c} \in D_{I_i^c}} \beta(x_1, \ldots, x_n)$$

  also called *"(i-th) local objective function"*.

- **Junction Tree** tree whose nodes are the local indices such that, $\forall I, J, K$, if K is on the path from I to J, then $I \cap J \subseteq K$. Figure C.6 shows an example of a junction tree over four variables.

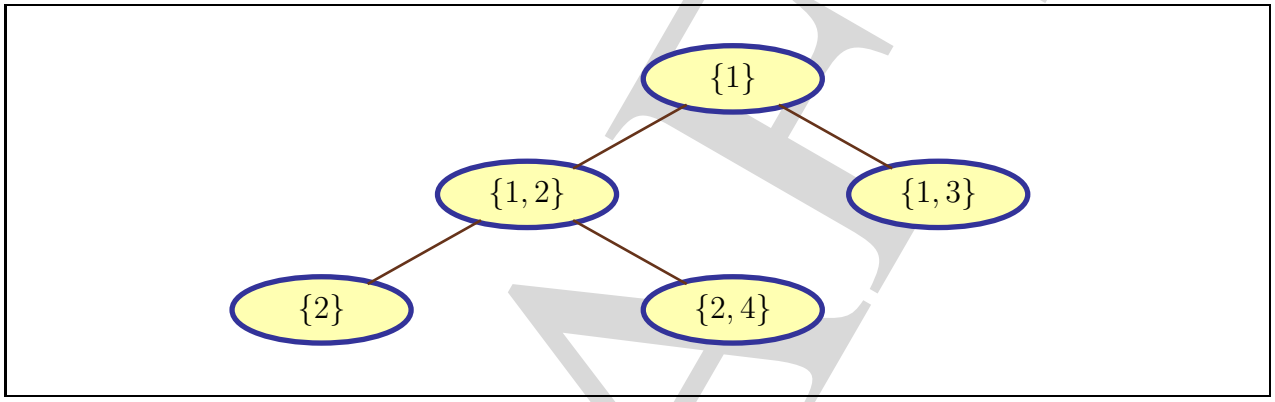


Figure C.6: Example of junction tree

The $\mathcal{GDL}$ algorithm consists in a variable elimination algorithm for computing local objective functions given local kernels on local domains.[7]

- *Basic idea:* "push sums in" as far and as much as possible, using distributivity: $\sum_i (a_i b) \to (\sum_i a_i) b$.
- *Basic technique:* Dynamic Programming on a *"junction tree."*

The $\mathcal{GDL}$ as a forward-backward message-passing algorithm: The $\mathcal{GDL}$ can be expressed as a message-passing algorithm [118] in Figure C.7. The updated "messages" are the elements of the $\mu_{i,j}$ table. They are passed from node I_i to node I_j — each node "sends a message" to a neighbor when it has received one from all its other neighbors *(first "upward" then "downward")*. Figure C.8 shows an example of the effect of the $\mathcal{GDL}$ message-passing algorithm of Figure C.7 using the junction tree of Figure C.6.

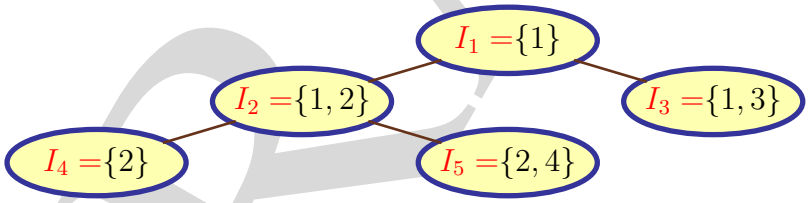
The $\mathcal{GDL}$'s (sequential) complexity is $\mathcal{O}\big(\sum_{I \in \boldsymbol{I}} |\boldsymbol{N}(I)||D_I|\big)$. See [28] for a detailed analysis, depending on whether one wishes to compute marginalization for a single vertex or for all vertices. Remarkably, computing marginalization for the complete set of vertices (rather than for just one vertex in the set) is only 4 times more expensive, no matter how many vertices in the set.

[7] The expression *bucket elimination* has been used to denote the basic junction tree technique [69, 70]. This is because it maximizes the number of eliminated variables per "pivot step."

Given a junction tree, the $\mathcal{GDL}$ algorithm iteratively updates a table $\mu_{i,j} : D_{I_i \cap I_j} \to \boldsymbol{K}$ for each node I_i, for all $I_j \in \boldsymbol{N}(I_i)$, where $\boldsymbol{N}(I)$ is the set of all neighbors of I, proceeding as follows:

0. for each node I_i, for each node $I_j \in \boldsymbol{N}(I_i)$: $\mu_{i,j} \leftarrow 1$
1. for each node I_i, for each node $I_j \in \boldsymbol{N}(I_j)$:
$$\mu_{i,j}(x_{I_i \cap I_j}) \leftarrow \sum_{x_{I_i - I_j} \in D_{I_i - I_j}} \alpha_i(x_{I_i}) \prod_{\substack{I_k \in \boldsymbol{N}_{(I_i)} \\ k \neq j}} \mu_{k,i}(x_{I_k \cap I_i})$$
2. for each node I_i: $\beta_i(x_{I_i}) \leftarrow \alpha_i(x_{I_i}) \prod_{I_k \in \boldsymbol{N}(I_i)} \mu_{k,i}(x_{I_k \cap I_i})$

Figure C.7: The $\mathcal{GDL}$ message-passing algorithm



	i,j	$\mu_{i,j}(x_{I_i \cap I_j})$
1	$3,1$	$\mu_{3,1}(x_1) = \sum_{x_3} \alpha_3(x_1, x_3)$
2	$4,2$	$\mu_{4,2}(x_2) = \alpha_4(x_2)$
3	$5,2$	$\mu_{5,2}(x_2) = \sum_{x_4} \alpha_5(x_2, x_4)$
4	$2,1$	$\mu_{2,1}(x_1) = \sum_{x_2} \alpha_2(x_1, x_2)\mu_{4,2}(x_2)\mu_{5,2}(x_2)$
5	$1,2$	$\mu_{1,2}(x_1) = \alpha_1(x_1)\mu_{3,1}(x_1)$
6	$1,3$	$\mu_{1,3}(x_1) = \alpha_1(x_1)\mu_{2,1}(x_1)$
7	$2,4$	$\mu_{2,4}(x_2) = \sum_{x_1} \alpha_2(x_1, x_2)\mu_{1,2}(x_1)\mu_{5,2}(x_2)$
8	$2,5$	$\mu_{2,5}(x_2) = \sum_{x_1} \alpha_2(x_1, x_2)\mu_{1,2}(x_1)\mu_{4,2}(x_2)$

Figure C.8: Example of the effect of the $\mathcal{GDL}$ message-passing algorithm

The key information it relies on is the existence of a junction tree. Testing for existence of a junction tree is simple. If one exists it is easy to build one: any maximum weight tree of the Local Domain Graph, whose vertices are the local indices I_i and weights $w_{i,j} = |I_i \cap I_j|$.[8] Otherwise, one can be built from the graph of all cliques of the triangulated moral graph, whose vertices are the variables x_i, and there is an edge between x_i and x_j iff $\{i, j\} \subseteq I \in \boldsymbol{I}$, for some local index I. This is an NP-hard problem, so information specific to an instance (such as properties of the domain, the data, or the operations) must be exploited.

[8] *N.B.*: $w_{\max} = \sum_{i=1}^{m}(|I_i| - n)$.

Bibliography

[1] Ágnes Achs and Attila Kiss. Fuzzy extension of Datalog. *Acta Cybernetica*, 12(2):153–166, 1995. [Available online].

[2] Alfred Aho, John Hopcroft, and Jeffrey Ullmann. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.

[3] Hassan Aït-Kaci. Outline of a calculus of type subsumption. Technical Report MS-CIS-83-34, Department of Computer and Information Science, The Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, PA (USA), August 1983. [Available online].

[4] Hassan Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures*. PhD thesis, Computer and Information Science, University of Pennsylvania, Philadelphia, PA (USA), September 1984. Abstract: [Available online].

[5] Hassan Aït-Kaci. An algorithm for finding a minimal recursive path ordering. *Revue d'Automatique, d'Informatique, et de Recherche Opérationnelle—Informatique théorique*, 19(4):359–382, 1985. [Available online].

[6] Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351, 1986. [Available online].

[7] Hassan Aït-Kaci. An introduction to LIFE: Programming with Logic, Inheritance, Functions, and Equations. In Dale Miller, editor, *Proceedings of the Logic Programming Symposium (ILPS-93)*, pages 52–68, Vancouver, BC (Canada), October 1993. MIT Press. [Available online]—see also the presentation slides.

[8] Hassan Aït-Kaci. Data models as constraint systems—A key to the Semantic Web. *Constraint Processsing Letters*, 1(1):33–88, November 2007. [Available online].

[9] Hassan Aït-Kaci. Description logic *vs.* order-sorted feature logic. In Enrico Franconi, editor, *Proceedings of the 20th International Workshop on Description Logics*, Lecture Notes in Computer Science. Springer-Verlag, 2007. [Available online].

[10] Hassan Aït-Kaci. $\mathcal{HOOT}$: a language for expressing and querying $\mathcal{H}$ierarchical $\mathcal{O}$ntologies, $\mathcal{O}$bjects, and $\mathcal{T}$ypes—a specification. Technical Report Number 16, $\mathcal{CEDAR}$ Project, LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, Villeurbanne, France, December 2014. [Available online].

[11] Hassan Aït-Kaci. A set-complete domain construction for order-sorted set-valued features. $\mathcal{CEDAR}$ Technical Report 11, LIRIS, Université Claude Bernard Lyon 1, Villeurbanne (France), September 2014. [Available online].

[12] Hassan Aït-Kaci. $\mathcal{FFF}$: a $\mathcal{F}$uzzy $\mathcal{F}$acility for $\mathcal{F}$irst-order terms. Java Software, August 2018. [Available online].

[13] Hassan Aït-Kaci. Generic linear fixed-point equation solving in arbitrary semi-rings: Object-oriented design and implementation. Technical Report, December 2019. [Available online].

[14] Hassan Aït-Kaci and Samir Amir. Classifying and querying very large taxonomies with bit-vector encoding. *Journal of Intelligent Information Systems*, 45(2):1–25, October 2015. [Available online].

[15] Hassan Aït-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January 1989. [Available online].

[16] Hassan Aït-Kaci and Roberto di Cosmo. Compiling order-sorted feature term unification. Technical Note 7, Digital Paris Research Laboratory, Rueil-Malmaison, France, December 1993. [Available online].

[17] Hassan Aït-Kaci, Bruno Dumant, Richard Meyer, Andreas Podelski, and Peter Van Roy. The Wild $\mathcal{LIFE}$ handbook. [Available online], 1994.

[18] Hassan Aït-Kaci and Patrick Lincoln. LIFE—a natural language for natural language. *T.A. Informations, Association pour le Traitement Automatique des Langues, Paris, France*, 30(1–2):37–67, 1989. [Available online].

[19] Hassan Aït-Kaci and Roger Nasr. Logic and inheritance. In *Proceeedings of the* 13^{th} *ACM Sigplan Conference on Principles of Programming Languages (POPL 1986)*, pages 219–228, St. Petersburg Beach, FL (USA), January 1986. Association for the Computing Machinery, ACM. [Available online].

[20] Hassan Aït-Kaci and Gabriella Pasi. Lattice operations on terms over similar signatures. In Fabio Fioravanti and John Gallagher, editors, *Proceedings of the* 27^{th} *International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2017)*, Namur, Belgium, October 20-12 2017. [Available online].

[21] Hassan Aït-Kaci and Gabriella Pasi. Lattice operations on terms over similar signatures—a constraint-based approach. Presentation slides for the 27^{th} *International Symposium on Logic-Based Program Synthesis and Transformation* (LOPSTR 2017), October 10–12, 2017. [Available online].

[22] Hassan Aït-Kaci and Gabriella Pasi. Fuzzy lattice operations on first-order terms over signatures with similar constructors: A constraint-based approach. *Fuzzy Sets & Systems*, 2020. [Available online].

[23] Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of $\mathcal{LIFE}$. *Journal of Logic Programming*, 16(3-4):195–234, 1993. [Available online].

[24] Hassan Aït-Kaci, Andreas Podelski, and Seth C. Goldstein. Order-sorted feature theory unification. *Journal of Logic Programming*, 30(2):99–124, 1997. [Available online].

[25] Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1–2):263–283, January 1994. [Available online].

DRAFT—PLEASE DO NOT DISTRIBUTE

[26] Hassan Aït-Kaci and Yutaka Sasaki. An axiomatic approach to feature term generalization. In Luc de Raedt and Peter Flach, editors, *Proceedings of the 12th European Conference on Machine Learning (ECML'01)*, pages 1–12, Berlin Heidelberg, September 2001. LNCS 2167, Springer-Verlag. [Available online].

[27] Srinivas M. Aji. *Graphical Models and Iterative Decoding*. PhD thesis, California Institute of Technology, Pasadena, CA (USA), May 2000. [Available online].

[28] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, March 2000. [Available online].

[29] Naseem Ajmal and K. V. Thomas. Fuzzy lattices. *Information Sciences*, 79:271–291, 1994. [Available online].

[30] María Alpuente, Santiago Escobar, Javier Espert, and José Meseguer. A modular order-sorted equational generalization algorithm. *Information and Computation*, 235:98–136, 2014. [Available online].

[31] María Alpuente, Santiago Escobar, José Meseguer, and Pedro Ojeda. Order-sorted generalization. *Electronic Notes in Theoretical Computer Science*, 246:27–38, 2009. [Available online].

[32] Teresa Alsinet, Llúıs Godo, and Sandra Sandri. Two formalisms of extended possibilistic logic programming with context-dependent fuzzy unification: a comparative description. *Electronic Notes in Theoretical Computer Science*, 66(5):21 pages, 2002. [Available online].

[33] Samir Amir and Hassan Aït-Kaci. An efficient and large-scale reasoning method for the semantic web. *Journal of Intelligent Information Systems*, 47(3):1–22, December 2016. [Available online].

[34] Krzysztof R. Apt. Logic programming. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 492–574. Elsevier, 1990.

[35] Francesca Arcelli and Ferrante Formato. A fuzzy logic programming language. In Luca Console, Evelina Lamma, Paola Mello, and Michela Milano, editors, *Programmazione Logica e Prolog*, pages 319–332. UTET Università, 1997. [Available online].

[36] Francesca Arcelli and Ferrante Formato. Likelog: A logic programming language for flexible data retrieval. In Hisham Al Haddad, editor, *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 260–267, San Antonio, TX (USA), February 28–March 2, 1999. Association for Computing Machinery, ACM. [Available online].

[37] Francesca Arcelli, Ferrante Formato, and Giangiacomo Gerla. Extending unification through similarity relations. *Bulletin pour les Sous Ensembles Flous et leurs Applications (BUSEFAL)*, pages 3–12, 1997. [Available online].

[38] Francesca Arcelli-Fontana. Likelog for flexible query answering. *Soft Computing*, 7(2):107–114, December 2002.

[39] Francesca Arcelli-Fontana and Ferrante Formato. A similarity-based resolution rule. *International Journal of Intelligent Systems*, 17:853–872, 2002. [Available online].

[40] Peter R.J. Asveld. Algebraic aspects of families of fuzzy languages. *Theoretical Computer Science*, 293(2):417–445, February 2003. [Available online].

[41] Franz Baader and Ulrike Sattler. Description logics with aggregates and concrete domains. In *Proceedings of the International Workshop on Description Logics*, Gif sur Yvette, France, 1997. [Available online].

[42] Rolf Backofen. Regular path expressions in feature logic. In Claude Kirchner, editor, *Proceedings of the 5th International Conference on Rewriting Techniques and Applications (RTA'93)*, pages 121–135, Montreal, QC (Canada), June 16–8, 1993. Springer. LNCS 690, [Available online].

[43] Mustapha Baziz, Mohand Boughanem, Gabriella Pasi, and Henri Prade. A fuzzy set approach to concept-based information retrieval. In E. Montseny and P. Sobrevilla, editors, *Proceedings of the Joint 4^{th} Conference of the European Society for Fuzzy Logic and Technology and the 11èmes Rencontres Francophones sur la Logique Floue et ses Applications, Barcelona (Spain)*, pages 1287–1292. EUSFLAT/LFA, September 7–9, 2005. [Available online].

[44] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, September 2, 1954. Invited address at the annual summer meeting of the AMS, Laramie, WY (USA) [Available online].

[45] P. George Benson, Shawn P. Curley, and Gerald F. Smith. Belief assessment: An underdeveloped phase of probability elicitation. *Management Science*, 41(10):1639–1653, October 1995. [Available online].

[46] Garrett Birkhoff. *Lattice Theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, Providence, RI (USA), 1940, revised 1948, 1967, 1979, and 1979. [Available online].

[47] Garrett Birkhoff and Jonh Non Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37(4):823–843, October 1936. [Available online].

[48] Stephano Bistarelli, Philippe Codognet, and Francesca Rossi. Abstracting soft constraints: Framework, properties, examples. *Artificial Intelligence Journal*, 139:175–211, 2002. [Available online].

[49] Fernando Bobillo and Umberto Straccia. Fuzzy ontology representation using OWL 2. *International Journal of Approximate Reasoning*, 52(7):1073–1094, October 2011. [Available online].

[50] Fernando Bobillo and Umberto Straccia. The fuzzy ontology reasoner *fuzzydl*. *Knowledge-Based Systems*, 95:12–34, December 2015. [Available online].

[51] Fernando Bobillo and Umberto Straccia. Fuzzy ontology representation using OWL 2. *International Journal of Approximate Reasoning*, 87:40–66, May 2017. [Available online].

[52] Gloria Bordogna, Dario Lucarella, and Gabriella Pasi. A fuzzy object oriented data model. In *Proceedings of the 3^{rd} IEEE Conference on Fuzzy Systems*. IEEE World Congress on Computational Intelligence, July 1994. [Available online].

[53] Ronald J. Brachman. *A Structural Paradigm for Representing Knowledge*. PhD thesis, Artificial Intelligence, Harvard University, Cambridge, MA (USA), 1977. Available as BBN Technical Report 3605 from Bolt Beranek and Newman Inc. (1978).

[54] Tru H. Cao and Peter N. Creasy. Fuzzy types: a framework for handling uncertainty about types of objects. *International Journal of Approximate Reasoning*, 25(3):217–253, November 2000. [Available online].

[55] Yongzhi Cao and Yoshinori Ezawac. Nondeterministic fuzzy automata. *Information Sciences*, 191:86–97, 2012. [Available online].

[56] Bob Carpenter. Typed feature structures: A generalization of first-order terms. In Vijay Saraswat and Kazunori Ueda, editors, *Proceedings of the 1991 International Symposium on Logic Programming*, pages 187–201, Cambridge, MA, 1991. MIT Press.

[57] Bernard Carré. *Graphs and Networks*. Oxford University Press, 1979. [Available online].

[58] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, March 1989. [Available online].

[59] Aarthi Chandramohan and M. V. C. Rao. Novel, useful, and effective definitions for fuzzy linguistic hedges. *Discrete Dynamics in Nature and Society*, 2006(Article ID 46546):1–13, 2006. [Available online].

[60] Inheung Chon. Fuzzy partial order relations and fuzzy lattices. *Korean Journal of Mathematics*, 17(4):361–374, 2009. [Available online].

[61] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer-Verlag, second edition, 1984.

[62] Alain Colmerauer. Prolog and infinite trees. In Keith L. Clark and Sten Åke Tärnlund, editors, *Logic Programming*, pages 231–251. Academic Press, 1982.

[63] Maria Eugenia Cornejo, Jesús Medina Moreno, and Clemente Rubio-Manzano. Towards a full fuzzy unification in the bousi prolog system. In Fernando Gomide, editor, *Proceedings of the International Conference on Fuzzy Systems (FUZZ-IEEE 2018)*, July 2018. [Available online].

[64] Patrick Cousot. Verification by abstract interpretation, soundness and abstract induction. In Association for Computing Machinery, editor, *Proceedings of the 17th International Conference on Principles and Practice of Declarative Programming (PPDP-2015)*, June 2015. [Available online].

[65] Patrick Cousot. *Principles of Abstract Interpretation*. MIT Press, Cambridge, MA (USA), first edition, 2020. (forthcoming).

[66] Patrick Cousot and Radhia Cousot. Semantic foundations of program analysis. In Steven S. Muchnick and Neil D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 10, pages 303–342. Prentice-Hall, Englewood Cliffs, NJ (USA), 1981. [Available online].

[67] Patrick Cousot and Radhia Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Pogramming*, 13(2-3):103–179, 1992. [Available online] https://www.di.ens.fr/ cousot/COUSOTpapers/publications.www/CousotCousot-JLP-v2-n4-p511–547-1992.pdf https://www.di.ens.fr/∼cousot/COUSOTpapers/publications.www/CousotCousot-JLP-v2-n4-p511–547-1992.pdf.

[68] Andreas de Vries. Algebraic hierarchy of logics unifying fuzzy logic and quantum logic. *ArXiv e-prints*, 0707(2161), July 2007. [Available online].

[69] Rina Dechter. Bucket elimination: A unifying framwork for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[70] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[71] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Transactions on Algorithms*, 6(1):article 2, December 2009. [Available online].

[72] Arthur P. Dempster, Nan M. Laird, Donald B. Rubin, and *et al.*. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society*, 39(1):1–38, 1977.

[73] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science—Volume B: Formal Methods and Semantics*, chapter 6, pages 243–320. North-Holland, Amsterdam, The Netherlands, 1990. [Available online].

[74] Didier Dubois and Henri Prade. *Fuzzy Sets and Systems: Theory and Applications*, volume 144 of *Mathematics in Science and Engineering, Edited by William F. Ames, Georgia Institute of Technology*. Academic Press, 180. [Available online].

[75] Elmar Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, 1:31–46, 1985. [Available online].

[76] Santiago Escobar, José Meseguer, and Prasanna Thati. Narrowing and rewriting logic: from foundations to applications. *Electronic Notes in Theoretical Computer Science*, 177:5–33, 2007. [Available online].

[77] M.J. Fay. First order unification in an equational theory. In W.H. Joyner, editor, *Proceedings of the 4^{th} Workshop on Automated Deduction (Austin, TX)*, pages 161–167, London, UK, 1979. Academinc Press.

[78] Leonidas Fegaras. Query unnesting in object-oriented databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 49–60, Seattle, WA (USA), June 2–4 1998. [Available online].

[79] Leonidas Fegaras and David Maier. Optimizing object queries using an effective calculus. *ACM Transactions on Database Systems*, 25(4):457–516, December 2000. [Available online].

[80] Roman Frič and Martin Papčo. Fuzzification of crisp domains. *Kybernetika*, 46(6):1009–1024, 2010. [Available online].

[81] David Gilbert and Michael Schroeder. FURY: Fuzzy unification and resolution based on edit distance. In Nikolaos G. Bourbakis, editor, *Proceedings of the 1^{st} IEEE International Symposium on Bioinformatics and Biomedical Engineering (BIBE 2000)*, pages 330–336, Arlington, VA (USA), November 8–10, 2000. IEEE Computer Society. [Available online].

[82] Joseph Goguen. What is unification? In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures—Algebraic Techniques*, chapter 6, pages 217–261. Academic Press, 1989. [Available online].

[83] Joseph A. Goguen. L-fuzzy sets. *Journal of Mathematical Analysis and Applications*, 18:145–174, 1967. [Available online].

[84] Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial algebra semantics and continuous algebras. *Journal of the Association for Computing Machinery*, 24(1):68–95, January 1977. [Available online].

[85] Peter V. Golubtsov and Stepab S. Moskaliuk. Bayesian decisions and fuzzy logic. ESI Preprint 626, Erwin Schrödinger Internation Institute for Mathematics and Physics, Vienna, Austria, October 29, 1998. [Available online].

[86] Saul Gorn. Explicit definitions and linguistic dominoes. In John F. Hart and Satoru Takasu, editors, *Systems and Computer Science*, pages 77–105. University of Toronto Press, Toronto, ON (Canada), 1965.

[87] Saul Gorn. Data representation and lexical calculi. *Information Processing & Management*, 20(1,2):151–174, 1984.

[88] Saul Gorn. Self-annihilating sentences: Saul gorn's compendium of rarely used clichés. Technical Report N^o. MS-CIS-92-25, University of Pennsylvania, Department of Computer and Information Science, Philadelphia, PA (USA), January 1985. [Available online].

[89] Lovre Grisogono. Classical logic *vs.* quantum logic. Lecture slides, University of Zagreb, Croatia, July 8–9 2013. [Available online].

[90] Torsten Grust. A versatile representation for queries. In P.M.D. Gray, L. Kerschberg, P.J.H. King, and A. Poulovassilis, editors, *The Functional Approach to Data Management: Modeling, Analyzing and Integrating Heterogeneous Data*. Springer, September 2003. [Available online].

[91] Stanley Gudder. What is fuzzy probability theory? *Foundations of Physics*, 30(10), 2000. [Available online].

[92] Hyowon Gweon, Joshua B. Tenenbaum, and Laura E. Schulz. Infants consider both the sample and the sampling process in inductive generalization. In Susan E. Carey, editor, *Proceedings of the National Academy of Sciences*, pages 9066–9071, Harvard University, Cambridge, MA (USA), May 18, 2010. PNAS vol. 107, no. 20, National Academy of Sciences of the United States of America. [Available online].

[93] Olle Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[94] Yann-Michaël De Hauwere. Declarative programming: Inductive logic programming. Tutorial lecture slides, 2015. [Available online].

[95] Jacques Herbrand. *Recherches sur la théorie de la démonstration*. PhD thesis, Faculté des sciences de l'université de Paris, Paris (France), 1930. [Available online]; English translation in [96].

[96] Jacques Herbrand. *Logical Writings*. Harvard University Press, Cambridge, MA, 1971. Edited by Warren D. Goldfarb.

[97] Gérard Huet. *Résolution d'Équations dans des Langages d'Ordre* $1, 2, \ldots, \omega$. *Thèse d'état, Université de Paris VII*, Paris (France), 1976. [Available online].

[98] Jelena Ignjatović and Miroslav Ćirić. Myhill-nerode theory for fuzzy languages and automata. Lecture Slides, AUTOMATHA Workshop on Algebraic Theory of Automata and Logic, Szeged, Hungary[1], September 30–October 1 2006. [Available online].

[1] http://www.inf.u-szeged.hu/∼csl06/ws.php

[99] Pascual Julián Iranzo and Clemente Rubio Manzano. A similarity-based WAM for Bousi∼Prolog. In *Proceedings of the 10th International Work-Conference on Artificial Neural Networks (IWANN '09)—Part I: Bio-Inspired Systems: Computational and Ambient Intelligence*, pages 245–252, Salamanca, Spain, June 10–12, 2009. Springer. [Available online].

[100] Pascual Julián Iranzo and Clemente Rubio Manzano. An efficient fuzzy unification method and its implementation into the Bousi∼Prolog system. In *Proceedings of 19th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2010)*, pages 658–665, Barcelona, Spain, July 18–23, 2010. IEEE. [Available online].

[101] Pascual Julián Iranzo and Clemente Rubio Manzano. A sound and complete semantics for a similarity-based logic programming language. *Fuzzy Sets & Systems*, 317:1–26, 2017.

[102] Pascual Julián Iranzo, Ginés Moreno, Jaime Penabad, and Carlos Vázquez. A fuzzy logic programming environment for managing similarity and truth degrees. In Santiago Escobar, editor, *Proceedings XIV Jornadas sobre Programación y Lenguajes (PROLE 2014)*, pages 71–86. Electronic Proceedings in Theoretical Computer Science, EPTCS 173, January 2015. [Available online].

[103] Joxan Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2(3):207 – 219, September 1984. [Available online].

[104] Mark Johnson. Memoization in constraint logic programming. In Paris Kanellakis, Jean-Louis Lassez, and Vijay Saraswat, editors, *Proceedings of the 1st International Workshop on Principles and Practice of Constraint Programming (PPCP 1993)*, pages 130–138, Newport, RI (USA), 1993. [Available online].

[105] Petteri Jokinen, Jorma Tarhio, and Esko Ukkonen. A comparison of approximate string matching algorithms. *Software–Practice and Experience*, 1(1988):1–4, January 1. [Available online].

[106] Janusz Kacprzyk and Augustine O. Esogbue. Fuzzy dynamic programming: Main developments and applications. *Fuzzy Sets & Systems*, 81:31–45, 1996. [Available online].

[107] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object oriented and frame based languages. *Journal of the ACM*, 42(4):741–843, 1995. [Available online].

[108] Uffe Kjærulff and Anders Madsen. Probabilistic networks—an introduction to Bayesian networks and influence diagrams, May 2005. [Available online].

[109] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, UK, 1970. Reprinted in *Automatic Reasoning*, 2, Springer-Verlag, pp. 342–276 (1983).

[110] Bart Kosko. Fuzziness *vs.* probability. *International Journal of General Systems*, 17(1):211–240, 1990. [Available online].

[111] Robert A. Kowalski. *Logic for Problem Solving*, volume 7 of *Artificial Intelligence Series*. North Holland, New York, NY, 1979.

[112] Temur Kutsia. Pattern unification with sequence variables and flexible arity symbols. *Electronic Notes in Theoretical Computer Science*, 66(5):52–69, 2002. [Available online].

[113] Temur Kutsia, Jordi Levy, and Mateu Villaret. Anti-unification for unranked terms and hedges. *Journal of Automated Reasoning*, 52(2):155–190, 2014. [Available online].

[114] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani. SiGMa: Simple greedy matching for aligning large knowledge bases. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, editors, *Proceedings of the 19th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2013—Chicago, IL (USA))*, pages 572–580, New York, NY (USA), August 11–14, 2013. Association for Computing Machinery, ACM. [Available online]; see also [Available online].

[115] Harri Lähdesmäki and Ilya Shmulevich. Learning the structure of dynamic Bayesian networks from time series and steady state measurements. *Machine Learing*, 71(2–3):185–217, 2008.

[116] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics–Doklady, Cybernetics and Control Theory*, 10(8):707–710, February 1966. [Available online].

[117] Ravi K. M. *Some Investigations in Fuzzy Automata*. PhD thesis, Jaypee Institute of Information Technology, Noida (India), February 2012. [Available online].

[118] David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[119] Bernd Mahr and Johann A. Makowsky. Characterizing specification languages which admit initial semantics. *Theoretical Computer Science*, 31(1–2):49–59, 1984. [Available online].

[120] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982. [Available online].

[121] Takashi Mitsuishi and Grzegorz Bancerek. Lattice of fuzzy sets. *Formalized Mathematics*, 11(4):393–398, 2003. [Available online].

[122] Todd K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996.

[123] Jiří Močkoř. Fuzzy and non-deterministic automata. Research Report 8, University of Ostrava, Institute for Research and Applications of Fuzzy Modeling, Ostrava (Czech Republic), November 6, 1997. [Available online].

[124] Jiří Močkoř. Fuzzy and non-deterministic automata. *Soft Computing*, 3(4):221–226, December 1999. See also [123].

[125] Radu Stefan Niculescu, Tom M. Mitchell, and R. Bharat Rao. Bayesian network learning with parameter constraints. *Journal of Machine Learning Research*, 7:1357–1383, July 2006. [Available online].

[126] Santiago Onta nón and Enric Plaza. Similarity measures over refinement graphs. *Machine Learning*, 87(1):57–92, 2012. [Available online].

[127] Peter Norvig and Stuart J. Russell. Constraint satisfaction problems. Chapter 6 of [128]. [Available online] (**slides**): [Available online].

[128] Peter Norvig and Stuart J. Russell. *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice-Hall, third edition, 2010. (see chapter on $\mathcal{CSP}$ [127]).

[129] Angmin O, Jae Won Lee, Sung-Bae Parkl, and Byoung-Tak Zhangl. Stock trading by modelling price trend with dynamic Bayesian networks. In *Intelligent Data Engineering and Automated Learning (IDEAL 2004)*, pages 794–799. Springer, Exeter, UK, August 25–27 2004.

[130] World Intellectual Property Organization. *Artificial Intelligence*. WIPO Technology Trends. WIPO, Geneva (Switzerland), 2019. [Available online].

[131] Osonde Osoba, Sanya Mitaim, and Bart Kosko. Bayesian inference with adaptive fuzzy priors and likelihoods. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 41(5):1183–1197, October 2011. slides: [Available online].

[132] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, second edition, 1988.

[133] Benjamin Pierce. *Basic Category Theory for the Computer Scientists*. MIT Press, 1991.

[134] Gordon D. Plotkin. Lattice theoretic properties of subsumption. Technical Memo MIP-R-77, Department of Machine Intelligence and Perception, University of Edinburgh, Edinburgh, Scotland (UK), June 1970.

[135] Gordon D. Plotkin. A note on inductive generalization. In Bernard Metzer and Donald Michie, editors, *Machine Intelligence 5*, chapter 8, pages 154–163. Edinburgh University Press, Edinburgh, Scotland (UK), 1970. [Available online].

[136] Alok K. Porwal, E. John M. Carranza, and Martin Hale. Bayesian network classifiers for mineral potential mapping. *Computers & Geosciences*, 32(1):1–16, February 2006. [Available online].

[137] Dag Prawitz. An improved proof procedure. *Theoria*, 26:102–139, August 1960. [Available online].

[138] Jarosław Pykacz. Fuzzy quantum logic I. *International Journal of Theoretical Physics*, 32(10):1691–1708, October 1993.

[139] Jarosław Pykacz. Quantum structures and fuzzy set theory. In Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann, editors, *Handbook of Quantum Logic and Quantum Structures—Quantum Structures*, pages 55–74. Elsevier, 2007. [Available online].

[140] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[141] John C. Reynolds. Transformational systems and the algebraic nature of atomic formulas. In Bernard Metzer and Donald Michie, editors, *Machine Intelligence 5*, chapter 7, pages 135–151. Edinburgh University Press, Edinburgh, Scotland (UK), 1970. [Available online].

[142] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, January 1965. [Available online].

[143] Natalia Díaz Rodríguez. Dealing with uncertainty: Fuzzy (description) logics and fuzzy ontologies. Logic and AI Lecture Notes 2019–2020, September 2018. [Available online].

[144] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier, 2006. [Available online].

[145] Kevin Sancho. `Cedar.Gdl` java library user manual. Technical Report Number 10, $\mathcal{CEDAR}$ Project, LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, Villeurbanne, France, July 2014. [Available online].

[146] Kevin Sancho and Hassan Aït-Kaci. The `Cedar.Gdl` java library for the generalized distributive law—design and implementation. Technical Report Number 9, $\mathcal{CEDAR}$ Project, LIRIS, Département d'Informatique, Université Claude Bernard Lyon 1, Villeurbanne, France, July 2014. [Available online].

[147] Yutaka Sasaki. Applying type oriented ILP to IE rule generation. In *Proceedings of the Workshop on Machine Learning for Information Extraction*, pages 43–47, Orlando, FL (USA), 1999. AAAI-99. [Available online].

[148] Yutaka Sasaki. Induction logic based on ψ-terms. In *Proceedings of the 10th International Conference on Algorithmic Learning Theory*, pages 169–181, Tokyo, Japan, 1999. ALT-99, Springer-Verlag LNAI 1720.

[149] Yutaka Sasaki. *Hierarchically Sorted Inductive Logic Programming and its Application to Information Extraction*. PhD thesis, Graduate School of Systems and Information Engineering, University of Tsukuba, Japan, September 2000.

[150] Yutaka Sasaki and Masahiko Haruno. RHB$^+$: A type oriented ILP system learning from positive data. In *Proceedings of IJCAI-97*, pages 894–899, Nagoya, Japan, 1997. [Available online].

[151] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[152] Ingo Schmitt, Andreas Nürnberger, and Sebastian Lehrack. On the relation between fuzzy and quantum logic. In Rudolf Seising, editor, *Views on Fuzzy Sets and Systems from Different Perspectives—Philosophy and Logic, Criticisms and Applications*, Studies in Fuzziness and Soft Computing, chapter 20, pages 417–438. Springer, March 2009. [Available online].

[153] Michael Schroeder. Re: your work on FURY. Private Communication (email), October 20, 2017. [Available online].

[154] Michael Schroeder and Ralf Schweimeier. Arguments and misunderstandings: Fuzzy unification for negotiating agents. *Electronic Notes in Theoretical Computer Science*, 70(5):1–19, October 2002. [Available online].

[155] Ralf Schweimeier and Michael Schroeder. Fuzzy unification and argumentation for well-founded semantics. In Peter Van Emde Boas and Július Štuller Jaroslav Pokorný, Mária Bieliková, editors, *Proceedings of the 30th Conference on Current Trends in Theory and Practice of Computer Science*, pages 102–121, Měřín, Czech Republic, 24–30 January 2004. LNCS 2932, Springer, Lecture Notes in Computer Science. [Available online].

[156] Mathieu Serrurier, Thomas Sudkamp, Didier Dubois, and Henri Prade. Fuzzy inductive logic programming: Learning fuzzy rules with their implication. In *Proceedings of the 14th IEEE International Conference on Fuzzy Systems (FUZZ'05)*, pages 613–618, Reno, NV (USA), May 25–25 2005. IEEE.

[157] Maria I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275:389–426, 2002. [Available online].

[158] Nate Silver. *The signal and the noise—Why so many predictions fail, but some don't*. Penguin Press, New York, NY (USA), 2012. [Available online].

[159] Umberto Straccia. A fuzzy description logic. In Jack Mostow and Chuck Rich, editors, *Proceedings of the 15th AAAI Conference on Artificial Intelligence (AAAI-98)*, Madison, WI (USA), July 26–30 1998. American Association for Artificial Intelligence. [Available online].

[160] Umberto Straccia. *Foundations of a Logic based approach to Multimedia Document Retrieval*. PhD thesis, *Universität Dortmund, Fachbereich Informatik*, Dortmund (Germany), June 6, 1999. [Available online].

[161] Umberto Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001. [Available online].

[162] S.P. Tiwari, Anupam K. Singh, and Shambhu Sharan. Fuzzy subsystems of fuzzy automata based on lattice-ordered monoid. *Annals of Fuzzy Mathematics and Informatics*, 7(3):437–445, March 2013. [Available online].

[163] S.P. Tiwari, Vijay K. Yadav, and Anupam K. Singh. On algebraic study of fuzzy automata. *International Journal of Machine Learning and Cybernetics*, 6(3):479–485, June 2015.

[164] Dorothea Tsatsou, Stamatia Dasiopoulou, Ioannis Kompatsiaris, and Vasileios Mezaris. LiFR: A lightweight fuzzy $\mathcal{DL}$ reasoner. In Valentina Presutti, Eva Blomqvist, Raphael Troncy, Harald Sack, Ioannis Papadakis, and Anna Tordai, editors, *Proceedings of the European Semantic Web Conference (ESWC-2014)*, pages 263–267, Anissaras, Crete (Greece), May 25–29 2014. ESWC, Springer. [Available online].

[165] Maarten H. van Emden and John W. Lloyd. A logical reconstruction of Prolog II. *Journal of Logic Programming*, 2:143–149, 1984. [Available online].

[166] Jean van Heijenoort. *From Frege to Gödel—A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, Cambridge, MA (USA), 1967.

[167] Claudio Vaucheret, Sergio Guadarrama, and Susana Mu noz Hernández. Fuzzy Prolog: a simple general implementation using CLP($\mathbb{R}$). In Matthias Baaz and Andrei Voronkov, editors, *Proceedings of the 9th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2002)*, pages 450–464, Tbilisi, Georgia, October 14–18, 2002. LNCS 2514, Springer. [Available online].

[168] Enrique Vidal, Andrés Marzal, and Pablo Aibar. Fast computation of normalized edit distances. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(9):899–902, Septembre 1995. [Available online].

[169] Reinhard Viertl and Owat Sunanta. Fuzzy bayesian inference. Forschungsbericht SM-2013-2, Institut für Statistik und Wahrscheinlichkeitstheorie, Vienna, Austria, June 2013. [Available online].

[170] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, March 1967.

DRAFT—PLEASE DO NOT DISTRIBUTE

[171] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974. [Available online].

[172] Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, January 1962.

[173] Kevin Wayne. Union-find. Tutorial lecture slides based on book "Algorithm Design" by Jon Kleinberg and Éva Tardos (Addison-Wesley, 2015). [Available online].

[174] Raymond T. Yeh. Toward an algebraic theory of fuzzy relational systems. Technical Report TR-25, Department of Computer Sciences and Electronics Research Center, the University of Texas at Austin, Austin, TX (USA), July 1973. [Available online].

[175] John Yen. Generalizing term subsumption languages to fuzzy logic. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 472–477, Sydney, Australia, August 24–30, 1991. IJCAI, Morgan Kaufmann. [Available online].

[176] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965. [Available online].

[177] Lotfi A. Zadeh. Similarity relations and fuzzy orderings. *Information Sciences*, 3:177–200, 1971. [Available online].

[178] Lotfi A. Zadeh. A fuzzy-set-theoretic interpretation of linguistic hedges. *Journal of Cybernetics*, 2(3):4–34, 1972. [Available online].

[179] Lotfi A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning—i. *Information Sciences*, 8:199–249, 1975. [Available online].

[180] Hans-Jürgen Zimmermann. *Fuzzy Set Theory—and Its Applications*. Springer Science+Business Media, New York, NY (USA), fourth edition, 2001. [Available online].

[181] Geoffrey Zweig and Stuart Russell. Probabilistic modeling with Bayesian networks for automatic speech recognition. *Australian Journal of Intelligent Information Processing*, 1999. [Available online].

Fuzzy Lattice–Theoretic Operations over Data and Knowledge Structures

AÏT-KACI, Hassan; PASI, Gabriella

Draft version of April 10, 2020
202 pages