Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

Categorical Unification

Ph.D. Thesis
UMINF 04.09

María Ángeles Galán García
magalan@cs.umu.se

Print & Media, Umeå University 2004

UMINF 04.09 ISBN 91-7305-642-1 ISSN 0348-0542

To my parents,
María Ángeles and José María

"The longer mathematics lives
the more abstract - and therefore, possibly
also the more practical - it becomes"
(Eric Temple Bell)

Abstract

This thesis deals with different aspects towards many-valued unification which have been studied in the scope of category theory. The main motivation of this investigation comes from the fact that in logic programming, classical unification has been identified as the provision of coequalizers in Kleisli categories of term monads. Continuing in that direction, we have used categorical instrumentations to generalise the classical concept of a term. It is expected that this approach will provide an appropriate formal framework for useful developments of generalised terms as a basis for many-valued logic programming involving an extended notion of terms.

As a first step a concept for *generalised terms* has been studied. A generalised term is given by a composition of monads that again yields a monad, i.e. compositions of powerset monads with the term monad provide definitions for generalised terms. A composition of monads does, however, not always produce a monad. In this sense, techniques for monads composition provide a helpful tool for our concerns and therefore the study of these techniques has been a focus of this research.

The composition of monads make use of a lot of equations. Proofs become complicated, not to mention the challenge of understanding different steps of the equations. In this respect, we have studied visual techniques and show how a graphical approach can provide the support we need.

For the purpose of many-valued unification, similarity relations, generalised substitutions and unifiers have been defined for generalised terms.

Acknowledgments

Many persons have contributed to this thesis, directly or indirectly. Without their help and support this work would not have been possible.

I wish to thank my supervisor Patrik Eklund for giving me the opportunity to start these studies. His guidance and his positive attitude (*'now we are cooking!'*) are much appreciated. He has been very generous with taking time to answer so many questions.

I also want to express my appreciation to Inmaculada Pérez de Guzmán Molina, head of the Department of Applied Mathematic, University of Málaga and head of GIMAC research group, for continuously encouraging me to get this thesis finished.

I would like to thank Lennart Edblom, head of the Department of Computer Science for generously supporting travels. This has given me opportunity of meeting researchers in my area.

Jesús Medina Moreno, Manuel Ojeda Aciego and Agustín Valverde Ramos, co-authors of the papers, thank you for your generous collaboration, for offering me your friendship and for helping and supporting me during the past years. Thanks also for the helpful comments on earlier versions of this thesis.

I would like to express gratitude to all the colleagues at the Department of Computing Science for providing a good working atmosphere. Special thanks go to Inger Sandgren and Anne-Lie Persson, secretaries at this department, for helping me with all possible administrative tasks and for their personal support.

Also thanks to the Department of Applied Mathematic and GIMAC research group, University of Málaga, for their generous cooperation.

Finally, on a more personal level I wish to thank my family and friends that always have supported me.

My parents, María Ángeles García Galán and José María Galán Abarca, deserve a *special* thanks for all their love, care, support and their understanding. They were the first ones encouraging me to pursue higher education. This thesis is for you!

Deep thanks to my sister Miki (and Pedro and Jesús and Lucía) and my brother José Luis (and Mavi and María) for all the support and love they have given me.

Johan, what can I say? Thanks for reminding me of the good in life and for sharing your life with me. You make everything else seem irrelevant!

Birgitta and Christer, thanks for your hospitality and your kindness!

Friends are very important in life. Thanks Annabella, Thomas, Pedher, Mike and Helena for making my stay in Sweden more pleasant.

I cannot forget my closest friends from Málaga, *'Las niñas'* i.e. Victoria, Raquel, Esther, Lourdes, Irene, Ana, Ana Belén, Fefa, that have provided me not only with their support but also with very good company, laughs, dinners and parties (even from the distance!). Thanks to all of you!

This work has been developed as a cooperation organised within COST Action 274, TARSKI.

This research has been partially supported by the Swedish Research Council for Engineering Sciences.

Umeå, April 2004

M.Ángeles

Table of Contents

Preface

Unification in the classical sense covers one important block in Computer Science. The classical unification algorithm can be implemented in languages such as Prolog, making it an important tool for logic programming. At this level concepts such as terms, substitutions, unifiers, have a very specific use. Nevertheless, unification tasks can also be explained from a more theoretical point of view.

By making a bit of abstraction we can bring ourselves to the area of Category Theory. We need to change our vocabulary of types and operations into a new vocabulary of categories (objects and morphisms). Substitutions translate to certain mappings and unifiers to certain kind of limits of diagrams.

In 1987 Rydeheard and Burstall [56] presented unification in classical logic as finding "coequalizers" in a suitable category. The aim of this thesis is to generalise those results to approach the many-valued case. The abstract side would bring us to higher order categorical logic, where the main ingredients to do many-valued unification should be defined.

In this thesis we answer the question about how to define those concepts. Terms translate now into generalised terms, variable substitutions into generalised variable substitutions, equality relations into similarity relations.

We start by presenting different ways of extending the concept of terms using monads compositions. Techniques for monad composition are also studied to provide a powerful tool to reach our objectives. Graphical representations are introduced to handle equations in a hopefully more readable format. This is particularly important

as proof support. In the vocabulary of category theory, Kleisli constructions will allow us to compose the generalised variable substitutions. Finally, keeping in mind our interest in many-valued unification, similarity relations and unifiers are presented for generalised terms.

This thesis is structured as follows: Part I consists of a introduction to the area of our research together with a short summary of our research work. A summary of some of our papers and some final remarks can be found in Part II. Finally the reader can find some representative papers that are selected to be the main contributions to this thesis:

1. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Set functors, L-fuzzy set categories and generalised terms*", Computers and Mathematics with Applications, 43 (2002), pp. 693-705.

2. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*A graphical approach to monad compositions*", In Electronic Notes in Theoretical Computer Science 40 (2001).

3. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Powersets of Terms and Composite Monads*", Technical Report UMINF-04.07, 2004. Department of Computing Science, Umeå University, Sweden.

4. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*A categorical approach to unification of generalised terms*", In Electronic Notes in Theoretical Computer Science 66.5 (2002).

5. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Similarities between powersets of terms*", Fuzzy Sets and Systems (2003). In press.

In addition, the following papers have also been produced in relation to this research area:

6. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*A framework for unification using powersets of terms*", Proc. 9th Information Processing and Management of Uncertainty in Knowledge Based Systems Conference (IPMU 2002), pp. 1095-1098.

7. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Set functors, L-fuzzy set categories: towards a fuzzy programming paradigm*", Proc. International ICSC Symposium on Fuzzy Logic and Applications (FLA'2001), pp. 658-664.

8. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Composing submonads*", Proc. International Symposium on Multiple Valued Logics, ISMVL 2001, 367-372.

9. Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Generalised terms and composition of monads*", Proc. X Spanish Conference on Fuzzy Logic and Technologies (ESTYLF 2000), pp. 155-160.

10. Eklund, P., Galán, M.A., Ojeda-Aciego, M. and Valverde, A. "*Set functors and generalised terms*", Proc. 8th Information Processing and Management of Uncertainty in Knowledge-Based Systems Conference (IPMU 2000), pp. 1595-1599.

And finally, a Licentiate thesis dissertation was also achieved:

11. Galán, M.A., "*Generalised Terms*", Licentiate Thesis, UMINF 01.09, ISSN-0348-0542, May 11, 2001. Department of Computing Science, Umeå University, Sweden.

The list of references is included at the end of Part II.

Categorical Unification

Part I

This first part of the thesis gives an overview of our research in many-valued logic. Starting in Chapter 1, by presenting in an informal (maybe not that informal) way some of the main concepts we are working with and by describing the main motivations for this thesis, we continue in Chapter 2 by giving a general overview about how our research have developed within the scope of monads. General techniques for monad compositions presented in Chapter 2, will be used in Chapter 3 to give rise to the concept of *generalised term*. Visual representations will be introduced in Chapter 4, providing a new notation and a better way to handle equations coming from the naturality properties of the monads. Finally, in Chapter 5, similarity relations are defined as a tool to approach categorical unification in the many-valued case.

Chapter 1

Introduction

The need to describe in a formal way how a mathematical structure could give rise to another, was how categories arrived to mathematics. In fact, category theory arose when some formal characterizations for 'canonic constructions' were developed in topology. In this sense, category theory provides a new way to formalize mathematics.

Formal descriptions in mathematical logic are traditionally given as formal languages with rules for forming terms, axioms and equations. From an algebraic point of view, based on signature operations and equations one can give formalisms to describe logic. The contribution of category theory to mathematical logic is based on the fact that many logical constructs can be characterized in terms of relatively few categorical ones, principal among which is the concept of adjoint functor.

One of the basic connections between category theory, logic and computer science is that *cartesian closed categories* are intimately related to the λ *calculus*. In this context we speak about the *internal language of a topos* [6].

Topology, algebra, geometry and functional analysis are good examples of areas in which category theory has been successfully applied. Nowadays many other research areas, including computer science, are being developed focusing on categorical aspects.

In functional programming, the abstraction provided by category theory brings a

recognition of some of the constructions as categories: deduction systems are essentially categories. In the recent years there has also been a growing interest towards categorical models for term rewriting systems [45]. Some other current applications of category theory are in the area of game semantics, concurrency. See [54], [1] for instance.

In this sense we could consider category theory as an open and developing field among researchers in computing science. For a detailed motivation on how category theory is useful in computing science, the reader can check [30]. In this paper Goguen gives some guidelines for applying some of the basic categorical concepts to specific areas within computer science.

1.1 Basic Concepts

This section presents a brief introduction of the very basic notions of category theory. Some examples are also included in order to clarify the definitions. The goal of this section is to prepare and provide the reader that is not familiar with category theory with some of the basic concepts that are mentioned in the thesis.

1.1.1 Categories

We begin by defining the notion of *category* and by presenting some examples.

Definition 1.1 *A category is a quintuple* $\mathcal{C} = (\mathrm{Ob}, \mathrm{Mor}, \mathrm{dom}, \mathrm{cod}, \circ)$ *where*

(i) Ob *is a class whose members are called* $\mathcal{C}$*-objects,*

(ii) Mor *is a class whose members are called* $\mathcal{C}$*-morphisms,*

(iii) dom *and* cod[1] *are functions from* Mor *to* Ob*,*

[1] $\mathrm{dom}(f)$ *is called the domain of* f *and* $\mathrm{cod}(f)$ *is called the codomain of* f

(iv) $\circ$ *is a function from*

$$D = \{(f, g) \mid f, g \in \text{Mor } \textit{and } \text{dom}(f) = \text{cod}(g)\}$$

into Mor*, called the composition law*[2] *of* $\mathcal{C}$*;*

such that the following conditions are satisfied:

1. Matching Condition: If $f \circ g$ *is defined, then* $\text{dom}(f \circ g) = \text{dom}(g)$ *and* $\text{cod}(f \circ g) = \text{cod}(f)$*;*

2. Associativity Condition: If $f \circ g$ *and* $h \circ f$ *are defined, then* $h \circ (f \circ g) = (h \circ f) \circ g$*;*

3. Identity Existence Condition: For each $\mathcal{C}$*-object* A *there exists a* $\mathcal{C}$*-morphism* e *such that* $\text{dom}(e) = A = \text{cod}(e)$ *and*

$$(a)\ \ f \circ e = f \ \textit{ whenever } \ f \circ e \ \textit{ is defined}$$

$$(b)\ \ e \circ g = g \ \textit{ whenever } \ e \circ g \ \textit{ is defined}$$

4. Smallness of Morphism Class Condition: For any pair (A, B) *of* $\mathcal{C}$*-object, the class*

$$hom_{\mathcal{C}}(A, B) = \{f \mid f \in \text{Mor}, \ \text{dom}(f) = A \ \textit{and } \text{cod}(f) = B\}$$

is a set.

Common examples of categories come from mathematics: the category of groups and group homomorphisms, the category of vector spaces and linear transformations, the category of topological spaces and continuous functions and so on. Let us mention now some categories that have a specific use in different areas of computer science:

[2] $\circ(f, g)$ *is usually written* $f \circ g$ *and we say that* $f \circ g$ *is defined if, and only if,* $(f, g) \in D$

Example 1.1 *A deductive system is a category where objects are formulas and morphisms are proofs. The composition corresponds to an inference rule. Identities are axioms which state that a proposition is provable by itself.*

Example 1.2 *In the category of sets (denoted by* `Set`*), objects are sets and morphisms are set theoretic functions. The composition is the usual composition of functions. Identities are the identity functions.*

Example 1.3 *Partially ordered sets form a category where objects are partially ordered sets and morphisms are monotone mappings. The composition is the composition of mappings. Identities are the identity functions.*

Example 1.4 *A poset (partially ordered set) form also a category where objects are its elements and there is exactly one morphism from an element x to an element y if and only if $x \leq y$. Composition is forced by transitivity. Identity is forced by reflexivity.*

1.1.2 Functors

One reasonable question could now be if there is a category of categories. The answer is yes. In this category of categories the objects are categories and the morphisms are certain structure-preserving maps between categories, called *functors*.

Definition 1.2 *Let $\mathcal{C}$ and $\mathcal{D}$ be categories. A functor (covariant functor) from $\mathcal{C}$ to $\mathcal{D}$ is a triple $(\mathcal{C}, F, \mathcal{D})$ where F is a function from the class of morphisms of $\mathcal{C}$ to the class of morphisms of $\mathcal{D}$, i.e., $F : \mathrm{Mor}(\mathcal{C}) \longrightarrow \mathrm{Mor}(\mathcal{D})$, satisfying the following conditions:*

(i) F preserves identities; i.e., if e is a $\mathcal{C}$-identity, then $F(e)$ is a $\mathcal{D}$- identity.

(ii) F preserves composition; $F(f \circ g) = F(f) \circ F(g)$: i.e., whenever $\mathrm{dom}(f) = \mathrm{cod}(g)$, then $\mathrm{dom}(F(f)) = \mathrm{cod}(F(g))$ and the above equality holds.

As we can see, the definition of covariant functor preserves also the direction of the morphisms in $\mathcal{C}$. Nevertheless, sometimes it is interesting to study transformations that reverse such directions. A transformation of this kind between two categories $\mathcal{C}$ and $\mathcal{D}$ may be considered as a functor from the *opposite category*[3], $\mathcal{C}^{op}$ to $\mathcal{D}$ and it is known as *contravariant functor*.

Examples of functors can be easily found in mathematics. Some examples are very clear, for instance, for any category $\mathcal{C}$, there is an identity functor $id : \mathcal{C} \longrightarrow \mathcal{C}$. Here follows some other examples:

Example 1.5 *If $\mathcal{G}$ and $\mathcal{H}$ are groups considered as categories with a single object, then a functor from $\mathcal{G}$ to $\mathcal{H}$ is exactly a group homomorphism.*

Example 1.6 *If $\mathcal{P}$ and $\mathcal{Q}$ are posets, a functor from $\mathcal{P}$ to $\mathcal{Q}$ is exactly a nondecreasing map.*

Example 1.7 *The list functor List* : $\mathtt{Set} \longrightarrow \mathtt{Set}$ *(*$\mathtt{Set}$ *denotes the category of sets) is defined by $List(A)$ being the set of finite lists with elements in A, i.e. $List(A) = \bigcup_{n\in\mathbf{N}} A^n$, and further for $f : A \longrightarrow B$ we have*

$$List\, f(L) = [f(a_1), \ldots, f(a_n)]$$

for finite lists $L = [a_1, \ldots, a_n]$ with $a_1, \ldots, a_n \in A$.

In the following two examples, two *set functors* are defined, e.g. functors from $\mathtt{Set}$ to $\mathtt{Set}$. These two functors are introduced here since they play an important role in our research and will be developed further in the next chapters.

Example 1.8 *Let L be a completely distributive lattice. For $L = \{0, 1\}$ we write $L = 2$. The covariant powerset functor L_{id} is obtained by $L_{id}X = L^X$, i.e. the set of*

[3]The *opposite category* $\mathcal{C}^{op}$ of $\mathcal{C}$, has the same objects and morphisms as $\mathcal{C}$ but $Hom_{\mathcal{C}}(A, B) = Hom_{\mathcal{C}^{op}}(B, A)$ for all objects $A, B \in \mathcal{C}$.

mappings (or L-fuzzy sets) $A : X \longrightarrow L$, *and following [28], for a morphism* $f\colon X \longrightarrow Y$ *in* `Set`, *by defining*

$$\begin{aligned} L_{id}f(A)(y) &= \bigvee_{x\in X} A(x) \wedge f^{-1}(\{y\})(x) \\ &= \bigvee_{f(x)=y} A(x). \end{aligned}$$

This definition extends the *crisp powerset functor*[4] (case $L = 2$) to the many-valued case.

Example 1.9 *Even though terms usually are defined inductively, we adopt here a more functorial presentation of the set of terms. A purely functorial presentation might seem complicated at the start but there are advantages when we come to monads. Specially significative are the simplifications that a functorial presentation offers to prove results concerning monad compositions. Notations here follow [27], which were adopted also in [13].*

For a set A, *the constant set functor* $A_{\mathtt{Set}}$ *is the covariant set functor which assigns sets* X *to* A, *and mappings* f *to the identity map* id_A. *The sum* $\sum_{i\in I}\varphi_i$ *of covariant set functors* φ_i *assigns to each set* X *the disjoint union* $\bigcup_{i\in I}(\{i\} \times \varphi_i X)$, *and to each morphism* $X \xrightarrow{f} Y$ *in* `Set` *the mapping* $(i, m) \mapsto (i, \varphi_i f(m))$, *where* $(i, m) \in (\sum_{i\in I}\varphi_i)X$.

Let k *be a cardinal number and* $(\Omega_n)_{n\leq k}$ *be a family of sets. We will write* $\Omega_n id^n$ *instead of* $(\Omega_n)_{\mathtt{Set}} \times id^n$. *Note that* $\sum_{n\leq k}\Omega_n id^n X$ *is the set of all triples* $(n, \omega, (x_i)_{i\leq n})$ *with* $n \leq k$, $\omega \in \Omega_n$ *and* $(x_i)_{i\leq n} \in X^n$.

A disjoint union $\Omega = \bigcup_{n\leq k}\{n\} \times \Omega_n$ *is an operator domain, and an* Ω*-algebra is a pair* $(X, (s_{n\omega})_{(n,\omega)\in\Omega})$ *where* $s_{n\omega} : X^n \longrightarrow X$ *are* n*-ary operations. The* $\sum_{n\leq k}\Omega_n id^n$*-morphisms between* Ω*-algebras are precisely the homomorphisms between the algebras.*

[4]Recall the usual definition for the *crisp powerset functor*: PX is the set of subsets of X, i.e. $PX = \{A \mid A \subseteq X\}$ and for each morphism $f : X \longrightarrow Y$, $Pf : PX \longrightarrow PY$ is defined by $Pf(A) = \{f(x) \mid x \in A\}$.

The term functor can now be defined by transfinite induction. In fact, let $T_\Omega^0 = id$ *and define*

$$T_\Omega^\alpha = (\sum_{n\leq k} \Omega_n id^n) \circ \bigcup_{\beta<\alpha} T_\Omega^\beta$$

for each positive ordinal α*. Finally, let*

$$T_\Omega = \bigcup_{\alpha<\bar{k}} T_\Omega^\alpha$$

where $\bar{k}$ *is the least cardinal greater than* k *and* $\aleph_0$*. Clearly,* $(n, \omega, (m_i)_{i\leq n}) \in T_\Omega^\alpha X$*,* $\alpha \neq 0$*, implies* $m_i \in T_\Omega^{\beta_i} X$*,* $\beta_i < \alpha$*.*

Note that $(T_\Omega X, (\sigma_{n\omega})_{(n,\omega)\in\Omega})$ *is an* Ω*-algebra, if for* $(n, \omega) \in \Omega$ *and* $m_i \in T_\Omega X$ *we define* $\sigma_{n\omega}((m_i)_{i\leq n}) = (n, \omega, (m_i)_{i\leq n})$*. Actually, this algebra is a freely generated algebra in the category of* Ω*-algebras, that is, for an* Ω*-algebra* $B = (Y, (t_{n\omega})_{(n,\omega)\in\Omega})$*, a morphism* $X \xrightarrow{f} Y$ *in* `Set` *can be extended by transfinite induction to a* Ω*-homomorphism*

$$(T_\Omega X, (\sigma_{n\omega})_{(n,\omega)\in\Omega}) \xrightarrow{f^\star} (Y, (t_{n\omega})_{(n,\omega)\in\Omega})$$

called the Ω*-extension of* f *associated to* B*, by*

$$\begin{aligned} f^\star_{|T_\Omega^0 X} &= f \qquad \text{for the base case, and} \\ f^\star(n, \omega, (m_i)_{i\leq n}) &= t_{n\omega}((f^\star(m_i))_{i\leq n}) \end{aligned}$$

for each positive ordinal α *satisfying* $\alpha < \bar{k}$ *and* $(n, \omega, (m_i)_{i\leq n}) \in T_\Omega^\alpha X$*.*

A morphism $X \xrightarrow{f} Y$ *in* `Set` *can also be extended to the corresponding* Ω*-homomorphism*

$$(T_\Omega X, (\sigma_{n\omega})_{(n,\omega)\in\Omega}) \xrightarrow{T_\Omega f} (T_\Omega Y, (\tau_{n\omega})_{(n,\omega)\in\Omega}),$$

where $T_\Omega f$ *is defined to be the* Ω*-extension of* $X \xrightarrow{f} Y \hookrightarrow T_\Omega Y$ *associated to* $(T_\Omega Y, (\tau_{n\omega})_{(n,\omega)\in\Omega})$*.*

In the next chapters, we will see how to extend the powerset functor and the term functor to more complex structures. In fact, these two functors are central for our research since, as we will see, they provide a way to approach the many-value case.

1.1.3 Natural Transformations

The concept of naturality is central in many of the applications of category theory. *Natural transformations* are certain structure-preserving maps from one functor to another.

Definition 1.3 *Let $F : \mathcal{A} \longrightarrow \mathcal{B}$ and $G : \mathcal{A} \longrightarrow \mathcal{B}$ be functors. A natural transformation from F to G is a triple (F, η, G) where $\eta : Ob(\mathcal{A}) \longrightarrow Mor(\mathcal{B})$ is a function satisfying the following conditions:*

(i) For each $X \in Ob(\mathcal{A})$, $\eta(X)$(usually denoted by η_X) is a $\mathcal{B}$-morphism $\eta_X : FX \longrightarrow GX$.

(ii) For each $\mathcal{A}$-morphism $f : X \longrightarrow Y$, the diagram

$$\begin{array}{ccc} X & & FX \xrightarrow{\eta_X} GX \\ {\scriptstyle f}\downarrow & \qquad & {\scriptstyle Ff}\downarrow \qquad\qquad \downarrow{\scriptstyle Gf} \\ Y & & FY \xrightarrow[\eta_Y]{} GY \end{array}$$

commutes, that is, $Gf \circ \eta_X = \eta_Y \circ Ff$.

Example 1.10 *For the covariant powerset functor L_{id}, the transformations given by, $\eta_X : X \to L_{id}X$*

$$\eta_X(x)(x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

and $\mu_X : L_{id}L_{id}X \to L_{id}X$ by

$$\mu_X(\mathcal{A})(x) = \bigvee_{A \in L_{id}X} A(x) \wedge \mathcal{A}(A). \tag{1.2}$$

are natural transformations.

Example 1.11 *For the term functor T_Ω, the transformations*

$$\eta_X^{T_\Omega}(x) = x \tag{1.3}$$

$$\mu_X^{T_\Omega} = id^\star_{T_\Omega X} \tag{1.4}$$

where $id^\star_{T_\Omega X}$ *is the* Ω*-extension of* $id_{T_\Omega X}$ *with respect to* $(T_\Omega X, (\sigma_{n\omega})_{(n,\omega)\in\Omega})$*, are natural transformations.*

The natural transformations in examples 1.10 and 1.11 provide the powerset functor and the term functor respectively with the *monad* structure. These results will be described in Chapter 2.

1.2 Motivations

Category theory is an abstract and general theory that allows us to identify and isolate basic constructions and therefore enable a more universal study.

A categorical approach to *unification* can abstract information and provide a useful tool within logic programming. For instance, the use of unification is central to solve equations. Unification algorithms are widely use in the literature and several algorithms for computing *most general unifiers* have been developed over the years, starting with Herbrand in 1930. In 1965 Robinson applied the unification algorithm to automated inference. After that several variations on the algorithm have been proposed to improve efficiency. In [56] a categorical version is presented where *coequalizers* represent most general unifiers.

For those not familiar with unification we start by giving an informal explanation of what it is and which is the role of category theory in this context. We describe how unification in classical logic can be seen as a provision of *coequalizers* in `Set`.

Based on a categorical approach, some generalisations to the many-valued case can be found at the end of the section.

1.2.1 Unification

Unification is the process of making two terms equal by suitably instantiating the variables in the terms. Pattern matching, type checking and theorem proving are

some areas where unification is used.

Let us give an informal example. Consider the terms $(1+x)$ and $(y+z)$. Unifying these terms means substituting terms for the variables x, y and z so that the terms become identical. One possibility, in this case, is to substitute z for x and 1 for y. After this substitution both terms become equal to $(1 + z)$. But that is not the only possibility. We could also substitute 1 for all of x, y and z, thus after the substitutions we get the term $(1 + 1)$. This latter substitution is however less general than the previous one.

Unification involves seeking a *'most general unifier'* for two given terms. In the example above, the first substitution is more general than the second one in the sense that one can consider a third substitution (substitute 1 for z) such that its composition with the first one yield the second one. So a most general unifier is such that any other unifier factors through it.

Of course, it is not always the case that a given pair of terms is unifiable. A trivial example is to consider the terms 0 and 1. Another example of terms that are not unifiable are for instance the pair of terms x and $x + 1$.

1.2.2 Substitutions

Substitutions play an important role in unification. Let us consider for instance the equation

$$p(a, x, f(g(y))) = p(z, f(z), f(u))$$

where p, f and g are operator symbols with arities $3, 1$ and 1 respectively, x, y, z and u are variables and a is a constant.

To solve the equation we need to find a substitution σ that makes both sides of the equation become the same term. This is the concept of *unifier*.

The substitution

$$\sigma = [z/a, x/f(a), u/g(y)]$$

is a solution of the equation since applying it to both sides of the equation we get the same term

$$p(a, f(a), f(g(y)))$$

In fact this substitution is not only a unifier, it is the most general unifier.

One important aspect about substitutions relates to their composition. Substitutions can be composed since they can also be applied to terms just replacing variables in terms by terms.

For instance, the substitutions σ_1 and σ_2 defined by

$$\sigma_1 = [x/f(y, g(a, z)), y/b, z/h(y, c)]$$

$$\sigma_2 = [x/a, y/g(z, f(b, z)), z/h(g(a, z), f(a, b))]$$

can be composed and its composition $\sigma_1 \star \sigma_2$ is given by

$$\sigma_1 \star \sigma_2 = [x/a, y/g(\sigma_1(z), f(b, \sigma_1(z))), z/h(g(a, \sigma_1(z)), f(a, b))]$$

1.2.3 Categorical Aspects of Classical Unification

Example 1.9 shows how the set of terms can be categorically represented by defining the term functor T_Ω. The set of terms with variables from a set X over the operator domain Ω can then be represented by $T_\Omega X$.

With this terminology, a substitution σ can now be seen as a mapping, $\sigma\colon X \longrightarrow T_\Omega Y$ from the set of variables X to the set of terms $T_\Omega Y$ with variables in the set Y.

Applying σ to a term t (in logic programming usually written as $t\sigma$) means replacing each variable x in t by $\sigma(x)$. With this notation, a unifier of two terms s, t is a substitution σ such that $s\sigma = t\sigma$. The main idea of considering a most general unifier (mgu) is to consider the substitution (if it exists) that does 'as little as possible'. A substitution σ is more general than τ if $\tau = \sigma\bar{\tau}$ for some $\bar{\tau}$ and a unifier of s and t is a mgu if it is more general than all other unifiers.

We mentioned before how substitutions may be composed by replacing variables in terms by terms. Given the substitutions $\sigma_1 \colon X \longrightarrow T_\Omega Y$ and $\sigma_2 \colon Y \longrightarrow T_\Omega Z$, a remark here is that their composition cannot be the usual mapping composition, but there is indeed a 'standard' way of composing them. Naturality properties associated to the term functor make it possible to define the composition of substitution as the following sequence of mappings:

$$X \xrightarrow{\sigma_1} T_\Omega Y \xrightarrow{T_\Omega \sigma_2} T_\Omega T_\Omega Z \xrightarrow{\mu_Z} T_\Omega Z$$

Note here that the 'flattening' operator μ_Z is one of the natural transformations defined in Example 1.11. In this case, since the term functor is idempotent, the transformation is the identity. This correspond with the classical definition 'terms over terms are terms', i.e. $T_\Omega T_\Omega = T_\Omega$.

At this point the discussion is incomplete, since we have still not introduced some of the concepts that appears here. Later, in Chapter 2, we will see that the previous composition is nothing but the composition in the *Kleisli category* associated with the *term monad*.

From a categorical point of view, unification can be considered as an instance of something more abstract: a *colimit* in a suitable category. Goguen made the observation that the concepts of coequalizers and most general unifiers are equivalent [29], [56]. A unification algorithm using coequalizers in the language ML can be found in [57].

For further information about how general constructions of *colimits* capture the compositional structure of unification algorithms, see [58].

1.2.4 The Many-Valued Case

We could now attempt a *'generalisation of terms'* in the sense that a generalised term will be a many-valued set of terms.

Consider the following situation: When solving a functional equation we might apply different iterative methods and therefore different approximations are obtained.

The different approximations obtained to the actual solution f can be classified in classes of functions $[f]_{\epsilon_i}$ according to an increasing numerable sequence of bounds $\epsilon_1, \epsilon_2, \ldots, \epsilon_n < 1$ and the following measure: a function g is in the class $[f]_{\epsilon_i}$ if and only if $|f(x) - g(x)| < \epsilon_i$ and $g \notin [f]_{\epsilon_j}$ for all $j < i$. Let us designate a canonical representative g_{ϵ_i} for each class $[g]_{\epsilon_i}$.

Let us now consider the (crisp) sets of terms M_1 and M_2,

$$M_1 = \{f(a), g_\epsilon(x)\}$$

$$M_2 = \{f(x), g_\epsilon(a)\}$$

We might want to compare somehow the previous sets and get a criteria about whether it is possible to 'unify' them. Note here that such a comparison should include not only a comparison between the former terms but also will allow a comparison between f and its approximation g_ϵ.

In this case, a substitution will replace the variable x by a set of terms, e.g. a substitution here is a mapping $\theta\colon X \longrightarrow\!\!\!\!\!\rightarrow PT_\Omega Y$, where P denotes the powerset functor.

To apply such a *'generalised substitution'* to a set of terms means to replace each occurrence of the term containing the variable by new occurrences of the terms in the new variables given by the substitution. For instant, the substitution that replaces the variable x by the set consisting in the constant a and the variable y, i.e, $\theta_1(x) = \{a, y\}$ can be applied to the set M_1 leading to the set of terms $\{f(a), g_\epsilon(a), g_\epsilon(y)\}$.

Consider now the generalised substitution $\theta_2(x) = \{a\}$. Applying θ_2 to M_1 and M_2 we obtain the same set of terms, $\{f(a), g_\epsilon(a)\}$. Actually, in this case θ_1 is a unifier of the two set of terms.

In general we could consider generalised substitutions that replace variables by many-valued set of terms, i.e, $\theta\colon X \longrightarrow\!\!\!\!\!\rightarrow LT_\Omega Y$, where L denotes the powerset functor (see Example 1.8).

Similarity relations will be needed to compare many-valued sets of terms, as a

step towards many-valued unification. Similarities between powersets of terms are described in Chapter 5.

Composition of Generalised Substitutions

As in the classical case, the naturality properties together with the monad structure, are the key concepts that will allow us to compose generalised substitution. In short, given the generalised substitutions $\theta_1 \colon X \longrightarrow LT_\Omega Y$ and $\theta_2 \colon Y \longrightarrow LT_\Omega Z$, its composition can be described by the following sequences of mappings:

$$X \xrightarrow{\theta_1} LT_\Omega Y \xrightarrow{LT_\Omega \theta_2} LT_\Omega LT_\Omega Z \xrightarrow{\mu_Z^{LT_\Omega}} T_\Omega Z$$

One of the important achievements of this thesis is to show that the functor composition LT_Ω can be extended to a monad and therefore, $\mu_Z^{LT_\Omega}$ can be considered. This composition is again the composition in the Kleisli category associated to the monad $\mathbf{LT_\Omega}$.

We can now ask ourselves the following questions: Can we use categorical methods to approach the area of many-valued unification? Can a similar approach as in classical logic provide us with some answers?

1.3 Contributions

Figure 1.1 can be seen as a summary of our research goal: Unification in classical logic can be seen as the provision of coequalizers by making use of the term monad as described before. We are now in the process of investigating a possible extension of unification into the many-valued case by following a similar approach.

As a first step, a concept for *generalised terms* was studied. As mentioned before a generalised term will be a composition of monads that again yields a monad, i.e. compositions of powerset monads with the term monad provide definitions for generalised terms [23], [20], [17].

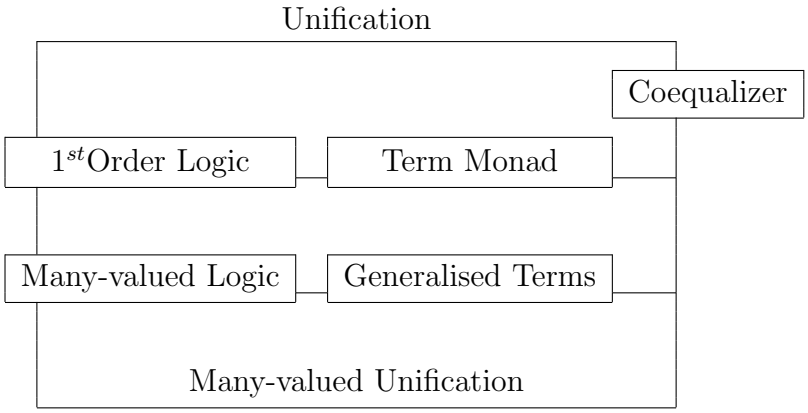


Figure 1.1: Summary of the Research Goal

A composition of monads does, however, not always produce a monad. In this sense, techniques for monads composition provide a helpful tool for our concerns [14], [15].

The composition of monads make use of a lot of equations. Proofs become complicated, not to mention the challenge of understanding different steps on the equations. In this respect, the use of a visual techniques can provide the support we need [16], [21].

While an abstract study related to monads certainly is interesting, it can also be risky in the sense that you could forget your main goals and get into a never-ending exploration of the structures. It is difficult to find a limitation of what we really need for a particular goal and what we are interested in exploring. It is time for changing our thoughts and retaking again our main goals. For the purpose of many-valued unification, similarity relations are now one of the next steps and they need to be defined for generalised terms [22], [19], [18].

Chapter 2

Monads

As remarked in [6], the naming and identification of monads, in particular as associated with *adjoints*, can be seen initiated around 1958. Godement was at that time one of the very first authors to use monads, even if then only named '*standard constructions*'. Huber in 1961 shows that adjoint pairs give rise to monads. Kleisli [41] and Eilenberg and Moore [11] proved the converse in 1965. The construct of a Kleisli category was thus made explicit in those contributions. Lawvere [43] introduced universal algebra into category theory. This can be seen as the birth of the term monad. These developments then contain all categorical elements for substitution theories. The exploitation of terms and unifications thereof within logic programming is formally described in [55] as early as in 1965. It is therefore somewhat surprising that the categorical connection to unification was not found until twenty years later by Rydeheard and Burstall in [56].

With respect to Computer Science, the applications and use of monads and category theory in general, provide an abstract tool to handle properties of structures. Using a category-theoretic methodology, the study of programming languages could avoid irrelevant syntactic details and focus on the important structures underlying programming languages as showed in [53], where a categorical semantics of computations based on monads is developed.

The impact of monads on functional programming is well known and monads have been recognised to be an important concept in many applications of functional programming. In pure functional programming languages such as Haskell, monads are data types that encapsulate the functional I/O-activity (input/output), in such a manner that the side-effects of I/O are not allowed to spread out of the part of the program that is not functional.

Monads can also be used to give a semantics to term rewriting systems by generalising the well-known equivalence between universal algebra and monads on `Set`, [45].

It is worth to mention that monads are also an important tool in topology when handling regularity, iteratedness and compactifications, and also in the study of toposes and related topics.

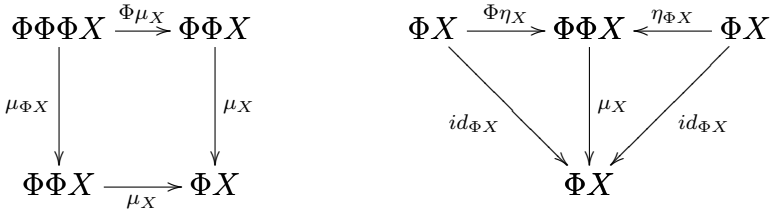
In this chapter we study some examples of monads and, more generally, the concept of submonad as a tool for obtaining new monads. We remark the non-triviality of extending a functor to a monad. Techniques for composing monads are presented at the end of the chapter.

2.1 Monads, Triples, Algebraic Theories

Monads (also known as triples or algebraic theories) consist of a functor that has associated two particular natural transformations called *unit* and *multiplication*.

Definition 2.1 *Let* $\mathtt{C}$ *be a category. A monad (or triple, or algebraic theory) over* $\mathtt{C}$ *is written as* $\mathbf{\Phi} = (\Phi, \eta, \mu)$*, where* $\Phi : \mathtt{C} \longrightarrow \mathtt{C}$ *is a (covariant) functor, and* $\eta : id \longrightarrow \Phi$ *and* $\mu : \Phi \circ \Phi \longrightarrow \Phi$ *are natural transformations for which the following*

diagrams



commute,i.e. $\mu \circ \Phi\mu = \mu \circ \mu\Phi$ *and* $\mu \circ \Phi\eta = \mu \circ \eta\Phi = id_\Phi$ *hold.*

A monad defined in this way is said to be in *monoid form*.

For a natural transformation ξ, we will write $(\xi\Phi)_X = \xi_{\Phi X}$ and $(\Phi\xi)_X = \Phi\xi_X$. It is convenient to write η^Φ and μ^Φ if we need to distinguish between natural transformations in different monads.

In the following subsections some examples of monads are presented. These examples will be further studied in next chapter, when the attention is devoted to the generalisation of terms.

2.1.1 The Powerset Monad

The powerset functor L_{id} defined in Example 1.8 can be made a monad with the natural transformations showed in Example 1.10.

Proposition 2.1 ([46]) $\mathbf{L}_{id} = (L_{id}, \eta, \mu)$ *is a monad, where* $\eta_X : X \longrightarrow L_{id}X$ *is given by*

$$\eta_X(x)(x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$

and $\mu_X \colon L_{id}L_{id}X \longrightarrow L_{id}X$ *given by*

$$\mu_X(\mathcal{A})(x) = \bigvee_{A \in L_{id}X} A(x) \wedge \mathcal{A}(A).$$

For $L = 2$, the two element lattice, we obtain the crisp powerset monad (P, η, μ), where $\eta_X(x) = \{x\}$ and $\mu_X(\mathcal{A}) = \bigcup \mathcal{A}$.

2.1.2 The Term Monad

The term functor (see Example 1.9) can also be made a monad by considering the natural transformations in Example 1.11.

Proposition 2.2 ([46]) $\mathbf{T}_\Omega = (T_\Omega, \eta^{T_\Omega}, \mu^{T_\Omega})$ *is a monad, where* $\eta_X : X \longrightarrow T_\Omega X$ *is given by*

$$\eta_X^{T_\Omega}(x) = x$$

and $\mu_X : T_\Omega T_\Omega X \longrightarrow T_\Omega X$ *given by*

$$\mu_X^{T_\Omega} = id^\star_{T_\Omega X}$$

with $id^\star_{T_\Omega X}$ *being the* Ω*-extension of* $id_{T_\Omega X}$ *with respect to* $(T_\Omega X, (\sigma_{n\omega})_{(n,\omega)\in\Omega})$*.*

2.1.3 Extension Principles for Fuzzy Sets

Fuzzy set theory is founded on the idea that many nonmathematical properties cannot be adequately described in terms of crisp sets comprising those elements that fulfil a given property. Often the notion of membership is considered as a gradual property for fuzzy sets.

In [33] some constructions of fuzzy set categories were introduced. In Paper I we generalised these results by considering L-fuzzy sets in which the values of the characteristic functions run on a completely distributive lattice, rather than in the unit real interval. These L-fuzzy sets are used to define the L-fuzzy categories. L-fuzzy functors given by the *extension principles* are shown to have a monadic structure.

Our approach to the *fuzzification* of a set of terms will be by considering a 'fuzzy set' of terms. Therefore the different generalizations of the powerset functor become important, specially, if the composition of these fuzzy powerset functors with the term monad can be extended to a monad.

Given $X, Y \in \mathtt{Set}$ and a mapping $f : X \longrightarrow Y$, it is possible to define a mapping between the corresponding powersets $\hat{f} : PX \longrightarrow PY$ by means of the direct image of f, that is, given $A \in P(X)$ then $\hat{f}(A) = f(A) \in P(Y)$.

When working in the fuzzy case, the extension of f given above admits different generalisations according to the optimistic or pessimistic interpretation of the fuzziness degree. These generalisations give rise to the following extension principles:

1. Maximal extension principle: $\hat{f}_M\colon \mathcal{F}(X) \longrightarrow \mathcal{F}(Y)$ is defined such that given $A \in \mathcal{F}(X)$ then $\hat{f}_M(A)(y) = \sup\{A(x) \mid x \in f^{-1}(y) \text{ and } A(x) > 0\}$ if the set is nonempty and $\hat{f}_M(A)(y) = 0$ otherwise.

2. Minimal extension principle: $\hat{f}_m\colon \mathcal{F}(X) \longrightarrow \mathcal{F}(Y)$ is defined such that given $A \in \mathcal{F}(X)$ then $\hat{f}_m(A)(y) = \inf\{A(x) \mid x \in f^{-1}(y) \text{ and } A(x) > 0\}$ if the set is nonempty and $\hat{f}_m(A)(y) = 0$ otherwise.

Both extensions $\hat{f}_M$ and $\hat{f}_m$ coincide with the direct image extension in the case of crisp subsets, that is, given $A \in P(X) \subset \mathcal{F}(X)$, then $\hat{f}_M(A) = \hat{f}_m(A) = f(A) \in P(Y) \subset \mathcal{F}(Y)$.

The maximal and minimal extension principles just introduced can be further generalized to the L-fuzzy power sets, by simply changing the calculations of suprema and infima to the lattice join and meet operators. In the following, we will use the set $I = \{x \in X \mid x \in f^{-1}(y) \text{ and } A(x) > 0\}$:

1. Maximal L-fuzzy extension principle: $\widetilde{f}_M\colon \mathcal{L}(X) \longrightarrow \mathcal{L}(Y)$ is defined in such a way that given $A \in \mathcal{L}(X)$ then $\widetilde{f}_M(A)(y) = \bigvee_I A(x)$ if the set I is nonempty and $\widetilde{f}_M(A)(y) = 0$ otherwise.

2. Minimal L-fuzzy extension principle: $\widetilde{f}_m\colon \mathcal{L}(X) \longrightarrow \mathcal{L}(Y)$ is defined in such a way that given $A \in \mathcal{L}(X)$ then $\widetilde{f}_m(A)(y) = \bigwedge_I A(x)$ if the set I is nonempty and $\widetilde{f}_m(A)(y) = 0$ otherwise.

The extension principles just stated suggest the possibility of extending the definition of $\mathcal{L}$ to be a functor between classical sets and L-fuzzy sets but, obviously, the first step should be to define the appropriate concept of L-fuzzy set *category*. The natural way to build a categorical structure on the classes of L-fuzzy sets is to

consider the arrows between $\mathcal{L}(X)$ and $\mathcal{L}(Y)$ as those given by any of the extension principles introduced above.

Definition 2.2 *For all $\alpha \in L$, the classes of the α-upper L-fuzzy sets and the α-lower L-fuzzy sets, denoted $\mathcal{L}_\alpha(X)$ and $\mathcal{L}^\alpha(X)$ respectively, are defined as follows:*

$$\begin{aligned} \mathcal{L}_\alpha(X) &= \{A \mid A \in \mathcal{L}(X), A(x) \geq \alpha \text{ or } A(x) = 0, \text{ for all } x \in X\} \\ \mathcal{L}^\alpha(X) &= \{A \mid A \in \mathcal{L}(X), A(x) \leq \alpha \text{ or } A(x) = 1, \text{ for all } x \in X\} \end{aligned}$$

In Paper I the categories $\mathcal{L}_\alpha$-`Set` and $\mathcal{L}^\alpha$-`Set` were formally defined. We also showed that those categories are equivalent to the category `Set`, i.e. the extended categories are essentially the category of sets.

In the particular case of the functor $\mathcal{L} = L_{id}$, as we mentioned before in Section 2.1.1, (L_{id}, η, μ) is a monad. The generalization of this result can be easily shifted to the case of $(\mathcal{L}^\alpha, \eta^\alpha, \mu^\alpha)$.

Also $(\mathcal{L}_\alpha, \eta_\alpha, \mu_\alpha)$ can be made a monad where $\eta_{\alpha X}\colon X \longrightarrow \mathcal{L}_\alpha X$ is defined by

$$\eta_{\alpha X}(x)(x') = \begin{cases} 1 & \text{if } x = x' \\ \\ 0 & \text{otherwise} \end{cases}$$

and $\mu_{\alpha X}\colon \mathcal{L}_\alpha\mathcal{L}_\alpha X \longrightarrow \mathcal{L}_\alpha X$ is defined by

$$\mu_{\alpha X}(\mathcal{A})(x) = \begin{cases} \bigwedge\limits_{A \in I} A(x) \wedge \mathcal{A}(A) & \text{if } I = \{A \in \mathcal{L}_\alpha X \mid A(x) \wedge \mathcal{A}(A) > 0\} \neq \varnothing \\ \\ 0 & \text{otherwise} \end{cases}$$

2.1.4 Submonads

The concept of *subfunctors* and *submonads* can be used to provide a technique for constructing new monads from given ones. In Section 2.3 we will study some properties of these constructions and in Chapter 3 we will see how these results can be used to provide more examples on monad compositions.

Definition 2.3 *Let Φ be a set functor. A set functor Φ' is a subfunctor of Φ, written $\Phi' \leq \Phi$, if there exists a natural transformation $e\colon \Phi' \longrightarrow \Phi$, called the inclusion transformation, such that $e_X\colon \Phi'X \longrightarrow \Phi X$ are inclusion maps, i.e., $\Phi'X \subseteq \Phi X$. The conditions on the subfunctor imply that $\Phi f \mid_{\Phi' X} = \Phi' f$ for all mappings $f : X \longrightarrow Y$. Further, $\leq$ is a partial ordering.*

Proposition 2.3 *Let $\mathbf{\Phi} = (\Phi, \eta, \mu)$ be a monad over* `Set`*, and consider a subfunctor Φ' of Φ, with the corresponding inclusion transformation $e : \Phi' \longrightarrow \Phi$, together with natural transformations $\eta' : id \longrightarrow \Phi'$ and $\mu' : \Phi'\Phi' \longrightarrow \Phi'$ satisfying the conditions*

$$e \circ \eta' = \eta, \tag{2.1}$$

$$e \circ \mu' = \mu \circ \Phi e \circ e\Phi'. \tag{2.2}$$

Then $\mathbf{\Phi}' = (\Phi', \eta', \mu')$ is a monad, called the submonad of $\mathbf{\Phi}$, written $\mathbf{\Phi}' \preceq \mathbf{\Phi}$.

Proof

Let $\mathbf{\Phi}' = (\Phi', \eta', \mu')$ be a submonad of the monad $\mathbf{\Phi} = (\Phi, \eta, \mu)$, $\mathbf{\Phi}' \preceq \mathbf{\Phi}$. Let us verify that $\mathbf{\Phi}'$ indeed is a monad. Firstly,

$$\begin{aligned} e \circ \mu' \circ \eta'\Phi' &= \mu \circ \Phi e \circ e\Phi' \circ \eta'\Phi' = \mu \circ \Phi e \circ (e \circ \eta')\Phi' \\ &\overset{(2.1)}{=} \mu \circ \Phi e \circ \eta\Phi' = \mu \circ \eta\Phi \circ e \\ &= e, \end{aligned}$$

which shows, $\mu'_X \circ \eta'_{\Phi' X} = \mathrm{id}_{\Phi' X}$. Secondly,

$$\begin{aligned} e \circ \mu' \circ \Phi'\eta' &= \mu \circ \Phi e \circ e\Phi' \circ \Phi'\eta' = \mu \circ e\Phi \circ \Phi' e \circ \Phi'\eta' = \mu \circ e\Phi \circ \Phi'(e \circ \eta') \\ &\overset{(2.1)}{=} \mu \circ e\Phi \circ \Phi'\eta = \mu \circ \Phi\eta \circ e \\ &= e, \end{aligned}$$

i.e. $\mu'_X \circ \Phi'\eta'_X = \mathrm{id}_{\Phi' X}$. Finally,

$$\begin{aligned} e \circ \mu' \circ \Phi'\mu' &\overset{(2.2)}{=} \mu \circ \Phi e \circ e\Phi' \circ \Phi'\mu' = \mu \circ \Phi(e \circ \mu') \circ e\Phi'\Phi' \\ &\overset{(2.2)}{=} \mu \circ \Phi\mu \circ \Phi\Phi e \circ \Phi e\Phi' \circ e\Phi'\Phi' = \mu \circ \mu\Phi \circ \Phi\Phi e \circ e\Phi\Phi' \circ \Phi' e\Phi' \\ &= \mu \circ \Phi e \circ \mu\Phi' \circ e\Phi\Phi' \circ \Phi' e\Phi' = \mu \circ \Phi e \circ (e \circ \mu')\Phi' \\ &= e \circ \mu' \circ \mu'\Phi', \end{aligned}$$

and therefore, $\mu'_X \circ \Phi'\mu'_X = \mu'_X \circ \mu'_{\Phi' X}$. ■

Proposition 2.4 *$\preceq$ is a partial ordering.*

Proof

Reflexivity is obvious.

For antisymmetry, if $\mathbf{\Phi}' \preceq \mathbf{\Phi}$ and $\mathbf{\Phi} \preceq \mathbf{\Phi}'$, then $\Phi = \Phi'$, and corresponding inclusion transformations are identities, and in fact equal. Therefore, it follows that $\eta = \eta'$ and $\mu = \mu'$. Thus, $\mathbf{\Phi}' = \mathbf{\Phi}$.

For transitivity, let $\mathbf{\Phi}'' \preceq \mathbf{\Phi}'$ and $\mathbf{\Phi}' \preceq \mathbf{\Phi}$. Consider $e^* = e^{\Phi} \circ e^{\Phi'} \colon \Phi'' \longrightarrow \Phi$, where $e^{\Phi} \colon \Phi' \longrightarrow \Phi$, and $e^{\Phi'} \colon \Phi'' \longrightarrow \Phi'$ are the transformations given by the subfunctor conditions. Then, $\Phi'' \leq \Phi$, with e^* as the inclusion transformation. Obviously, $e^* \circ \eta'' = \eta$. Further, because of naturality of e^{Φ}, we have $e^{\Phi}\Phi' \circ \Phi' e^{\Phi'} = \Phi e^{\Phi'} \circ e^{\Phi}\Phi''$, and therefore

$$\begin{aligned} e^* \circ \mu'' &= \mu \circ \Phi e^{\Phi} \circ e^{\Phi}\Phi' \circ \Phi' e^{\Phi'} \circ e^{\Phi'}\Phi'' \\ &= \mu \circ \Phi e^{\Phi} \circ \Phi e^{\Phi'} \circ e^{\Phi}\Phi'' \circ e^{\Phi'}\Phi'' \\ &= \mu \circ \Phi e^* \circ e^*\Phi''. \end{aligned}$$

Thus $\mathbf{\Phi}'' \preceq \mathbf{\Phi}$. ■

Example 2.1 *Let K and L be completely distributive lattices. Assume K to be a sublattice of L, with $\iota : K \longrightarrow L$ being the inclusion homomorphism. Further, assume $\iota(0) = 0$ and $\iota(1) = 1$, and additionally, that $\iota(\vee_i x_i) = \vee_i \iota(x_i)$ also in the non-finite case.*

Define $(\iota_{id})_X : K_{id}X \longrightarrow L_{id}X$ by $(\iota_{id})_X(A) = \iota \circ A$, $A : X \longrightarrow K$. It is easily checked that $\iota_{id} : K_{id} \longrightarrow L_{id}$ becomes a natural transformation, and that $\mathbf{K}_{id}$ is a submonad of $\mathbf{L}_{id}$.

Example 2.2 *In Section 2.1.3 we defined functors for α-upper L-fuzzy sets and α-lower L-fuzzy sets, denoted $\mathcal{L}_\alpha$ and $\mathcal{L}^\alpha$, respectively, given as follows:*

$$\begin{aligned} \mathcal{L}_\alpha X &= \{A \in L_{id}X \mid A \in \mathcal{L}(X), A(x) \geq \alpha \text{ or } A(x) = 0, \text{ for all } x \in X\}, \\ \mathcal{L}^\alpha X &= \{A \in L_{id}X \mid A(x) \leq \alpha \text{ or } A(x) = 1, \text{ for all } x \in X\}. \end{aligned}$$

For mappings $f : X \longrightarrow Y$, we could obtain $\mathcal{L}_\alpha f$ and $\mathcal{L}^\alpha f$, respectively, as $L_{id} f \mid_{\mathcal{L}_\alpha X}$ and $L_{id} f \mid_{\mathcal{L}^\alpha X}$. Thus, $\mathcal{L}_\alpha$ and $\mathcal{L}^\alpha$ become subfunctors of L_{id}.

Further, $\mathcal{L}_\alpha$ and $\mathcal{L}^\alpha$ were extended to monads with $\eta^{\mathcal{L}_\alpha}$ and $\eta^{\mathcal{L}^\alpha}$ defined using Equation (1.1), and additionally $\mu^{\mathcal{L}_\alpha}$ and $\mu^{\mathcal{L}^\alpha}$ defined using Equation (1.2). It is now not difficult to show that $(\mathcal{L}_\alpha, \eta^{\mathcal{L}_\alpha}, \mu^{\mathcal{L}_\alpha})$ and $(\mathcal{L}^\alpha, \eta^{\mathcal{L}^\alpha}, \mu^{\mathcal{L}^\alpha})$ are submonads of $\mathbf{L}_{id}$.

Example 2.3 *Let Ω' and Ω be operator domains with $\Omega' \subseteq \Omega$, and let $\epsilon : \Omega' \longrightarrow \Omega$ be the inclusion mapping.*

Define $\nu_X : T_{\Omega'} X \longrightarrow T_\Omega X$ by $\nu_X(x) = x$, $x \in X$, and $\nu_X((n, \omega', (t'_i)_{i \leq n})) = (n, \epsilon(\omega'), (\nu_X(t'_i))_{i \leq n})$ for $t'_i \in T_{\Omega'} X$. It is easily seen that $\nu : T_{\Omega'} \longrightarrow T_\Omega X$ is a natural transformation, in fact an inclusion, and that $\mathbf{T}_{\Omega'}$ is a submonad of $\mathbf{T}_\Omega$.

2.1.5 The Covariant Double Contravariant Powerset Monad

The contravariant powerset functor L^{id} is the contravariant hom-functor related to L, i.e. $L^{id} = hom(-, L) : \mathtt{Set} \longrightarrow \mathtt{Set}$, which to each set X and mapping $f : X \longrightarrow Y$ assigns the set L^X of all mappings of X into L, and the mappings $hom(f, L)(g) = g \circ f$ $(g \in L^Y)$, respectively. Note that 2^{id} is the usual contravariant powerset functor, where $2^{id} X = PX$, and morphisms $X \xrightarrow{f} Y$ in $\mathtt{Set}$ are mapped to $2^{id} f$ representing the mapping $M \mapsto f^{-1}[M]$ $(M \in PY)$ from PY to PX.

For double powerset functors it is convenient to write $L_{L_{id}} = L_{id} \circ L_{id}$ and $L^{L^{id}} = L^{id} \circ L^{id}$. Note that $L^{L^{id}}$ is a covariant functor. It may be interesting also to note that the *filter*[1] functor is a subfunctor of $2^{2^{id}}$, but not a subfunctor of $2_{2_{id}}$.

In the case of $L^{L^{id}}$, for $X \xrightarrow{f} Y$ in $\mathtt{Set}$ and $\mathcal{M} \in L^{L^X}$, we have $L^{L^{id}} f(\mathcal{M}) = \mathcal{M} \circ L^{id} f$, and hence, $L^{L^{id}} f(\mathcal{M})(g) = \mathcal{M}(g \circ f)$.

Proposition 2.5 ([13]) *The covariant set functor $LL = L^{id} \circ L^{id}$ can be extended to a monad, considering the following definitions of the natural transformations η^{LL}*

[1] A *filter* on a set X is a nonempty set $\mathcal{F}$ of subsets of X such that: (i) $\emptyset \notin \mathcal{F}$, (ii) $A, B \in \mathcal{F} \Rightarrow A \cap B \in \mathcal{F}$, (iii) $A \in \mathcal{F}$ $A \subseteq B \Rightarrow B \in \mathcal{F}$.

and μ^{LL}*:*

$$\eta^{LL}_X(x)(A) = A(x), \qquad \mu^{LL}_X(\mathcal{U}) = \mathcal{U} \circ \eta^{LL}_{LX}.$$

It is well-known that the proper[2] filter functor F_0 becomes a monad where $\eta^{F_0} : id \longrightarrow F_0$ is the unique natural transformation and $\mu^{F_0} : F_0 \circ F_0 \longrightarrow F_0$ is given by

$$\mu^{F_0}_X(\mathcal{U}) = \bigcup_{R\in\mathcal{U}} \bigcap_{\mathcal{M}\in R} \mathcal{M}$$

i.e. the contraction mapping suggested in [42].
In relation with the functor $2^{2^{id}}$, it can easily be seen that $\mu^{2^{2^{id}}}_X(\mathcal{U}) = \mu^{F_0}_X(\mathcal{U})$.

2.2 Kleisli Category of a Monad

Given a monad, there are two standard ways of defining a category associated to it; the *Eilenberg-Moore* construction and the *Kleisli* construction. In this section we concentrate on the latter one.

Definition 2.4 *A Kleisli category* $\mathtt{C}_{\mathbf{\Phi}}$ *for a monad* $\mathbf{\Phi}$ *over a category* $\mathtt{C}$ *is defined as follows: Objects in* $\mathtt{C}_{\mathbf{\Phi}}$ *are the same as in* $\mathtt{C}$*, and the morphisms are defined as* $hom_{\mathtt{C}_{\mathbf{\Phi}}}(X, Y) = hom_{\mathtt{C}}(X, \Phi Y)$*, that is morphisms* $f\colon X \rightharpoonup Y$ *in* $\mathtt{C}_{\mathbf{\Phi}}$ *are simply morphisms* $f\colon X \longrightarrow \Phi Y$ *in* $\mathtt{C}$*, with* $\eta^{\Phi}_X\colon X \longrightarrow \Phi X$ *being the identity morphism.*

Composition of morphisms is defined as

$$(X \stackrel{f}{\rightharpoonup} Y) \circ (Y \stackrel{g}{\rightharpoonup} Z) = X \stackrel{\mu^{\Phi}_Z \circ \Phi g \circ f}{\longrightarrow} \Phi Z.$$

The Kleisli category is equivalent to the full subcategory of free Φ-algebras of the monad, and its definition makes it clear that the arrows are substitutions. Indeed, the categorical unification algorithm in [56] is based on the Kleisli category of the term monad.

A monad (Φ, η, μ) written as $(\Phi, \eta, \circ)$, where $\circ$ is the composition of morphisms in the corresponding Kleisli category, is said to be a monad in *clone form*. In fact,

[2] $F_0X = FX \setminus \{\emptyset\}$

there is a one-to-one correspondence between monads, respectively, in monoid and clone forms [46].

For historical remarks let us mention the important role that Kleisli and Eilenberg-Moore categories play in the study of the relation between adjunctions and monads [6]. Every adjunction defines a monad, and conversely every monad can be seen as generated by an adjunction, called a *resolution* for the monad. Eilenberg-Moore and Kleisli categories give rise to resolutions, e.g., they are respectively the terminal and initial objects in the category of all resolutions.

Example 2.4 *The Kleisli category of the powerset monad (for $L = 2$) is the category of sets and relations,* `SetRel`*. This is a consequence of the fact that given a relation $R \subseteq X \times Y$, we can define the morphism $\varphi(R) : X \longrightarrow PY$ where*

$$\varphi(R)(x) = \{y \in Y \mid (x, y) \in R\}.$$

On the other hand, given the morphism $f : X \longrightarrow PY$, since $f(x) \subseteq Y$, we can define the relation $R_f \subseteq X \times Y$ given by

$$R_f = \{(x, y) \in X \times Y \mid y \in f(x)\}.$$

It is trivial to see that $\varphi(R_f) = f$ and $R_{\varphi(R)} = R$.

2.2.1 The Classical Case

For our research purposes we are interested in checking how Kleisli categories provide a way for composing substitution in classical logic and how this technique could be extended for many-valued unification.

In Section 1.2.3 we presented in an informal way how the composition of the substitutions $\sigma_1 \colon X \longrightarrow T_\Omega Y$ and $\sigma_2 \colon Y \longrightarrow T_\Omega Z$ in classical logic were defined as the following sequence of mappings:

$$X \xrightarrow{\sigma_1} T_\Omega Y \xrightarrow{T_\Omega \sigma_2} T_\Omega T_\Omega Z \xrightarrow{\mu_Z} TZ$$

Now we are ready to see that this sequence of mappings is nothing but the composition in the Kleisli category associated to the term monad, i.e. the category where the morphisms are variable substitutions.

2.3 Monad Compositions

This section is devoted to the general study of composition of monads: Given monads $\mathbf{\Phi}$ and $\mathbf{\Psi}$, let us consider the composition of the corresponding functors, $\Phi \circ \Psi$: Is it possible to extend this composition to a monad? Is this extension compatible with the monad structure in the initial monads? Does there exist a natural definition of the unit and multiplication in the composed functor which uses the units and multiplication in $\mathbf{\Phi}$ and $\mathbf{\Psi}$? Is this extension, in some sense, unique?

The problem of extending a functor to a monad is not a trivial one, and some strange situations may well arise as shown below. Note that the id^2 functor can be extended to a monad with $\eta_X(x) = (x, x)$ and $\mu_X((x_1, x_2), (x_3, x_4)) = (x_1, x_4)$. Similarly, id^n can be extended to a monad. In addition, the proper[3] powerset functor P_0, as well as $id^2 \circ P_0$ can, respectively, be extended to a monad in a unique way. However, $P_0 \circ id^2$ cannot be made to a monad [13].

2.3.1 General Techniques for Composing Monads

One of the major problems that arises when composing two monads $(\Phi, \eta^\Phi, \mu^\Phi)$ and $(\Psi, \eta^\Psi, \mu^\Psi)$, is how to define a multiplication $\mu^{\Phi\Psi} : \Phi\Psi\Phi\Psi \longrightarrow \Phi\Psi$. To that purpose, our construction will make use of a natural transformation, called *swapper* transformation, $\sigma\colon \Psi \circ \Phi \longrightarrow \Phi \circ \Psi$ that will allow us to use μ^Φ and μ^Ψ to define a suitable multiplication for the composition of the functors $\Phi \circ \Psi$.

In the following, some theorems that provide general techniques for monad compositions are presented. Proofs here are omitted and for more details, the reader is referred to the corresponding papers. The proof of Theorem 2.1 follows the same

[3] $P_0 X = PX \setminus \{\emptyset\}$

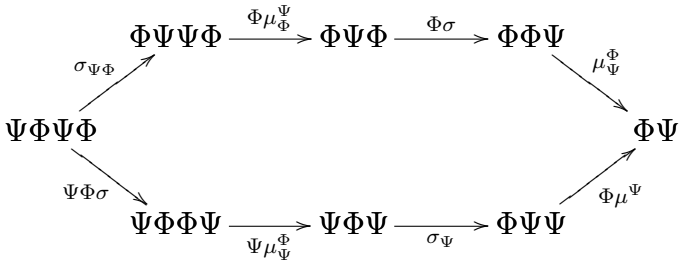


Figure 2.1: Independence of the swapper

steps as the proof of Proposition 3.1, that it is included in the next chapter. A proof of Theorem 2.2 using a *graphical approach* can be found in Paper II.

Theorem 2.1 ([23]) *Let* $\mathbf{\Phi} = (\mathbf{\Phi}, \eta^{\mathbf{\Phi}}, \mu^{\mathbf{\Phi}})$ *and* $\mathbf{\Psi} = (\mathbf{\Psi}, \eta^{\mathbf{\Psi}}, \mu^{\mathbf{\Psi}})$ *be monads and let* $\sigma\colon \Psi \circ \Phi \longrightarrow \Phi \circ \Psi$ *be a natural transformation such that the following properties hold:*

$$\Phi\mu^{\Psi} \circ \sigma\Psi \circ \Psi\mu^{\Phi\Psi} = \mu^{\Phi\Psi} \circ \Phi\mu^{\Psi}\Phi\Psi \circ \sigma\Psi\Phi\Psi, \tag{2.3}$$

$$\sigma \circ \eta^{\Psi}\Phi = \Phi\eta^{\Psi}, \tag{2.4}$$

$$\sigma\Psi \circ \Psi\eta^{\Phi\Psi} = \eta^{\Phi}\Psi\Psi \circ \eta^{\Psi}\Psi. \tag{2.5}$$

Then $\mathbf{\Phi} \bullet \mathbf{\Psi} = (\mathbf{\Phi} \circ \mathbf{\Psi}, \eta^{\mathbf{\Phi}}\mathbf{\Psi} \circ \eta^{\mathbf{\Psi}}, \mathbf{\Phi}\mu^{\mathbf{\Psi}} \circ \mu^{\mathbf{\Phi}}\mathbf{\Psi}\mathbf{\Psi} \circ \mathbf{\Phi}\sigma\mathbf{\Psi})$ *is a monad.*

Theorem 2.2 ([14]) *Given monads* $\mathbf{\Phi} = (\mathbf{\Phi}, \eta^{\mathbf{\Phi}}, \mu^{\mathbf{\Phi}})$ *and* $\mathbf{\Psi} = (\mathbf{\Psi}, \eta^{\mathbf{\Psi}}, \mu^{\mathbf{\Psi}})$*, let* $\sigma\colon \Psi \circ \Phi \longrightarrow \Phi \circ \Psi$ *be a natural transformation, then the composition can be provided with a multiplication* $\mu(\sigma)\colon \Phi\Psi\Phi\Psi \longrightarrow \Phi\Psi$ *defined by* $\mu(\sigma) = (\mu^{\Phi} * \mu^{\Psi}) \circ \Phi\sigma\Psi$ *which makes* $\mathbf{\Phi} \bullet \mathbf{\Psi} = (\mathbf{\Phi} \circ \mathbf{\Psi}, \eta^{\mathbf{\Phi}} * \eta^{\mathbf{\Psi}}, \mu(\sigma))$ *to be a monad if the following properties hold:*

$$\Phi\mu^{\Psi} \circ \sigma\Psi \circ \Psi\mu^{\Phi}\Psi \circ \Psi\Phi\sigma = \mu^{\Phi}\Psi \circ \Phi\sigma \circ \Phi\mu^{\Psi}\Phi \circ \sigma\Psi\Phi, \tag{2.6}$$

$$\sigma \circ \eta^{\Psi}\Phi = \Phi\eta^{\Psi}, \tag{2.7}$$

$$\sigma \circ \Psi\eta^{\Phi} = \eta^{\Phi}\Psi. \tag{2.8}$$

Note that Condition (2.6) expresses the commutativity of the diagram in Figure 2.1 which, essentially, states the independence of the swapper from the order of application. By Condition (2.7) the swapper extracts Φ from η^{Ψ}. By Condition (2.8) the swapper introduces Ψ in η^{Φ}.

2.3.2 Reverse engineering of monads

A converse result to Theorem 2.2 can be partially achieved under some extra assumptions on the behavior of the multiplication of the composite monad w.r.t. either the multiplications or the units of the base monads.

Theorem 2.3 ([14]) *If $\mathbf{\Phi} \bullet \mathbf{\Psi} = (\mathbf{\Phi} \circ \mathbf{\Psi}, \eta^{\mathbf{\Phi}} * \eta^{\mathbf{\Psi}}, \mu)$ is a monad, then a natural transformation $\sigma(\mu)\colon \Psi\Phi \longrightarrow \Phi\Psi$ can be defined by $\sigma(\mu) = \mu \circ \Phi\Psi\Phi\eta^{\Psi} \circ \eta^{\Phi}\Psi\Phi$ such that the Conditions (2.7) and (2.8) are satisfied. In addition, Condition (2.6) holds and $\mu = \mu(\sigma(\mu))$ under the assumption of any pair of properties (A_i, B_j) with $i, j \in \{1, 2\}$, where*

$$\mu^{\Phi}\Psi \circ \Phi\mu = \mu \circ \mu^{\Phi}\Psi\Phi\Psi, \qquad (A_1)$$

$$\Phi\mu \circ \Phi\eta^{\Phi}\Psi\Phi\Psi = \Phi\mu \circ \eta^{\Phi}\Phi\Psi\Phi\Psi. \qquad (A_2)$$

$$\Phi\mu^{\Psi} \circ \mu\Psi = \mu \circ \Phi\Psi\Phi\mu^{\Psi}, \qquad (B_1)$$

$$\mu\Psi \circ \Phi\Psi\Phi\eta^{\Psi}\Psi = \mu\Psi \circ \Phi\Psi\Phi\Psi\eta^{\Psi}. \qquad (B_2)$$

Again, a proof of this theorem using the graphical approach, can be found in Paper II.

2.3.3 Submonad Compositions

Given $\mathbf{\Phi}' \preceq \mathbf{\Phi}$ and $\mathbf{\Psi}' \preceq \mathbf{\Psi}$, we will now provide conditions under which compositions of the functors $\Phi' \circ \Psi$, $\Phi \circ \Psi'$ and $\Phi' \circ \Psi'$ can be extended to submonads of $\mathbf{\Phi} \circ \mathbf{\Psi}$.

Theorem 2.4 *Let $\mathbf{\Phi} \bullet \mathbf{\Psi} = (\Phi \circ \Psi, \eta^{\Phi}\Psi \circ \eta^{\Psi}, \Phi\mu^{\Psi} \circ \mu^{\Phi}\Psi\Psi \circ \Phi\sigma\Psi)$ be the monad given as in Theorem 2.3 and let $\mathbf{\Phi}'$ and $\mathbf{\Psi}'$ be submonads of $\mathbf{\Phi}$ and $\mathbf{\Psi}$, respectively.*

If there exists a natural transformation $\sigma^*\colon \Psi'\Phi' \longrightarrow \Phi'\Psi'$ *such that* $e^{\Phi}\Psi \circ \Phi e^{\Psi} \circ \sigma^* = \sigma \circ e^{\Psi}\Phi \circ \Psi e^{\Phi}$*, where* e^{Ψ} *and* e^{Ψ} *are the mappings given by the submonad condition of* Φ' *and* Ψ'*, respectively, then* $\mathbf{\Phi' \bullet \Psi'} = (\Phi' \circ \Psi', \eta^{\Phi'\Psi'} \circ \eta^{\Psi'}, \Phi'\mu^{\Psi'} \circ \mu^{\Phi'\Psi'\Psi'} \circ \Phi'\sigma^*\Psi')$ *is a submonad of* $\mathbf{\Phi \bullet \Psi}$*.*

Proof

Consider $\mathbf{\Phi \bullet \Psi} = (\Phi \circ \Psi, \eta, \mu) = (\Phi \circ \Psi, \eta^{\Phi}\Psi \circ \eta^{\Psi}, \Phi\mu^{\Psi} \circ \mu^{\Phi}\Psi\Psi \circ \Phi\sigma\Psi)$ and $\mathbf{\Phi' \bullet \Psi'} = (\Phi' \circ \Psi', \eta^*, \mu^*) = (\Phi' \circ \Psi', \eta^{\Phi'}\Psi' \circ \eta^{\Psi'}, \Phi'\mu^{\Psi'} \circ \mu^{\Phi'}\Psi'\Psi' \circ \Phi'\sigma^*\Psi')$.

Clearly $\Phi' \circ \Psi' \leq \Phi \circ \Psi$ since the inclusion transformation $e^* : \Phi' \circ \Psi' \longrightarrow \Phi \circ \Psi$ is given by the star composition of the corresponding inclusion transformations for Φ' and Ψ' respectively, i.e, $e^* = e^{\Phi} * e^{\Psi} (= e^{\Phi}\Psi \circ \Phi e^{\Psi})$.

In the following we will verify the submonad conditions. Using the submonad conditions for Φ' and Ψ' we get,

$$\begin{aligned} e^* \circ \eta^* &= \Phi e^{\Psi} \circ e^{\Phi}\Psi' \circ \eta^{\Phi'}\Psi' \circ \eta^{\Psi'} = \Phi e^{\Psi} \circ (e^{\Phi} \circ \eta^{\Phi'})\Psi' \circ \eta^{\Psi'} \\ &= \Phi e^{\Psi} \circ \eta^{\Phi}\Psi' \circ \eta^{\Psi'} = \Phi e^{\Psi} \circ \Phi\eta^{\Psi'} \circ \eta^{\Phi} = \Phi(e^{\Psi} \circ \eta^{\Psi'}) \circ \eta^{\Phi} \\ &= \Phi\eta^{\Psi} \circ \eta^{\Phi} = \eta^{\Phi}\Psi \circ \eta^{\Psi} \\ &= \eta, \end{aligned}$$

and by the condition of σ^* and the naturality property, we get

$$\begin{aligned} e^* \circ \mu^* &= \Phi e^{\Psi} \circ e^{\Phi}\Psi' \circ \Phi'\mu^{\Psi'} \circ \mu^{\Phi'}\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi e^{\Psi} \circ \Phi\mu^{\Psi} \circ e^{\Phi}\Psi'\Psi' \circ \mu^{\Phi'}\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi(e^{\Psi} \circ \mu^{\Psi}) \circ (e^{\Phi} \circ \mu^{\Phi'})\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi(\mu^{\Psi} \circ \Psi e^{\Psi} \circ e^{\Psi}\Psi') \circ (\mu^{\Phi} \circ \Phi e^{\Phi} \circ e^{\Phi}\Phi')\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi\mu^{\Psi} \circ \Phi\Psi e^{\Psi} \circ \Phi e^{\Psi}\Psi' \circ \mu^{\Phi}\Psi'\Psi' \circ \Phi e^{\Phi}\Psi'\Psi' \circ e^{\Phi}\Phi'\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi\mu^{\Psi} \circ \Phi\Psi e^{\Psi} \circ \mu^{\Phi}\Psi\Psi' \circ \Phi\Phi e^{\Psi}\Psi' \circ \Phi e^{\Phi}\Psi'\Psi' \circ e^{\Phi}\Phi'\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi\mu^{\Psi} \circ \mu^{\Phi}\Psi\Psi \circ \Phi\Phi\Psi e^{\Psi} \circ \Phi\Phi e^{\Psi}\Psi' \circ \Phi e^{\Phi}\Psi'\Psi' \circ e^{\Phi}\Phi'\Psi'\Psi' \circ \Phi'\sigma^*\Psi' \\ &= \Phi\mu^{\Psi} \circ \mu^{\Phi}\Psi\Psi \circ \Phi\Phi\Psi e^{\Psi} \circ \Phi\Phi e^{\Psi}\Psi' \circ \Phi e^{\Phi}\Psi'\Psi' \circ \Phi\sigma^*\Psi' \circ e^{\Phi}\Psi'\Phi'\Psi' \end{aligned}$$

$$
\begin{aligned}
&= \Phi\mu^{\Psi} \circ \mu^{\Phi}\Psi\Psi \circ \Phi\Phi\Psi e^{\Psi} \circ \Phi\sigma\Psi' \circ \Phi\Psi e^{\Phi}\Psi' \circ \Phi e^{\Psi}\Phi'\Psi' \circ e^{\Phi}\Psi'\Phi'\Psi' \\
&= \Phi\mu^{\Psi} \circ \mu^{\Phi}\Psi\Psi \circ \Phi\sigma\Psi \circ \Phi\Psi\Phi e^{\Psi} \circ \Phi\Psi e^{\Phi}\Psi' \circ \Phi e^{\Psi}\Phi'\Psi' \circ e^{\Phi}\Psi'\Phi'\Psi' \\
&= \mu \circ \Phi\Psi(\Phi e^{\Psi} \circ e^{\Phi}\Psi') \circ (\Phi e^{\Psi} \circ e^{\Phi}\Psi')\Phi'\Psi' \\
&= \mu \circ \Phi\Psi e^{*} \circ e^{*}\Phi'\Psi'. \quad \blacksquare
\end{aligned}
$$

Remark 2.1 *Under the conditions of Theorem 2.4, in the particular case that* $\Psi' = \Psi$*, if there exists a natural transformation* $\sigma' \colon \Psi\Phi' \longrightarrow \Phi'\Psi$*, such that* $e^{\Phi}\Psi \circ \sigma' = \sigma \circ \Psi e^{\Phi}$*, where* e^{Φ} *is the mapping given by the submonad condition of* Φ'*, then* $\mathbf{\Phi' \bullet \Psi} = (\Phi' \circ \Psi, \eta^{\Phi'}\Psi \circ \eta^{\Psi}, \Phi'\mu^{\Psi} \circ \mu^{\Phi'}\Psi\Psi \circ \Phi'\sigma'\Psi)$ *is a submonad of* $\mathbf{\Phi \bullet \Psi}$*. A similar observation can be made in the case of* $\Phi' = \Phi$*.*

It is important to stress the difficulty of providing monad compositions. An apparently useful composition of functors in corresponding monads can turn out not to be extendable to a monad. In fact, it is not clear if extendability of such compositions of functors to a monad composition is more a rule than an exception. Furthermore, when trying to compose sequences of monads, the interplay between respective swappers is expectedly complicated.

Chapter 3

Generalised Terms

In the previous chapter we studied conditions under which monads could be composed in order to obtain again a monad. The composition of monads provides a method for extending the notion of terms. Variable substitutions, viewed as morphisms in the corresponding Kleisli categories over composed monads, can then be seen more generally as variables assigned e.g. to (many-valued) sets of terms.

Composing various powerset functors with the term monad gives rise to the concept of *generalised terms*. This in turn provides a technique for handling powersets of terms in a framework of variable substitutions, thus being the prerequisite for categorical unification in many-valued logic programming using an extended notion of terms.

The use of categorical tools in logic programming leads naturally to the problem of categorically expressing fuzzy sets of generalised terms, in order to achieve a categorical paradigm of fuzzy logic programming. There are different possibilities to get a fuzzy set of generalised terms. Extending the idea by Rydeheard and Burstall in [56] which considered most general unifiers of a set of terms as coequalizers in the Kleisli category associated with the term monad, one naturally has to consider the composition of the L-fuzzy powerset monads and the term monad. Specifically, we have to check whether the composition of these fuzzy powerset monads with the term

monad can be extended to a monad.

In this chapter we apply the results presented in the previous chapter to provide generalised terms.

3.1 Powerset of Terms

Powersets of terms are given by composition of various powerset functors, as monads, with the term monad. Interesting compositions are those where the resulting composed functor can be extended to a monad. This enables a study of substitutions and unifiers within Kleisli categories related to particular monads. Morphisms in these categories correspond to variables being assigned to set of terms.

A first approach to the study of the structure of monad of the composition of powerset monads and term monads was presented in [23], where it was shown how set functors can be composed to providing monads, and some motivation to investigate techniques for constructing new monads from given ones was presented.

In the previous chapter we introduced a set of sufficient conditions for two monads being composable, and presented results on the structure of the multiplication of the composite monad, i.e. its structure can be determined under certain conditions. We will now use those results for defining generalised terms.

3.1.1 Composition of L_{id} and T_Ω

In the following we will write L instead of L_{id} and T instead of T_Ω.

In [23] we presented our first result of generalised terms. In that paper we proved how the composition of the powerset monad $\mathbf{L}$ and the term monad $\mathbf{T}$, could be extended to a monad $\mathbf{L} \circ \mathbf{T}$. At that time we did not use any general theorem on monad composition. On the contrary, our first theorem related to techniques for composing monads appeared as a consequence of the results we obtained there.

The *ad hoc* results on the composition of L and T are reproduced here mainly for historical reasons that relates to the start of this research.

Our constructions will make use of the mapping $\sigma^L_X : TLX \to LTX$ defined as follows (note that we will also write σ instead of σ^L for brevity):

For the base case $\sigma_{X|T^0LX} = id_{LX}$. Further, for $l = (n, \omega, (l_i)_{i\leq n}) \in T^\alpha LX$, $\alpha > 0$, $l_i \in T^{\beta_i}LX$, $\beta_i < \alpha$, let

$$\sigma_X(l)((n', \omega', (m_i)_{i\leq n})) = \begin{cases} \bigwedge_{i\leq n} \sigma_X(l_i)(m_i) & \text{if } n = n' \text{ and } \omega = \omega' \\ \\ 0 & \text{otherwise} \end{cases}$$

Note that in the case of $\alpha > 0$, for $L = 2$ we have

$$\sigma_X(l) = \{(n, \omega, (m_i)_{i\leq n}) \mid m_i \in \sigma_X(l_i)\}.$$

Note also that, for $l \in TLX$ and $m \in TX$ we have $\sigma_X(l)(m) = 0$, if $l \in T^\alpha LX$ and $m \notin T^\alpha X$, or if $l \notin T^\alpha LX$ and $m \in T^\alpha X$.

Lemma 3.1 *$\sigma : T \circ L \to L \circ T$ is a natural transformation.*

Proof

For any $l \in TLX$, and any $X \xrightarrow{f} Y$ in `Set`, we need to show that $LTf \circ \sigma_X(l) = \sigma_Y \circ TLf(l)$. For $l \in T^0LX$, this is immediate. For $\alpha > 0$, we may write $l = (n, \omega, (l_i)_{i\leq n})$, where $l_i \in T^{\beta_i}LX$, $\beta_i < \alpha$, for all $i \leq n$. Let now $\overline{m} = (n, \omega, (\overline{m}_i)_{i\leq n}) \in TY$. Then,

$$LTf(\sigma_X(l))(\overline{m}) = \bigvee_{Tf((n,\omega,(m_i)_{i\leq n}))=\overline{m}} \bigwedge_{i\leq n} \sigma_X(l_i)(m_i).$$

Further, by induction, we get

$$\begin{aligned} \sigma_Y(TLf(l))(\overline{m}) &= \bigwedge_{i\leq n} \sigma_Y(TLf(l_i))(\overline{m}_i) \\ &= \bigwedge_{i\leq n} LTf(\sigma_X(l_i))(\overline{m}_i) \\ &= \bigwedge_{i\leq n} \bigvee_{Tf(m_i)=\overline{m}_i} \sigma_X(l_i)(m_i). \end{aligned}$$

By complete distributivity of L, we then obtain naturality of σ. ∎

We will use this natural transformation σ in order to define the natural transformations η^{LT} and μ^{LT}, which provide LT with the structure of a monad.

Definition 3.1 *The natural transformations* $\eta^{LT}: id \longrightarrow LT$ *and* $\mu^{LT}: LTLT \longrightarrow LT$ *are defined as follows*

$$\eta^{LT} = \eta^L T \circ \eta^T \qquad\qquad \mu^{LT}_X = L\mu^T_X \circ \mu^L_{TTX} \circ L\sigma_{TX}$$

Note that $\eta^{LT}_X(x) = \eta^L_{TX}(x)$, and in the case of $L = 2$ then $\eta^{2T}_X(x) = \{x\}$. Further for $R \in LT^\alpha LTX$, $\alpha > 0$, and $m \in TX$, note that

$$\mu^{LT}_X(R)(m) = \bigvee_{r \in TLTX} R(r) \wedge \sigma_{TX}(r)(m).$$

and also note that in the case $L = 2$, for $R = \{(n_j, \omega_j, (r_{ij})_{i \leq n_j}) \mid j \in J\} \in 2T^\alpha 2TX$, $\alpha > 0$, we have

$$\mu^{2T}_X(R) = \{(n_j, \omega_j, (m_{ij})_{i \leq n_j}) \mid j \in J, m_{ij} \in \sigma_{TX}(r_{ij})\}$$

The following technical lemma gives some conditions which guarantee the monad structure for the composition LT.

Lemma 3.2 *The following properties hold:*

$$\sigma_{TX} \circ T\eta^{LT}_X = \eta^L_{TTX} \circ \eta^T_{TX}, \tag{3.1}$$

$$L\mu^T_X \circ \sigma_{TX} \circ T\mu^{LT}_X = \mu^{LT}_X \circ L\mu^T_{LTX} \circ \sigma_{TLTX}, \tag{3.2}$$

$$\sigma_X \circ \eta^T_{LX} = L\eta^T_X. \tag{3.3}$$

Note that Properties (3.1) and (3.2) in the case of $L = 2$ become

(3.1)' $\sigma_{TX}(T\eta^{2T}_X(m)) = \{m\}$, for all $m \in TX$,

(3.2)' $2\mu^T_X \circ \sigma_{TX} \circ T\mu^{2T}_X(d) = \bigcup_{R \in \sigma_{T2TX}(d)} \sigma_{TX}(R).$

Proof

(3.1) This holds trivially for $\alpha = 0$. In case of $\alpha > 0$, for $m = (n, \omega, (m_i)_{i \leq n}) \in TX$ and $m' = (n, \omega, (m'_i)_{i \leq n}) \in TX$, by induction, we get

$$\begin{aligned} \sigma_{TX}(T\eta_X^{LT}(m))(m') &= \sigma_{TX}((n, \omega, (T\eta_X^{LT}(m_i))_{i \leq n}))(m') \\ &= \bigwedge_{i \leq n} \sigma_{TX}(T\eta_X^{LT}(m_i))(m'_i) \\ &= \bigwedge_{i \leq n} \eta_{TX}^{L}(m_i)(m'_i). \end{aligned}$$

Since $m = m'$ if and only if $m_i = m'_i$ for all $i \leq n$, we immediately get

$$\eta_{TX}^{L}(m)(m') = \bigwedge_{i \leq n} \eta_{TX}^{L}(m_i)(m'_i).$$

(3.2) Again this holds trivially for $\alpha = 0$. In case of $\alpha > 0$, let $m = (n, \omega, (m_i)_{i \leq n}) \in TX$ and $d = (n, \omega, (d_i)_{i \leq n}) \in TLTLTX$. By induction and complete distributivity of L we then have

$$\begin{aligned} \sigma_{TX}(T\mu_X^{LT}(d))(m) &= \bigwedge_{i \leq n} \sigma_{TX}(T\mu_X^{LT}(d_i))(m_i) \\ &= \bigwedge_{i \leq n} \mu_X^{LT}(\sigma_{TLTX}(d_i))(m_i) \\ &= \bigwedge_{i \leq n} \bigvee_{r \in TLTX} \sigma_{TLTX}(d_i)(r) \wedge \sigma_{TX}(r)(m_i) \\ &= \bigvee_{(n, \omega, (r_i)) \in TLTX} \bigwedge_{i \leq n} \sigma_{TLTX}(d_i)(r_i) \wedge \sigma_{TX}(r_i)(m_i) \\ &= \bigvee_{r \in TLTX} \sigma_{TLTX}(d)(r) \wedge \sigma_{TX}(r)(m) \\ &= \mu_X^{LT}(\sigma_{TLTX}(d))(m). \end{aligned}$$

(3.3) By definition, as $\sigma_{X|T^0LX} = id_{LX}$. ∎

Proposition 3.1 *$(L_{id} \circ T_\Omega, \eta^{L_{id} \circ T_\Omega}, \mu^{L_{id} \circ T_\Omega})$, denoted $\mathbf{L}_{id} \bullet \mathbf{T}_\Omega$, is a monad.*

Proof

We have

$$\begin{aligned}\mu_X^{LT} \circ LT\eta_X^{LT} &= L\mu_X^T \circ \mu_{TTX}^L \circ L\sigma_{TX} \circ LT\eta_X^{LT} \\ &= L\mu_X^T \circ \mu_{TTX}^L \circ L\eta_{TTX}^L \circ L\eta_{TX}^T \\ &= L\mu_X^T \circ L\eta_{TX}^T = id_{LTX}\end{aligned}$$

and

$$\begin{aligned}\mu_X^{LT} \circ \eta_{LTX}^{LT} &= L\mu_X^T \circ \mu_{TTX}^L \circ L\sigma_{TX} \circ \eta_{TLTX}^L \circ \eta_{LTX}^T \\ &= L\mu_X^T \circ \mu_{TTX}^L \circ \eta_{LTTX}^L \circ \sigma_{TX} \circ \eta_{LTX}^T \\ &= L\mu_X^T \circ \sigma_{TX} \circ \eta_{LTX}^T \\ &= L\mu_X^T \circ L\eta_{TX}^T = Lid_{TX} = id_{LTX}\end{aligned}$$

Further we have the associativity of μ^{LT}

$$\begin{aligned}\mu_X^{LT} \circ LT\mu_X^{LT} &= L\mu_X^T \circ \mu_{TTX}^L \circ L\sigma_{TX} \circ LT\mu_X^{LT} \\ &= \mu_{TX}^L \circ LL\mu_X^T \circ L\sigma_{TX} \circ LT\mu_X^{LT} \\ &= \mu_{TX}^L \circ L\mu_X^{LT} \circ LL\mu_{LTX}^T \circ L\sigma_{TLTX} \\ &= \mu_{TX}^L \circ LL\mu_X^T \circ L\mu_{TTX}^L \circ LL\sigma_{TX} \circ LL\mu_{LTX}^T \circ L\sigma_{TLTX} \\ &= L\mu_X^T \circ \mu_{TTX}^L \circ L\mu_{TTX}^L \circ LL\sigma_{TX} \circ LL\mu_{LTX}^T \circ L\sigma_{TLTX} \\ &= L\mu_X^T \circ \mu_{TTX}^L \circ \mu_{LTTX}^L \circ LL\sigma_{TX} \circ LL\mu_{LTX}^T \circ L\sigma_{TLTX} \\ &= L\mu_X^T \circ \mu_{TTX}^L \circ L\sigma_{TX} \circ \mu_{TLTX}^L \circ LL\mu_{LTX}^T \circ L\sigma_{TLTX} \\ &= L\mu_X^T \circ \mu_{TTX}^L \circ L\sigma_{TX} \circ L\mu_{LTX}^T \circ \mu_{TTLTX}^L \circ L\sigma_{TLTX} \\ &= \mu_X^{LT} \circ L\mu_{LTX}^T \circ \mu_{TTLTX}^L \circ L\sigma_{TLTX} \\ &= \mu_X^{LT} \circ \mu_{LTX}^{LT} \quad \blacksquare\end{aligned}$$

In the proof of the proposition above, the actual definition of the functors L and T has not been used, only universal properties and those stated in Lemma 3.2. A general version of the proposition just proved can be stated as in Theorem 2.1.

Note that the composition of L and T can also be seen as a particular case in Theorem 2.2.

3.1.2 L-fuzzy Functors and the Term Monad

In Section 2.1.3 we introduced a number of set functors, which extend the crisp powerset functor, together with their extension principles. In Paper I, L-fuzzy set categories are defined for each of these extended powerset functors and the rationality of the extension principle is proved in the categorical sense i.e. the associated L-fuzzy set categories are shown to be equivalent to the category of sets and mappings.

It was stated before in Section 2.1.3 that each of these new set functors are given a structure of monad.

Paper I shows the details about how the composition $\mathcal{L}_\alpha \circ \mathcal{T}$ can be made a monad. This composition provide a fuzzification of set of terms by means of fuzzy set of terms.

3.1.3 Composing with Submonads

Submonads provide a useful mechanism for constructing additional monad compositions from submonads of given main monads. In section 2.1.4 some examples of submonads were presented. Now those examples can be use to provide generalised terms:

Example 3.1 *Let K and L be completely distributive lattices. Assume K to be a sublattice of L, with $\iota : K \longrightarrow L$ being the inclusion homomorphism. Further, assume $\iota(0) = 0$ and $\iota(1) = 1$, and additionally, that $\iota(\vee_i x_i) = \vee_i \iota(x_i)$ also in the non-finite case. Define $(\iota_{id})_X : K_{id}X \longrightarrow L_{id}X$ by $(\iota_{id})_X(A) = \iota \circ A$, $A : X \longrightarrow K$. In Example 2.1 we saw that $\mathbf{K}_{id}$ is a submonad of $\mathbf{L}_{id}$. Note that by Remark 2.1, it is straightforward to show that $\mathbf{K}_{id} \bullet \mathbf{T}_\Omega$ is a submonad of $\mathbf{L}_{id} \bullet \mathbf{T}_\Omega$.*

Example 3.2 *In Section 2.1.3 we defined functors for α-upper L-fuzzy sets and α-lower L-fuzzy sets, denoted $\mathcal{L}_\alpha$ and $\mathcal{L}^\alpha$ respectively.*

Example 2.2 shows that $(\mathcal{L}_\alpha, \eta^{\mathcal{L}_\alpha}, \mu^{\mathcal{L}_\alpha})$ *and* $(\mathcal{L}^\alpha, \eta^{\mathcal{L}^\alpha}, \mu^{\mathcal{L}^\alpha})$ *are submonads of* $\mathbf{L}_{id}$*. By Remark 2.1, it can be seen that* $(\mathcal{L}_\alpha, \eta^{\mathcal{L}_\alpha}, \mu^{\mathcal{L}_\alpha}) \bullet \mathbf{T}_\Omega$ *and* $(\mathcal{L}^\alpha, \eta^{\mathcal{L}^\alpha}, \mu^{\mathcal{L}^\alpha}) \bullet \mathbf{T}_\Omega$ *are submonads of* $\mathbf{L}_{id} \bullet \mathbf{T}_\Omega$*.*

Example 3.3 *Let* Ω' *and* Ω *be operator domains with* $\Omega' \subseteq \Omega$*, and let* $\epsilon : \Omega' \longrightarrow \Omega$ *be the inclusion mapping. Define* $\nu_X : T_{\Omega'}X \longrightarrow T_\Omega X$ *by* $\nu_X(x) = x$*,* $x \in X$*, and* $\nu_X((n, \omega', (t'_i)_{i\leq n})) = (n, \epsilon(\omega'), (\nu_X(t'_i))_{i\leq n})$ *for* $t'_i \in T_{\Omega'}X$*.*

$\mathbf{T}_{\Omega'}$ *is a submonad of* $\mathbf{T}_\Omega$ *as indicated in Example 2.3. Again, by Remark 2.1, it is easy to verify that* $\mathbf{L}_{id} \bullet \mathbf{T}_{\Omega'}$ *is a submonad of* $\mathbf{L}_{id} \bullet \mathbf{T}_\Omega$*.*

3.2 Composing with the Double Contravariant Set Functor

In Section 2.1.5, the contravariant powerset functor L^{id} was presented. We established there how the covariant set functor $LL = L^{id} \circ L^{id}$ could be extended to a monad.

We can now think about the possibility of a new extension for generalised terms using that monad. According to the results in Section 2.3.1, a swapper transformation is needed in order to make the composition $LL \circ T$ a monad.

A swapper $\sigma_X : TLLX \to LLTX$ can be proposed as follows. For the base case $\sigma_{X|T^0LLX} = id_{LLX}$. Further, for $k = (n, \omega, (k_i)_{i\leq n}) \in T^\alpha LLX$, $\alpha > 0$, $k_i \in T^{\beta_i}LLX$, $\beta_i < \alpha$, let

$$\sigma_X(k)(M) = \bigvee_{M^{(\omega)} = \sqcap_{i\leq n} M_i > 0} \bigwedge_{i\leq n} \sigma_X(k_i)(M_i),$$

where $M^{(\omega)}, \sqcap_{i\leq n} M_i : (TX)^n \longrightarrow L$ are mappings given by

$$M^{(\omega)}(t_1, \ldots, t_n) = M((n, \omega, (t_i)_{i\leq n}))$$

and $(\sqcap_{i\leq n} M_i)(t_1, \ldots, t_n) = \wedge_{i\leq n} M_i(t_i)$.

Note that in the case of $\alpha > 0$, for $L = 2$ we have

$$\sigma_X(k) = \Big\{\{(n, \omega, (t_i)_{i\leq n}) \mid t_i \in M_i\} \cup A \mid M_i \in \sigma_X(k_i),\ A \subseteq \{(m, \omega', (t'_i)_{i\leq m}) \mid \omega' \neq \omega\}\Big\}.$$

Lemma 3.3 *$\sigma : T \circ LL \to LL \circ T$ is a natural transformation.*

Proof
For any $k \in TLLX$, and any $X \xrightarrow{f} Y$ in $\mathtt{Set}$, we need to show that $(LLTf \circ \sigma_X)(k) = (\sigma_Y \circ TLLf)(k)$. For $k \in T^0LLX$, this is immediate. For $\alpha > 0$, we may write $k = (n, \omega, (k_i)_{i\leq n})$, where $k_i \in T^{\beta_i}LLX$, $\beta_i < \alpha$, for all $i \leq n$. Let now $N \in LTY$ and consider $M_i \in LTX$ and $N_i \in LTY$. Then,

$$LLTf(\sigma_X(k))(N) = \sigma_X(k)(N \circ Tf) = \bigvee_{(N\circ Tf)^{(\omega)} = \sqcap_{i\leq n} M_i} \bigwedge_{i\leq n} \sigma_X(k_i)(M_i).$$

Further, by induction, we get

$$\begin{aligned}
\sigma_Y(TLLf(k))(N) &= \bigvee_{N^{(\omega)} = \sqcap_{i\leq n} N_i} \bigwedge_{i\leq n} \sigma_Y(TLLf(k_i))(N_i) \\
&= \bigvee_{N^{(\omega)} = \sqcap_{i\leq n} N_i} \bigwedge_{i\leq n} LLTf(\sigma_X(k_i))(N_i) \\
&= \bigvee_{N^{(\omega)} = \sqcap_{i\leq n} N_i} \bigwedge_{i\leq n} \sigma_X(k_i)(N_i \circ Tf).
\end{aligned}$$

For $N^{(\omega)} = \sqcap_{i\leq n} N_i$, we get

$$\begin{aligned}
(N \circ Tf)^{(\omega)}(t_1, \ldots, t_n) &= N \circ Tf((n, \omega, (t_i)_{i\leq n})) = N((n, \omega, (Tf(t_i)_{i\leq n}))) \\
&= \bigwedge_{i\leq n} N_i(Tf(t_i)) = \bigwedge_{i\leq n} (N_i \circ Tf)(t_i),
\end{aligned}$$

which immediately gives

$$LLTf(\sigma_X(k))(N) \ \geq \ \sigma_Y(TLLf(k))(N).$$

On the other hand, given M_i, $L_{id}Tf(M_i) \in LTY$, for $s = (n, \omega, (s_i)_{i\leq n}) \in TY$ we have

$$LTf(M_i)(s_i) = \bigvee_{Tf(t_i)=s_i} M_i(t_i)$$

where $t = (n, \omega, (t_i)_{i\leq n}) \in TX$. Note that we here actually make use of the covariant functor L_{id}. For $(N \circ Tf)^{(\omega)} = \bigwedge_{i\leq n} M_i$ we then get

$$\begin{aligned}
\bigwedge_{i\leq n}(L_{id}Tf(M_i) \circ Tf)(t_i) &= \bigwedge_{i\leq n} \bigvee_{Tf(t'_i)=Tf(t_i)} M_i(t'_i) \\
&= \bigvee_{(Tf(t'_1),\ldots,Tf(t'_n))=(Tf(t_1),\ldots,Tf(t_n))} \bigwedge_{i\leq n} M_i(t'_i) \\
&= \bigvee_{(Tf(t'_1),\ldots,Tf(t'_n))=(Tf(t_1),\ldots,Tf(t_n))} (N \circ Tf)^{(\omega)}(t'_1, \ldots, t'_n) \\
&= \bigvee_{(Tf(t'_1),\ldots,Tf(t'_n))=(Tf(t_1),\ldots,Tf(t_n))} N(\omega(Tf(t'_1), \ldots, Tf(t'_n))) \\
&= (N \circ Tf)^{(\omega)}(t_1, \ldots, t_n).
\end{aligned}$$

Then, we have,

$$\sqcap_{i\leq n}(L_{id}Tf(M_i) \circ Tf) = (N \circ Tf)^{(\omega)}$$

and therefore

$$LLTf(\sigma_X(k))(N) \leq \sigma_Y(TLLf(k))(N). \quad \blacksquare$$

It is still an open question if this natural transformation σ can be used to provide LLT with the structure of a monad.

Once we have defined generalised terms, one of the next steps is to deal with an appropriate concept of *unifier* for generalised terms. In order to define this concept we need to first define a notion of *similarity* for generalised terms. We will come back to these issues in Chapter 5.

Chapter 4

Visual Representations

In the study related to monads and monad compositions we can see how proofs become difficult to handle. Naturality properties related to monads, are a reason for complicated equations to appear. The more complicated the monads get, the more elaborate the proofs become.

The foundational understanding of monads has been well-known for decades, but proof techniques, especially related to monad compositions have not been developed. As monad compositions are basically built upon operations of corresponding natural transformations, proof techniques require an adequate handling of the basic combinatorial properties of functors and natural transformations (*Godement rules*).

A graphical approach to proving compliance with composability conditions can provide a useful tool for handling rather complicated compositions. For handling more complicated monad compositions, proof support is necessary. In [9, 25] it was discovered that combinatorial properties of functors and natural transformations can be represented more visually, in that the basic observation relates to distributivity of the *star product* of natural transformation with respect to composition of natural transformations. This visual technique is not widely known and has been used mainly in purely algebraic contexts [5].

The aim of this chapter is to introduce and further develop the graphical approach.

To demonstrate its use for monad compositions, some of the results we studied in previous chapters will be considered again.

4.1 Notation

In this section, the basic definitions and notation of the graphical approach are given. But before that we need to define the two different ways in which natural transformation can be composed, e.g, the *horizontal* composition (or *star product*) and *vertical* composition (the usual composition).

Definition 4.1 *Let $F, G, H : \mathcal{C} \longrightarrow \mathcal{D}$ be three functors and $\sigma : F \longrightarrow G$ and $\tau : G \longrightarrow H$ two natural transformations. The vertical composition of σ and τ, denoted by $\tau \bullet \sigma$ is defined by*

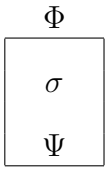
$$(\tau \bullet \sigma)_A = \tau_A \circ \sigma_A \qquad\qquad A \in Ob(\mathcal{C})$$

Definition 4.2 *Let $F, F' : \mathcal{C} \longrightarrow \mathcal{D}$ and $G, G' : \mathcal{D} \longrightarrow \mathcal{E}$ be functors and $\sigma : F \longrightarrow F'$ and $\tau : G \longrightarrow G'$ two natural transformations. The horizontal composition (or star product) of σ and τ, denoted by $\tau \star \sigma$ is defined, for each object A in the category $\mathcal{C}$, as the diagonal of the following commutative diagram:*

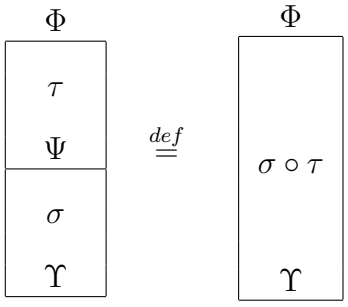
$$\begin{array}{ccc} GFA & \xrightarrow{\tau_{FA}} & G'FA \\ {\scriptstyle G\sigma_A}\downarrow & & \downarrow{\scriptstyle G'\sigma_A} \\ GF'A & \xrightarrow[\tau_{F'A}]{} & G'F'A \end{array}$$

i.e. $(\tau \star \sigma)_A = G'\sigma_A \circ \tau_{FA} = \tau_{F'A} \circ G\sigma_A$.

Let Φ and Ψ be (covariant) endofunctors in a category $\mathtt{C}$. Let σ be a natural transformation from Φ to Ψ, $\sigma\colon \Phi \longrightarrow \Psi$. The graphical notation uses a different representation, as a basic *building block* the natural transformation σ is depicted as:



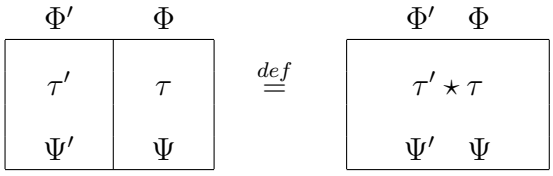
Definition 4.3 *Consider endofunctors Φ, Ψ, Υ in $\mathtt{C}$, together with natural transformations τ, σ between such endofunctors. For $\tau\colon \Phi \longrightarrow \Psi$ and $\sigma\colon \Psi \longrightarrow \Upsilon$, let $\sigma \circ \tau\colon \Phi \longrightarrow \Upsilon$ be the usual composition of natural transformations, represented by the vertical stacking of boxes*



Definition 4.4 *For $\tau'\colon \Phi' \longrightarrow \Psi'$, consider the star product $\tau' \star \tau\colon \Phi' \circ \Phi \longrightarrow \Psi' \circ \Psi$,*

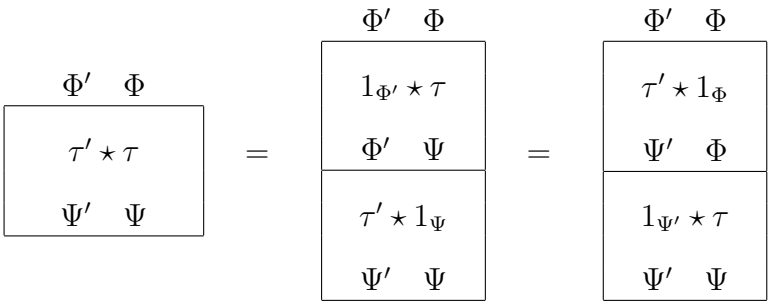
$$\tau' \star \tau = \tau'\Psi \circ \Phi'\tau = \Psi'\tau \circ \tau'\Phi \tag{4.1}$$

Its box representation is defined by



This representation is helpful in order to prove several well-known results regarding composition and star product.

For the identity transformation $id_\Phi\colon \Phi \longrightarrow \Phi$, also written as 1_Φ or simply 1, note that $1_\Phi \star 1_\Psi = 1_{\Phi\circ\Psi}$. For a natural transformation $\tau\colon \Phi \longrightarrow \Psi$, and a functor Υ, it is possible to define its composition $(\Upsilon\tau)_X = \Upsilon\tau_X$ and $(\tau\Upsilon)_X = \tau_{\Upsilon X}$, or equivalently, $\Upsilon\tau = 1_\Upsilon \star \tau$ and $\tau\Upsilon = \tau \star 1_\Upsilon$, which allows us to pictorially represent Equation (4.1) by

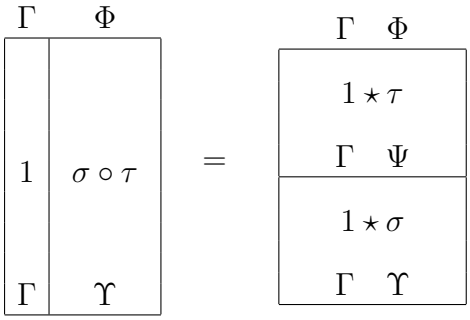


The following distributivity laws

$$1 \star (\sigma \circ \tau) = (1 \star \sigma) \circ (1 \star \tau), \tag{4.2}$$

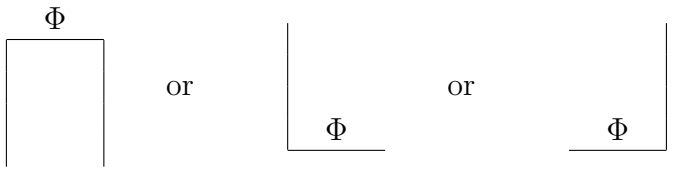
$$(\sigma \circ \tau) \star 1 = (\sigma \star 1) \circ (\tau \star 1). \tag{4.3}$$

have now a box representation. Equation (4.2) can be written as



i.e., in this case building blocks can be applied in any order. The same holds for equation (4.3).

In order to further improve readability of transformation expressions, identity transformations $1_\Phi : \Phi \longrightarrow \Phi$ as blocks within transformation expressions are depicted as



This choice for the representation of identity transformations will allow the use of asymmetric stacking of boxes.

4.1.1 Interchange Law

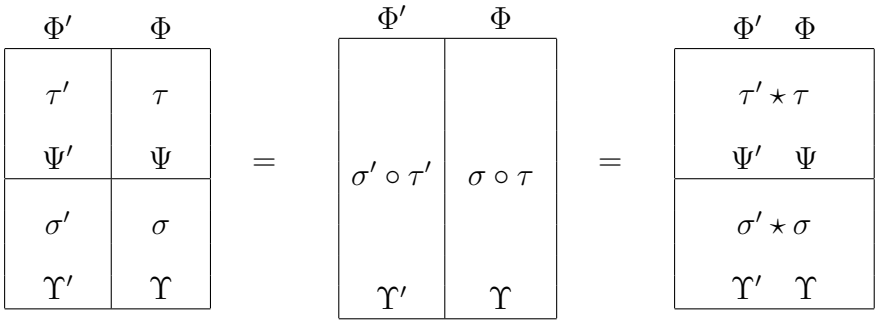
We stress the usefulness of this notation, based on the *interchange law*, which allows to perform calculations with natural transformations in an easy manner, e.g., the *interchange law* together with the box representation leads to a better management of operations with natural transformations; something which is not, at least directly, possible with the usual representation.

Let $\Phi \xrightarrow{\tau} \Psi \xrightarrow{\sigma} \Upsilon$ and $\Phi' \xrightarrow{\tau'} \Psi' \xrightarrow{\sigma'} \Upsilon'$ be natural transformations. The *Interchange Law* states that

$$(\sigma' \star \sigma) \circ (\tau' \star \tau) = (\sigma' \circ \tau') \star (\sigma \circ \tau) \tag{4.4}$$

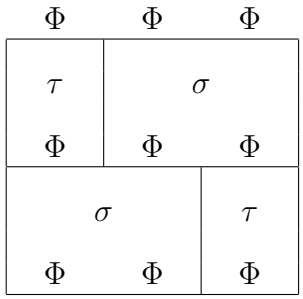
A proof for the Interchange Law using the graphical approach can be found in Paper II.

We can now visualize Equation (4.4) and obtain the following rule



which allows us to rearrange a stack of blocks as desired, showing how blocks with particular positions generally can be attached vertically and horizontally in any order without changing the resulting transformation.

Note in the transformation



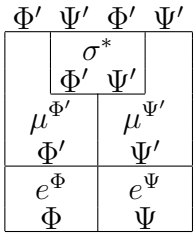
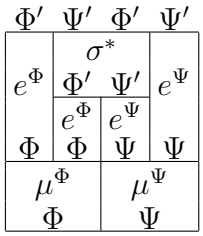
that the composition $(\sigma \star \tau) \circ (\tau \star \sigma)$ indeed exists, but neither $\tau \circ \sigma$ nor $\sigma \circ \tau$ do. This indicates how the applicability of the Interchange Law is more easily seen in the pictorial representation of the transformation.

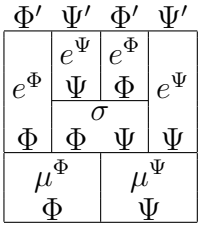
4.1.2 Comparisons

The impact of the graphical approach can be better understood by looking at a particular example. We will consider here part of a proof related to submonads (see Theorem 2.4). In the 'equational form' in order to prove that $e^* \circ \mu^* = \mu \circ (e^* \star e^*)$ the following steps are used:

$$
\begin{aligned}
e^* \circ \mu^* &= \Phi e^\Psi \circ e^\Phi \Psi' \circ \Phi' \mu^{\Psi'} \circ \mu^{\Phi'} \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi e^\Psi \circ \Phi \mu^\Psi \circ e^\Phi \Psi' \Psi' \circ \mu^{\Phi'} \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi(e^\Psi \circ \mu^\Psi) \circ (e^\Phi \circ \mu^{\Phi'}) \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi(\mu^\Psi \circ \Psi e^\Psi \circ e^\Psi \Psi') \circ (\mu^\Phi \circ \Phi e^\Phi \circ e^\Phi \Phi') \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi \mu^\Psi \circ \Phi \Psi e^\Psi \circ \Phi e^\Psi \Psi' \circ \mu^\Phi \Psi' \Psi' \circ \Phi e^\Phi \Psi' \Psi' \circ e^\Phi \Phi' \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi \mu^\Psi \circ \Phi \Psi e^\Psi \circ \mu^\Phi \Psi \Psi' \circ \Phi \Phi e^\Psi \Psi' \circ \Phi e^\Phi \Psi' \Psi' \circ e^\Phi \Phi' \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi \mu^\Psi \circ \mu^\Phi \Psi \Psi \circ \Phi \Phi \Psi e^\Psi \circ \Phi \Phi e^\Psi \Psi' \circ \Phi e^\Phi \Psi' \Psi' \circ e^\Phi \Phi' \Psi' \Psi' \circ \Phi' \sigma^* \Psi' \\
&= \Phi \mu^\Psi \circ \mu^\Phi \Psi \Psi \circ \Phi \Phi \Psi e^\Psi \circ \Phi \Phi e^\Psi \Psi' \circ \Phi e^\Phi \Psi' \Psi' \circ \Phi \sigma^* \Psi' \circ e^\Phi \Psi' \Phi' \Psi' \\
&= \Phi \mu^\Psi \circ \mu^\Phi \Psi \Psi \circ \Phi \Phi \Psi e^\Psi \circ \Phi \sigma \Psi' \circ \Phi \Psi e^\Phi \Psi' \circ \Phi e^\Psi \Phi' \Psi' \circ e^\Phi \Psi' \Phi' \Psi' \\
&= \Phi \mu^\Psi \circ \mu^\Phi \Psi \Psi \circ \Phi \sigma \Psi \circ \Phi \Psi \Phi e^\Psi \circ \Phi \Psi e^\Phi \Psi' \circ \Phi e^\Psi \Phi' \Psi' \circ e^\Phi \Psi' \Phi' \Psi' \\
&= \mu \circ \Phi \Psi (\Phi e^\Psi \circ e^\Phi \Psi') \circ (\Phi e^\Psi \circ e^\Phi \Psi') \Phi' \Psi' \\
&= \mu \circ \Phi \Psi e^* \circ e^* \Phi' \Psi' \\
&= \mu \circ (e^* \star e^*).
\end{aligned}
$$

Now, with a visual representation, we could easily switch to a more compact and hopefully readable expression:

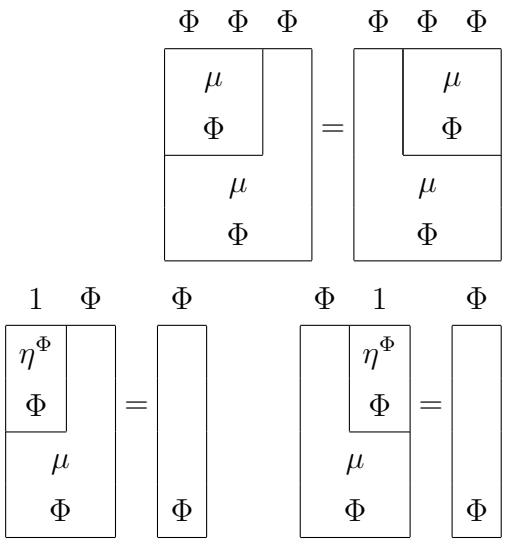
$e^* \circ \mu^* \quad = \quad$  $\quad = \quad$  $\quad =$

$\quad = \quad$  $\quad = \quad \mu \circ (e^* \star e^*)$

Comparisons between the standard proof and the pictorial one, allow us to see how the properties of natural transformations are more naturally handled in the graphical approach, so that one can abstract them from the main line of reasoning.

4.2 Visualizing Monads

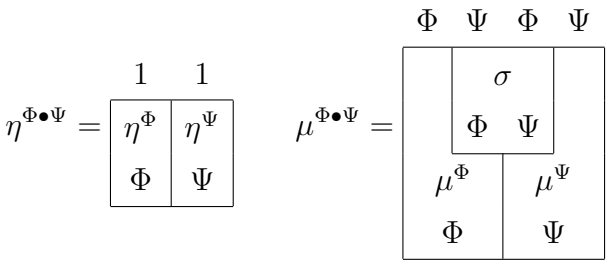
Recalling Definition 2.1 (definition of monad), the properties of the unit and multiplication, $\mu \circ \Phi\mu = \mu \circ \mu\Phi$ and $\mu \circ \Phi\eta = \mu \circ \eta\Phi = id_\Phi$ can now be visualized as



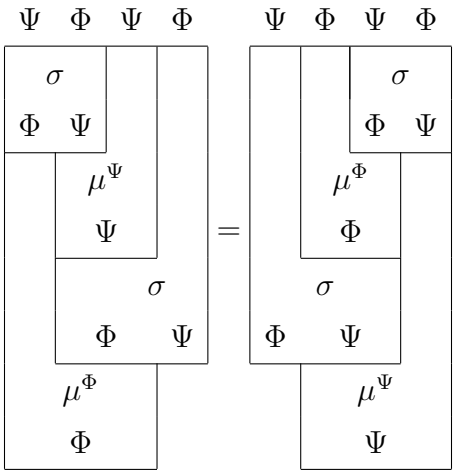
We can now see how the readability of the properties of a monad has improved by using the graphical approach. This notation allows to perform calculations with natural transformations in an easy manner and this fact becomes specially important when proving composability conditions.

4.2.1 Visualizing Monad compositions

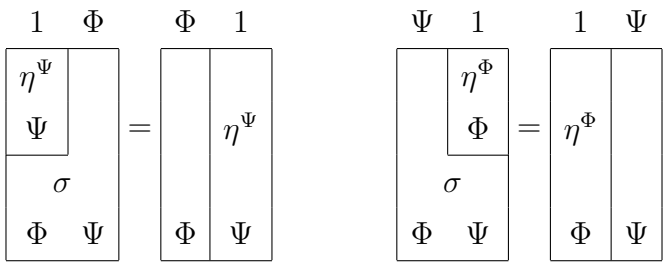
The graphical approach brings us to the following reformulation of Theorem 2.2: The definitions of the unit and multiplication for the composed monad in Theorem 2.2 are represented pictorially by



The pictorial representations for the conditions are given next.
The Condition (2.6) is:



The Conditions (2.7) and (2.8) are:



More applications of the graphical approach related to the study of monad compositions can be found in Papers II and III.

Chapter 5

Towards Categorical Unification

Generalised terms based on monad compositions are meant to be used in a unification framework. To that labor, some new concepts need to be considered in this chapter. *Fuzzy equality relation*, *fuzzy equivalence relation* and *similarity relation* are concepts that one often finds in the literature. We adopt here the latter terminology.

A formal treatment of *similarities* and *equalities* used in many-valued predicate logic can be seen in [32].

Even if not included in this thesis, similarities can be also considered from a *topos-theoretic* point of view. The *topos* oriented situation where *Heyting algebras* have idempotent conjunctions is not acceptable in many-valued considerations and thus we need amendments such as those involving monoidal non-idempotent conjunctions, leading to the weak topos framework (see [61, 34] for more detail).

In this chapter we propose similarity relations for the term functor, the powerset term functor and the term powerset functor. The main purpose of the chapter is to demonstrate how the concept of unifiers can be extended by considering powersets of terms. We will focus in particular in the case of the composition $P \circ T$, (case $L = 2$ in Section 3.1.1). In Section 5.4 an example is included to illustrate the use of the concepts defined in this chapter.

5.1 Similarity Relations for Generalised Terms

In this section, we start by introducing the concept of *similarity*. Further, *similarity relations* for the term functor, the powerset term functor and the term powerset functor will be offered. Proofs related to this section can be found in Paper V.

Let L be a completely distributive lattice and X an object in `Set`. A *fuzzy set* A is a mapping $A\colon X \longrightarrow L$ where the value $A(x) \in L$ is interpreted as the membership value of x into A. The set of all fuzzy sets associated to X is denoted by L^X.

One can define relations (fuzzy relations) between two objects X, Y, as a mappings $R\colon X \times Y \longrightarrow L$. A *similarity relation* is a particular kind of these relations.

Definition 5.1 *A* similarity *on X is a mapping $E\colon X \times X \longrightarrow L$ satisfying*

$$E(x,x) = 1 \qquad \textit{(reflexivity)}$$

$$E(x,y) = E(y,x) \qquad \textit{(symmetry)}$$

$$E(x,y) \wedge E(y,z) \leq E(x,z) \qquad \textit{(transitivity)}$$

for all $x, y, z \in X$.

It is easily seen that in the crisp case i.e. $L = 2$, similarity relations are equivalence relations: two elements x, y can be either fully similar ($E(x,y) = 1$) or fully dissimilar ($E(x,y) = 0$).

5.1.1 The Term Functor

Given two terms $(n, \omega, (m_i)_{i\leq n})$ and $(n', \omega', (m'_i)_{i\leq n'})$, we now need to obtain a similarity between them. Intuitively, we must join the similarity between ω and ω' with the combined similarities between respective components m_i and m'_i. In order to be more precise, let Ω be a set of operations, and let further E_Ω be a similarity on Ω, which will be used to define a similarity on TX.

Proposition 5.1 *Let $E_T: TX \times TX \longrightarrow L$ be a relation on TX such that*

$$E_T(x, x) = 1 \tag{5.1}$$

for all $x \in X$, and for terms $t = (n, \omega, (m_i)_{i \leq n})$, and $t' = (n', \omega', (m'_i)_{i \leq n'})$,

$$E_T(t, t') = \begin{cases} E_\Omega(\omega, \omega') \wedge \bigwedge_{i \leq n} E_T(m_i, m'_i) & \text{if } n = n' \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

Then E_T is a similarity on TX.

Note that the definition of similarity on TX adopted here differs from the one adopted in [59] in that the latter requires additionally that $E_T(x_1, x_2) = 0$ if $x_1 \neq x_2$.

5.1.2 The Powerset Term Functor

For unification purposes we will need a similarity between powersets of terms (generalised terms), and to this aim we will now use E_T in order to define a similarity on $2TX$. The choice of this similarity on $2TX$ is less obvious than the one on TX. For the similarity on $2TX$ we adopt the viewpoint of combining all similarities of pairwise terms from respective sets of terms.

Proposition 5.2 *Let E_T be a relation satisfying Conditions (5.1) and (5.2). The relation*

$$E_{2T}: 2TX \times 2TX \longrightarrow L$$

defined for all $M_1, M_2 \in 2TX$ as

$$E_{2T}(M_1, M_2) = \bigwedge_{m_1 \in M_1} \bigvee_{m_2 \in M_2} E_T(m_1, m_2) \wedge \bigwedge_{m_2 \in M_2} \bigvee_{m_1 \in M_1} E_T(m_1, m_2) \tag{5.3}$$

is a similarity on $2TX$.

Example 5.1 *Let $\Omega = \{c_1, c_2, u_1, u_2\}$, where c_i are nullary operators (constants), and u_i unary operators, $i = 1, 2$. Further, let x be a variable. Table 5.1 shows similarity values for some typical pairs of generalised terms.*

$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{E_{2T}(M_1, M_2)}$
$\{t_1\}$	$\{t_1, t_2, t_3\}$	$E_T(t_1, t_2) \wedge E_T(t_1, t_3)$
$\{u_1(c_1)\}$	$\{u_1(u_1(c_1))\}$	0
$\{u_1(c_1)\}$	$\{u_2(x)\}$	$E_\Omega(u_1, u_2) \wedge E_T(c_1, x)$
$\{u_1(c_1), u_1(c_2)\}$	$\{u_2(c_1), u_2(c_2)\}$	$E_\Omega(u_1, u_2)$
$\{u_1(c_1), u_2(c_2)\}$	$\{u_1(c_2), u_2(c_1)\}$	$E_\Omega(u_1, u_2) \vee E_\Omega(c_1, c_2)$

Table 5.1: Equality of generalised terms.

Regarding similarities in $2T$, let us recollect the approach used in [24] which introduced the *pseudosimilarity* below (it is not a similarity because it is not reflexive)

$$E'(M_1, M_2) = \bigwedge_{m_1, m_2 \in M_1 \cup M_2} E'_T(m_1, m_2).$$

however, the proposition above shows that there are reasonable similarities in $2T$.

5.1.3 The Term Powerset Functor

A similarity on $T2X$ can now easily be given using the similarity on $2TX$.

Proposition 5.3 *The relation*

$$E_{T2} \colon T2X \times T2X \longrightarrow L$$

defined as

$$E_{T2}(l_1, l_2) = E_{2T}\Big(\sigma_X(l_1), \sigma_X(l_2)\Big) \tag{5.4}$$

where $\sigma_X \colon T2X \longrightarrow 2TX$ is the swapper, is a similarity on $T2X$.

Note that, although we have the similarity of $T2$, it is still an open question whether the functor composition $T2$ can be extended to a monad.

5.2 Generalised Substitutions

In this section we recall the concepts of variable substitutions and unifiers, in particular within the context of powersets of terms. In the classical situation, variable substitutions are mappings assigning variables to terms, i.e. mappings $\theta\colon X \longrightarrow TY$.

For powersets of terms in a generalised setting, given a monad $\mathbf{\Phi}$ such that the composition $\mathbf{\Phi} \bullet \mathbf{T}_\Omega$ still provides a monad, variable substitutions should then be viewed as mappings

$$\theta\colon X \longrightarrow \Phi TY.$$

Given a term $M \in \Phi TX$ in form of a generalised powerset of terms, the result $M\theta$ of applying a variable substitution θ on M is given by

$$M\theta = (\mu_Y^{\Phi T} \circ \Phi T\theta)(M)$$

i.e. $M\theta$ is kind of a flattening of a set of terms over sets of terms, where $\mu_Y^{\Phi T}$ provides the flattening operation. Note that in the particular case of $\mathbf{\Phi}$ being the identity monad, $M\theta$ is nothing but the expected classical result when applying the variable substitution θ to the term $M \in TX$.

For $M_2 = \{u_2(x)\}$, in Table 5.1, note the effect of the variable substitutions $\theta(x) = c_1$ and $\theta(x) = u_1(c_1)$.

Variable substitutions can obviously be defined more generally over monads (F, η^F, μ^F). Indeed for an object $A \in FX$, and a variable substitution $\theta\colon X \longrightarrow FY$, we will have

$$A\theta = (\mu_Y^F \circ F\theta)(A).$$

When composing substitutions, the utility of the flattening operator again becomes explicit. The composition of two substitutions $\theta'\colon X \longrightarrow \Phi TY$ and $\theta''\colon Y \longrightarrow \Phi TZ$ is given by

$$\theta'\theta'' = \mu_Z^{\Phi T} \circ \Phi T\theta'' \circ \theta'$$

i.e. the composition in the Kleisli category $\mathtt{Set}_{\mathbf{\Phi T}}$ for the powerset monad $\mathbf{\Phi} \bullet \mathbf{T}$ over the category of sets.

For $\mathbf{\Phi}$ being the identity monad, variable substitutions correspond to the classical case, where use of the idempotency of 'terms over terms' is usually not made very explicit.

5.3 Unifiers

Let us continue with the case of $\mathbf{\Phi} = \mathbf{2}$. Given $M_1, M_2 \in 2TX$, let $[M_1; M_2]$ represent an *equation* over $2TX$. In order to propose a definition of generalised unifiers for equations we will assume the existence of similarities, as those in the previous section.

Definition 5.2 *A unifier of the equation* $[M_1; M_2]$ *over* $2TX$ *is a substitution,* $\theta\colon X \longrightarrow 2TY$*, such that* $E_{2T}(M_1\theta, M_2\theta)$ *equals*

$$\sup\{E_{2T}(M_1\vartheta, M_2\vartheta) \mid \vartheta \text{ is a substitution}\}.$$

It might be possible that the supremum above could not be attained by any substitution. The particular features of the lattice or the underlying application might require a weaker version of the definition, as follows:

Definition 5.3 *Let* θ *be a substitution, and* $[M_1; M_2]$ *an equation over* $2TX$*. We say that* θ *is a unifier if* $E_{2T}(M_1, M_2) \leq E_{2T}(M_1\theta, M_2\theta)$*, that is, if the substitution increases the similarity degree.*

Note that a unifier θ is a so called *extensional mapping* according to [39].

5.4 A 'Musical' Example

In this last section of the chapter, an informal example is included in order to provide illuminations of our notions. More examples can be found in Paper V.

A group of important musicians have been invited to play in an official event that will take place in a close future. But time will be limited and the musicians do not

E	Smashing P.	Nick Drake	Joao Gilberto	Bill Evans	Schumann
Smashing P.	1	0.8	0.8	0.3	0.3
Nick Drake	0.8	1	0.8	0.3	0.3
Joao Gilberto	0.8	0.8	1	0.3	0.3
Bill Evans	0.3	0.3	0.3	1	0.7
Schumann	0.3	0.3	0.3	0.7	1

Table 5.2: Similarities between songs.

have time to play one song each. As all of them are important, they must all play. The solution the organizers found was to group them according to the style of music of the musicians, so they can play together.

Of course, this task of grouping them was not an easy one and after consulting category theory experts, they found the following solution:

A representative number of songs (five, to simplify the example) were chosen with the intention of defining a similarity relation between the songs. After a rigorous study of some qualities of the songs (like tempo, rhythm, instrument used, voice qualities, etc), a similarity relation was proposed in Table 5.2.

According to [4], Schumann sonata and Bill Evans piece are similar because of their common romantic piano; A Nick Drake tune, an acoustic tune by Smashing Pumpkins and a Bossa Nova piece by Joao Gilberto are similar because they consist of a simple acoustic guitar and gentle male voice.

The organizers asked each of the musicians to identify their music style with a subset of 3 elements of the set of songs considered above. That was how each of the musicians was represented by a 'generalised term' i.e. the set of the songs he/she chose. For instance, $M = \{\text{Smashing P.}, \text{Bill Evans}, \text{Joao Gilberto}\}$ represents the musician that has chosen Smashing P., Bill Evans and Joao Gilberto pieces as a representative for his/her music style.

To make the groups was now a bit easier and the organizers just had to look at the similarity relations E_{2T} induced by the musicians. For instance consider the musicians

$M_1 = \{\text{Smashing P.}, \text{Joao Gilberto}, \text{Schumann}\}$,
$M_2 = \{\text{Nick Drake}, \text{Bill Evans}, \text{Schumann}\}$,
$M_3 = \{\text{Smashing P.}, \text{Nick Drake}, \text{Joao Gilberto}\}$.

The similarity levels $E_{2T}(M_1, M_2) = 0.7$ and $E_{2T}(M_1, M_3) = 0.3$ provide some information in which the organizers can base the grouping selection.

In the last minute, as one of the organizers was also a musician, the rest of the organizers thought it would be a good idea to allow him play together with some of the important musicians. The generated term describing the music style of the organizer is $O = \{\text{Smashing P.}, \text{Nick Drake}, \text{Schumann}\}$. The problem now is that none of the musicians that remained to be grouped chose Schumann sonata, but all of them chose both Smashing Pumpkins and Nick Drake's songs.

In order to be able to find a musician that matched the organizer's style, we will make use of a variable x. All free (not grouped) musicians are actually representable by the set $C = \{\text{Smashing P.}, \text{Nick Drake}, x\}$. The task now is to find a substitution θ that will allow us to find our final candidate $C\theta$ such that the similarity $E_{2T}(O, C\theta)$ is the best. A substitution like this corresponds to the concept of unifier and in this case $\theta(x) = \text{Schumann}$ increases the similarity level to 1. Nevertheless, as we said, Schumann sonata was not chosen, so we have to look at the substitutions that remain i.e. $\theta_1(x) = \text{Joao Gilberto}$ and $\theta_2(x) = \text{Bill Evans}$. The similarities in this cases are $E_{2T}(O, C\theta_1) = 0.3$ and $E_{2T}(O, C\theta_2) = 0.7$.

From this result we can finally choose musician

$$C\theta_2 = \{\text{Smashing P.}, \text{Nick Drake}, \text{Bill Evans}\}$$

as our final candidate.

Part II

This second part consists of two chapters. Chapter 6 presents a broad introduction to what is done in each of the papers included in this thesis. Chapter 7 provides the conclusions and presents open questions that need to be further studied.

Chapter 6

Summary of Papers

In this chapter a summary of the papers appearing in Part III, is included to provide guidance in the reading and understanding of them.

To ease the reading process, the papers are presented following a contents order rather than a chronological order.

6.1 Paper I

"***Set functors, L-fuzzy set categories and generalised terms***",
Computers and Mathematics with applications, 43 (2002), pp. 693-705.

This paper includes and extends results in [23], our beginning paper in this research area where we studied the composition of the powerset functor with the term functor and showed how the composition could be provided with the monad structure. As a consequence of the calculations and techniques used there, our first general result about techniques for monad compositions was obtained.

In Paper I, a number of set functors extending the crisp powerset functor together with their *extension principles* are introduced. By *extension principles* we mean the two possible generalisations of a mapping $f: X \longrightarrow Y$ where X, Y are sets, when

working in the fuzzy case according to an optimistic or pessimistic interpretation of the fuzziness degree, as we saw in Section 2.1.3 of this thesis.

L-fuzzy set categories are defined for each of these extended power set functors and the rationality of the extension principle is proved in the categorical sense, i.e. the associated L-fuzzy set categories are shown to be equivalent to the category of sets and mappings.

Finally, Paper I shows how each of these new set functors can be extended to be monads and, furthermore, how they provide an useful tool to build new monads when composed with the term monad.

6.2 Paper II

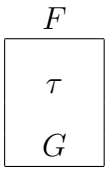
"***A graphical approach to monad compositions***",
In Electronic Notes in Theoretical Computer Science 40 (2001).

The foundational understanding of monads has been well-known for decades, but proof techniques, especially related to monad compositions have not been developed. As monad compositions are basically built upon operations of corresponding natural transformations, proof techniques require an adequate handling of the basic combinatorial properties of functors and natural transformations (*Godement rules*).

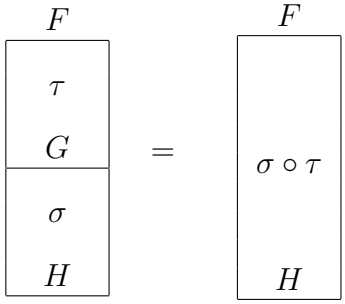
In [9, 25] it was discovered that combinatorial properties of functors and natural transformations can be represented more visually. This visual technique is not widely known and has been used mainly in purely algebraic contexts [5].

In this paper, this technique has been further developed in the scope of natural transformations and its compositions. As one may easily notice, when working with natural transformations, proofs become a succession of equations that could be difficult to follow. To improve the readability of this kind of expressions the visual representation is introduced in the paper and we demonstrate its use in some concrete examples on generalised terms where various set functors are composed with the conventional term functor.

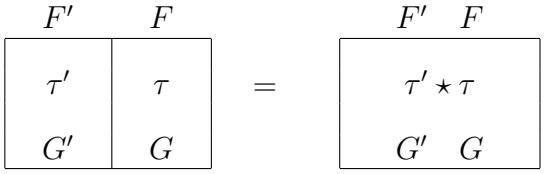
This technique is based on the visual representation of the *vertical and horizontal composition* of natural transformations. The pictorial representation is based on blocks representing natural transformations, i.e., a natural transformation $\tau : F \longrightarrow G$ as a basic building block is depicted as



Blocks $\tau : F \longrightarrow G$ and $\sigma : G \longrightarrow H$ are built, or composed, vertically by eliminating the horizontal line, e.g.:



For $\tau' : F' \longrightarrow G'$, horizontal block building is done similarly by eliminating the vertical line, e.g.:



Later in the paper the pictorial representation is used to demonstrate its use as proof support for monad compositions.

6.3 Paper III

"*Powersets of Terms and Composite Monads*",
Technical Report, UMINF 04.07, Department of Computing Science,
Umeå University, Sweden.

This paper could be considered in some respects as a summarization of parts of our research. The motivations for the paper are the connections presented there to Beck's *Distributive Laws*.

Distributive laws were introduced and studied, in terms of monads, by Beck in [8]. They provide a way of composing two algebraic structures (monads) into a more complex one. However, it is not evident how to provide composite monads in general.

In this paper we discuss the relation of our theorems for monad compositions with the definition of a distributive law as it was originally given by Beck. We show that both results are equivalent. A partially converse result is also presented here together with its relation to distributive laws.

It is not clear to what extent monads composed as functors are extendable to monads. In the case of complicated functors it is rather tempting intuitively to believe that monad compositions are not common simply by observing how the use of distributivity laws in proofs makes diagrams grow to sizes extremely difficult to overlook. On the other hand Beck says that proofs involving the distributivity law 'are just long naturality calculations'. It is the latter view that inspires us to provide a calculus for computing with natural transformations.

In this sense, by using the graphical calculus it is easy to obtain a set of equations which are sufficient conditions for defining a monad structure on the composition of two monads (Proposition 2 in the paper).

In Theorem 1 it is proved that conditions in Proposition 2 are equivalent to Beck's conditions on a distributive law as an easy application of the graphical calculus of natural transformations.

In section 6, we re-establish the well-known converse result to Proposition 2, using our graphical tool. Theorem 4 shows the connections between our theorem on 'reverse engineering of monad' (Theorem 3 in the paper) and the corresponding given by Beck. Again, the graphical representation permits us to give a straightforward proof which avoids the naturality calculations.

The connections presented in the paper between our results on monad compositions and Beck's distributive laws, provide different tools to check the composability of monads, e.g., depending on the example considered, one could choose the theorem in which the conditions are easier to check.

6.4 Paper IV

"***A categorical approach to unification of generalized terms***",
Electronic Notes in Theoretical Computer Science 66.5 (2002).

Paper IV begins with a review of how category theory is useful for unification in the classical case. The categorical language can identify and isolate trivial parts and therefore, universal parts of reasoning which make possible a direct concentration on the difficulties of the problem considered.

Basically, unification is the process of finding a substitution that makes two given terms equal. Some examples are presented to better understand and motivate the use of categorical structures in concrete situations.

As showed in [29], when solving a unification problem in the classical case, we are dealing with a concept of 'substitution system' which is suitably represented within a categorical framework. In [29] it is shown how the problem of unification appears in trivial examples, as for instance when trying to solve equations.

In a general way, a *substitution system* should have a set S of substitutions, a set $|S|$ of types, a partial composition operation on S together with source and target operations denoted $\delta_0, \delta_1 \colon S \longrightarrow |S|$ such that

(i) $\tau\sigma$ is defined iff $\delta_1(\sigma) = \delta_0(\tau)$.

(ii) $\delta_0(\tau\sigma) = \delta_0(\sigma)$ and $\delta_1(\tau\sigma) = \delta_1(\tau)$.

(iii) $\lambda(\tau\sigma) = (\lambda\tau)\sigma$ whenever all compositions are defined.

(iv) For each $T \in |S|$, there is a substitution id_T such that $\delta_0(id_T) = \delta_1(id_T) = T$, and further, $id_T\sigma = \sigma$ and $\tau id_T = \tau$ whenever these compositions are defined.

Categorically we would say 'map' or 'arrow' instead of 'substitution' and 'object' instead of 'type'. Thus we get a 'category' instead of a 'substitution system'.

In classical logic, substitutions can be seen as mappings $\sigma_1 \colon X \longrightarrow T_\Omega Y$, $\sigma_2 \colon Y \longrightarrow T_\Omega Z$ and its composition can be described as the conventional composition

$$X \xrightarrow{\sigma_1} T_\Omega Y \xrightarrow{T_\Omega \sigma_2} T_\Omega T_\Omega Z \xrightarrow{\mu_Z} T_\Omega Z$$

where μ is the 'flattening' operator $\mu_Z : T_\Omega T_\Omega Z \longrightarrow T_\Omega Z$, which in this case is the identity mapping because the functor T_Ω is idempotent. Categorically, μ is the natural transformation in the term monad (T_Ω, η, μ), and the composition given corresponds to composing morphisms in the corresponding so-called Kleisli category of the term monad. Goguen made the observation that the task of unification is exactly that of computing coequalizers in this Kleisli category. Further, the categorical unification algorithm obtained in this way is the same as Robinson's Algorithm used in classical logic to find the mgu (most general unifier). In [56] this situation was explored in detail together with implementation instructions in ML.

Categorical unification in classical logic involves substitutions of variables with terms. For non-classical logic, and with languages extended to include powersets of terms, the categorical approach turns out to be convenient. Techniques involving monads, which for classical terms is more trivial, now become explicit and powerful.

In a many-valued situation we need to include many-valuedness also when considering equalities between operators (and constants). Thus, dealing with many-valued unification, several generalisations should be considered, like those for substitutions, equalities and unifiers. What is the unification degree of $p(x)$ and $q(x)$ if the operators p and q are similar? What is the unification degree of $r(x, x)$ and $r(a, b)$ if the constants a and b are almost identical (in some sense)? In both cases the classical task of finding a unifier fails. However, in the many-valued case, these questions are

significant and solutions (unifiers) depend on how similarities are defined for operators and (powersets of) terms.

Adding the possibility to use powersets of terms and unifications thereof, involve further complications. Approaching the generalisation of terms implies then also to consider a generalised concept of substitutions, where variables are not replaced by terms but by various powersets of terms.

From this point of view, what happens if we want to substitute a variable by a set of terms, for instance $[x/\{t_1, t_3, t_6\}, y/\{t_2, t_3\}]$? A variable substitution should then be $\sigma : X \longrightarrow\!\!\!\!\!\rightarrow P \circ T_\Omega Y$ where P denotes the ordinary powerset functor. The powerset functor is extendable to a monad in the usual way.

In [23] we showed that $P \circ T_\Omega$ indeed is extendable to a monad. The result in Paper I is more general as we can use many-valued powerset functors. Thus the key to composing substitutions is again to consider composition of corresponding morphisms in the Kleisli category of the monad $P \circ T_\Omega$.

6.5 Paper V

"***Similarities between powersets of terms***",
Fuzzy Sets and Systems (2004). In Press.

Some steps towards unification are presented here. The paper represents both the 'end of this thesis' and the 'beginning of future work'. It is based on the use of previous results for unification purposes and provides a basis to continue investigating within these terms.

In this paper, we generalise a similarity frequently used between terms, that is, in the image of the functor T, to a similarity between standard sets of terms and between terms on sets of variables, interpreting the construction in the image of the functors PT and TP, respectively. This approach gives us the possibility of defining unifiers between generalised terms.

Given two terms $(n, \omega, (m_i)_{i\leq n})$ and $(n', \omega', (m'_i)_{i\leq n'})$, to obtain a similarity E_T between them, we must join the similarity between ω and ω' with the combined similarities between respective components m_i and m'_i. (This was seen in Chapter 5)

For unification purposes a similarity between powersets of terms is defined later in the paper, and for this purpose we use E_T in order to define a similarity on $2TX$. The choice of this similarity on $2TX$ is less obvious than the one on TX. For the similarity on $2TX$ we adopt the viewpoint of combining all similarities of pairwise terms from respective sets of terms, i.e.:

$$E_{2T}\colon 2TX \times 2TX \longrightarrow L$$

is defined for all $M_1, M_2 \in 2TX$ as

$$E_{2T}(M_1, M_2) = \bigwedge_{m_1\in M_1} \bigvee_{m_2\in M_2} E_T(m_1, m_2) \wedge \bigwedge_{m_2\in M_2} \bigvee_{m_1\in M_1} E_T(m_1, m_2)$$

Concepts of variable substitutions and unifiers need to be introduced, in particular within the context of powersets of terms. In the classical situation, variable substitutions are mappings assigning variables to terms, i.e. mappings $\theta\colon X \longrightarrow TY$.

For powersets of terms in a generalised setting, given a monad $\mathbf{\Phi}$ such that the composition $\mathbf{\Phi} \bullet \mathbf{T}_\Omega$ still provides a monad, variable substitutions should then be viewed as mappings

$$\theta\colon X \longrightarrow \Phi TY.$$

The composition of two substitutions $\theta'\colon X \longrightarrow \Phi TY$ and $\theta''\colon Y \longrightarrow \Phi TZ$ is given by

$$\theta'\theta'' = \mu_Z^{\Phi T} \circ \Phi T\theta'' \circ \theta'$$

i.e. the composition in the Kleisli category $\mathtt{Set}_{\mathbf{\Phi T}}$ for the powerset monad $\mathbf{\Phi} \bullet \mathbf{T}$ over the category of sets.

Finally a definition of generalised unifiers for equations is proposed: A unifier of the equation $[M_1; M_2]$ over $2TX$ is a substitution, $\theta\colon X \longrightarrow 2TY$, such that $E_{2T}(M_1\theta, M_2\theta)$ equals

$$\sup\{E_{2T}(M_1\vartheta, M_2\vartheta) \mid \vartheta \text{ is a substitution}\}.$$

It might be possible that the supremum above could not be attained by any substitution. The particular features of the lattice or the underlying application might require a weaker version of the definition, as follows:

Let θ be a substitution, and $[M_1; M_2]$ an equation over $2TX$. We say that θ is a unifier if $E_{2T}(M_1, M_2) \leq E_{2T}(M_1\theta, M_2\theta)$, that is, if the substitution increases the similarity degree.

In the last part of the paper some examples are included to illuminate the constructions and their use.

Chapter 7

Conclusions and Future Directions

This thesis has dealt with different aspects towards many-valued unification which have been studied in the scope of category theory. The main motivation of this investigation comes from the fact that in logic programming, classical unification has been identified as the provision of coequalizers in Kleisli categories of term monads [56]. Continuing in that direction, we have used categorical instruments to generalise the classical concept of a term. It is expected that this approach will provide an appropriate formal framework for useful developments of generalised terms as a basis for many-valued logic programming involving an extended notion of terms.

In this chapter we look back at the main contributions of this thesis and we indicate in what directions to continue this work.

7.1 Techniques for Monad Compositions

Monads and category theory in general, provide an abstract tool to handle properties of structures. It is somehow surprising how well an abstract framework (such as the one provided by monads) can fit an existing situation ([56]). It is precisely this fact what motivates us to continue in the abstract framework.

This work has focused on different theoretical aspects of the theory of monads,

specifically the study of conditions of composability of two monads, and the behavior of the substructures with respect to the composition.

There are no general methods on how to provide monad compositions, and it is important to stress the non-triviality of providing such compositions. An apparently useful composition of functors in corresponding monads can turn out not to be extendable to a monad. In Chapter 2 we have introduced various conditions for two monads being composable, and also shown how the structure of the multiplication of the composite monad can be determined under certain conditions.

Various approaches to composability of monads seem to indicate the necessity to view examples almost case by case. Further, proofs of compliance with composability conditions in particular cases of set functors tend to be rather complicated as the complexity of the functors increase. This is obvious e.g. for the double contravariant power set functor.

Given some main examples of monad compositions, we have shown how submonads provide a useful mechanism for constructing additional monad compositions from submonads of given main monads.

7.2 Generalised Terms

In order to develop a concept of generalised terms, we use composition of monads. Our particular focus has been on composing powerset monads with the term monad, therefore the different generalisation of the powerset functor become important, specially if they can be extended to a monad.

In Chapter 3 we introduced a number of set functors (L-fuzzy functors) extending the crisp powerset functor. We have generalised previous constructions of fuzzy set categories by considering L-fuzzy sets in which the values of the characteristic functions run on a completely distributive lattice. In addition, L-fuzzy categories have been defined, using L-fuzzy sets, and proved to be rational. We have proven that the L-fuzzy functors given by the extension principles can be extended to a monad

as well as the composition of the L-fuzzy functors with the term monad, and in this way provide a notion for generalised terms.

7.3 Visual Representations

Proof related to composition of monads make use of a lot of equations and therefore proofs become difficult to handle. A visual representation based on the pictorial representation of natural transformations and their composition was the focus of Chapter 4.

A graphical proof technique to provide evidence of compliance with composability conditions has shown to offer a more mechanised approach to handling complicated constructions involving calculations with natural transformations. We have shown how compositions and star products of natural transformations can be pictorially represented in order to provide proof support.

Proving composability conditions is complicated as the complexity of the functors increase. The graphical support is beneficial in that composability proofs are expected to reveal further examples of monad compositions that provide useful scenarios for generalised terms. Not only is the graphical approach a theoretical tool for a better understanding of the composition of natural transformations, but computing with natural transformations could be, to some extent, automatized and managed with such a graphical interface.

7.4 Towards Unification

In Chapter 5 we have shown how generalised terms, as given by powersets of terms, can be handled in equational settings involving substitutions and unifiers. The utility of categorical techniques as provided by monads is obvious and indeed encouraging for further investigations on more elaborate compositions and categorical techniques for unification as initiated in [56].

We have generalised a similarity frequently used between terms, that is, in the image of the functor T, to a similarity between standard sets of terms and between terms on sets of variables, interpreting the construction in the image of the functors PT and TP, respectively; this approach gives us the possibility of defining unifiers between generalised terms.

7.5 Future Work

Handling conditions for monad compositions involve manipulations of rather complicated expressions involving natural transformations, and as it is important to continue investigations on how to construct new monads from given ones, the graphical approach is expected to support proof procedures in this context.

Within the scope of many-valued logic programming, it is important to further investigate the possibilities of using categorical approaches to unification.

We have seen how generalised terms can be handled in equational settings involving substitutions and unifiers. Further work is needed to develop unification algorithms. It is also interesting to merge our efforts with developments, such as in [3], that have focused more on semantic aspects of many-valued logic programming. Semantic approaches seem to be fruitful in particular within possibilistic logic frameworks. These developments, however, still have a rather specialised use of terms as they typically restrict to using powersets of constants instead of generalised terms in their full range. However, restricting to powersets of constants seems more to be a struggle with unification than with proof procedural issues, and there are no indications that the specialised use of terms is enforced by the semantic developments, such as those seen in possibilistic logic. While the procedural issues are important it is equally worthwhile to underline the importance of further studies on monad compositions.

7.6 Related Work

Monads have gained considerable acceptance in the development of denotational semantics for programming languages, both in declarative and imperative cases. This indicates again how category theory is useful in practical applications in computer science.

The view that monads provide abstract mathematical interpretations of computational phenomena led Moggi [53] to use the internal language of a category with a strong monad, which he called the computational λ-calculus, for describing denotational semantics of programming languages. In [35] the denotational semantics of the Java composition is interpreted as the Kleisli composition, and the Java extension to its extension. In [10], the categorical interpretation of the composition of *call-by-value* programs is done in the Kleisli category of a monad T, whereas *call-by-name* programs compose in the base category, and develops a monadic categorical construction to cope with the composition of *call-by-need* programs. In addition, there is a translation scheme from λ-calculus into an arbitrary monad; indeed, there are two schemes, one yielding call-by-value semantics and one yielding call-by-name [63].

Monads provide a way of structuring functional programs. Structuring programs like parsers, type checkers or interpreters, it is often the case that the monad needed is a combination of several ones. Some examples of application of the compositions of monads, in a functional programming setting, can be seen in [38]. Some conditions about the composability of two given monads have been studied in [36].

7.6.1 Unification

Several approaches to the fuzzy unification problem can be found in the literature.

In [3], the fuzzy unification within a possibilistic framework, requires a fix domain and an interpretation of fuzzy propositional variables in terms of its membership functions in order to identify when two propositional variables A and B are related and to reason about the certainty of the propositional variable B from the fact that

the domain-related propositional variable A is certain with a necessity of at least α.

In [24], the unification process is "softened" by admitting that two first-order expressions can be "similar" up to a certain degree and not necessarily identical. Inspired by the unification algorithm of Martelli-Montanari ([50]), the systems of equations go through a series of "sound" transformations until a solvable form is found yielding a substitution that is proved to be a most general extended unifier for the given system of equations.

Bibliography

[1] Abramsky, S.: 2001, 'Algorithmic Game Semantics: a Tutorial Introduction'. In: *Proc. of Marktoberdorf 2001*.

[2] Adámek, J., H. Herrlich, and G. Strecker: 1990, *Abstract and Concrete Categories*. John Wiley & Sons.

[3] Alsinet, T. and L. Godo: 2000, 'A complete calculus for possibilistic logic programming with fuzzy propositional variables'. In: *Proc. Uncertainty in AI'00 conference*. pp. 1–10.

[4] Aucouturier, J. and F. Pachet: 2002, 'Music Similarity Measures: What's the Use?'. In: Ircam (ed.): *Proceedings of the 3rd International Symposium on Music Information Retrieval*.

[5] Barja Pérez, J., 'Personal communication'. April, 2000.

[6] Barr, M. and C. Wells: 1985, *Toposes, Triples and Theories*. Springer.

[7] Barr, M. and C. Wells: 1990, *Category Theory for Computing Science*. Prentice Hall.

[8] Beck, J.: 1969, 'Distributive Laws'. In: *Seminar on Triples and Categorical Homology Theory*, No. 80 in Lecture Notes in Mathematics. Springer-Verlag, pp. 119–130.

[9] Caruncho Castro, J.: 1971, 'Triples theory'. Technical Report 5, Department of Algebra of the Univ. de Santiago de Compostela. In Spanish.

[10] Cenciarelli, P.: 1998, 'An algebraic view of program composition'. In: *Proc. Algebraic Methodology and Software Technology*. pp. 325–340.

[11] Eilenberg, S. and J. Moore: 1965, 'Adjoint functors and triples'. *J. Math* **9**, 381–398.

[12] Eklund, P. and W. Gähler: 1990, 'Generalized Cauchy spaces'. *Math. Nachr.* **147**, 219–233.

[13] Eklund, P. and W. Gähler: 1992, 'Fuzzy filter functors and convergence'. In: S. E. Rodabaugh et al. (ed.): *Applications of category theory to fuzzy subsets*, Theory and Decision Library B. Kluwer, pp. 109–136.

[14] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2000, 'Generalised terms and composition of monads'. In: *Proc. of ESTYLF 2000, Spanish Congress on Fuzzy Logic and Technology*. pp. 155–160.

[15] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2001, 'Composing submonads'. In: *Proc. of ISMVL 2001, 31st IEEE Int. Symp. on Multiple-Valued Logic*. pp. 367–372.

[16] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2001, 'A graphical approach to monad compositions'. *Electronic Notes in Theoretical Computer Science* **40**.

[17] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2001, 'Set functors, L-fuzzy set categories: towards a fuzzy programming paradigm'. In: *Proc. of FLA 2001, International ICSC Symposium on Fuzzy Logic and Applications*. pp. 658–664.

[18] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2002, 'A categorical approach to unification of generalised terms'. *Electronic Notes in Theoretical Computer Science* **66**(5).

[19] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2002, 'A framework for unification using powersets of terms'. In: *Proc. of IPMU 2002*, Vol. III of *Information Processing and Management of Uncertainty in Knowledge-based Systems*. pp. 1095–1098.

[20] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2002, 'Set functors, L-fuzzy set categories and generalized terms'. *Computers and Mathematics with Applications* **43**, 693–705.

[21] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: 2004, 'Powersets of Terms and Composite Monads'. Technical Report UMINF-04.07, Department of Computing Science, Umeå University, Sweden.

[22] Eklund, P., M. A. Galán, J. Medina, M. Ojeda-Aciego, and A. Valverde: In Press 2004, 'Similarities between powersets of terms'. *Fuzzy Sets and Systems*.

[23] Eklund, P., M. A. Galán, M. Ojeda-Aciego, and A. Valverde: 2000, 'Set functors and generalised terms'. In: *Proc. of IPMU 2000*, Vol. III of *Information Processing and Management of Uncertainty in Knowledge-based Systems*. pp. 1595–1599.

[24] Formato, F., G. Gerla, and M. I. Sessa: 2000, 'Similarity-based unification'. *Fundamenta Informaticae* **41**, 393–414.

[25] Freire Nistal, J.: 1971, 'Universal properties in higher order triples'. Technical Report 11, Department of Algebra of the Univ. de Santiago de Compostela. In Spanish.

[26] Gähler, W.: 1981, 'A topological approach to structure theory'. *Math. Nachr.* **100**, 93–144.

[27] Gähler, W.: 1988, 'Monads and convergence'. In: *Proc. Conference Generalized Functions, Convergences Structures, and Their Applications*. pp. 29–46.

[28] Goguen, J.: 1967, 'L-fuzzy sets'. *J. Math. Anal. Appl.* **18**, 145–174.

[29] Goguen, J.: 1989, 'What is Unification? A Categorical View of Substitution, Equation and Solution'. In: M. Nivat and H. Ait-Kaci (eds.): *Algebraic Techniques*, Vol. 1 of *Resolution of Equations in Algebraic Structures*. Academic Press, pp. 217–261.

[30] Goguen, J.: 1991, 'A Categorical Manifesto'. *Mathematical Structures in Computer Science* **1**(1), 49–67.

[31] Gray, J. W.: 1974, *Formal Category Theory: Adjointness for 2-Categories*, No. 391 in Lecture Notes in Mathematics. Springer-Verlag.

[32] Hájek, P.: 1998, *Metamathematics of Fuzzy Logic*. Kluwer Academic Publishers.

[33] He, Q., H.-X. Li, C. Chen, and E. Lee: 2000, 'Extension principles and fuzzy set categories'. *Computers and Mathematics with Applications* **39**, 45–53.

[34] Höhle, U.: 1995, 'Presheaves over GL-monoids'. In: *Non-classical Logics and their Applications to Fuzzy Subsets*. Kluwer Academic Publisher, pp. 127–157.

[35] Jacobs, B. and E. Poll: 2000, 'A monad for basic Java semantics'. In: *Proc. Algebraic Methodology and Software Technology*. pp. 150–164.

[36] Jones, M. P. and L. Duponcheel: 1993, 'Composing monads'. Technical Report YALEU/DCS/RR-1004, Yale University.

[37] Kelly, G.: 1974, 'Coherence theorems for lax algebras and for distributive laws'. In: *Category Theory: Proceedings Sydney Category Theory Seminar 1972/1973*, No. 420 in Lecture Notes in Mathematics. Springer-Verlag, pp. 281–375.

[38] King, D. J. and P. Wadler: 1992, 'Combining monads'. In: *Glasgow workshop on Functional Programming*.

[39] Klawonn, F., J. Gebhardt, and R. Kruse: 1995, 'Fuzzy Control on the Basis of Equality Relations - with an Example from Idle Speed Control'. *IEEE Trans. Fuzzy Systems* **3**, 336–350.

[40] Klawonn, F. and R. Kruse: 1994, 'A Łukasiewicz logic based PROLOG'. *Mathware & Soft Comp.* **1**, 5–29.

[41] Kleisli, H.: 1965, 'Every standard construction is induced by a pair of adjoint functors'. In: *Proc. Amer. Math. Soc. 16.* pp. 544–546.

[42] Kowalsky, H.-J.: 1954, 'Limesräume und Komplettierung'. *Math. Nachr.* **12**, 301–340.

[43] Lawvere, F.: 1963, *Functorial semantics of algebraic theories*, Dissertation. Columbia University.

[44] Loeckx, J., H.-D. Ehrich, and M. Wolf: 1996, *Specification of Abstract Data Types.* Wiley-Teubner.

[45] Lüth, C. and N. Ghani: 1997, 'Monads and modular term rewriting'. In: *Category Theory and Computer Science.* Lecture Notes in Computer Science 1290, Springer, pp. 69–86.

[46] Manes, E.: 1976, *Algebraic Theories.* Springer-Verlag.

[47] Manes, E.: 1998, 'Implementing collection classes with monads'. *Mathematical Structures in Computer Science* **8**, 231–276.

[48] Marmolejo, F.: 1999, 'Distributive laws for pseudomonads'. *Theory and Applications of Categories* **5**(5), 91–147.

[49] Marmolejo, F., R. Rosebrugh, and R. Wood: 2002, 'A basic distributive law'. *Journal of Pure and Applied Algebra* **168**, 209–226.

[50] Martelli, A. and U. Montanari: 1982, 'An Efficient Unification Algorithm'. *ACM Transactions on Programming Languages and Systems* **4**, 258–282.

[51] Mitchell, B.: 1965, *Theory of categories*. Academic Press.

[52] Moggi, E.: 1989, 'Computational lambda-calculus and monads'. In: *In IEEE Symposium on Logic in Computer Science*.

[53] Moggi, E.: 1991, 'Notions of computation and monads'. *Information and Computation* **93**(1), 55–92.

[54] Nygaard, M. and G. Winskel: 2003, 'Domain Theory for Concurrency'. To appear in Theoretical Computer Science special issue on domain theory.

[55] Robinson, J.: 1965, 'A machine-oriented logic based on the resolution principle'. *J. Association for Computing Machinery* **12**, 23–41.

[56] Rydeheard, D. and R. Burstall: 1986, 'A categorical unification algorithm'. In: *Proc. Category Theory and Computer Programming*. Lecture Notes in Computer Science 240, Springer, pp. 493–505.

[57] Rydeheard, D. and R. Burstall: 1988, *Computational Category Theory*. Prentice Hall.

[58] Rydeheard, D. and J. Stell: 1987, 'A categorical treatment of equational proofs and unification algorithms'. In: *Category Theory and Computer Science*. Lecture Notes in Computer Science 283, Springer, pp. 114–139.

[59] Sessa, M. I.: 2001, 'Translations and similarity-based logic programming'. *Soft Computing* **5**, 160–170.

[60] Siekmann, J. H.: 1989, 'Unification theory'. *J. Symb. Comput.* **7**(3-4), 207–274.

[61] Stout, L.: 1991, 'A survey of fuzzy set and topos theory'. *Fuzzy Sets and Systems* **42**, 3–14.

[62] Street, R.: 1972, 'The formal theory of monads'. *Journal of Pure and Applied Algebra* **2**, 149–168.

[63] Wadler, P.: 1992, 'Comprehending monads'. *Mathematical Structures in Computer Science* **2**, 461–493.

[64] Wang, P. Z.: 1981, 'Fuzzy sets and fuzzy set categories, (in Chinese)'. *Advances in Mathematics* **11**(1), 1–18.

Index

Papers I-V

In this last part of the thesis, the following papers[1] can be found:

(I) Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Set functors, L-fuzzy set categories and generalised terms*", Computers and Mathematics with Applications, 43 (2002), pp. 693-705.

(II) Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*A graphical approach to monad compositions*", In Electronic Notes in Theoretical Computer Science 40 (2001).

(III) Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Powersets of Terms and Composite Monads*", Technical Report UMINF-04.07, 2004. Department of Computing Science, Umeå University, Sweden.

(IV) Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*A categorical approach to unification of generalised terms*", Electronic Notes in Theoretical Computer Science 66.5 (2002).

(V) Eklund, P., Galán, M.A., Medina, J., Ojeda-Aciego, M. and Valverde, A. "*Similarities between powersets of terms*", Fuzzy Sets and Systems (2003). In press.

[1] Papers I, II, IV and V reprinted with permission from Elsevier.