TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

Technische Berichte
Technical Reports

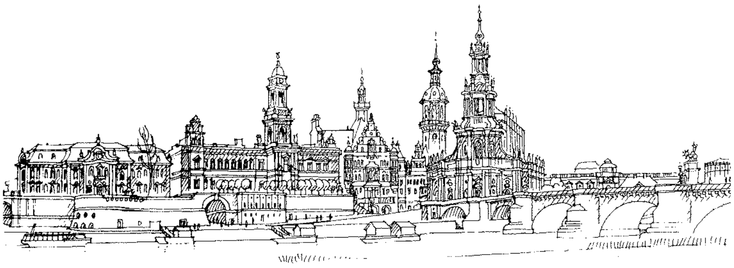
ISSN 1430-211X

TUD-FI00-13 - November 2000

Claus Jürgensen

Institut für Theoretische Informatik

A Formalization of Hylomorphism Based Deforestation with an Application to an Extended Typed λ-Calculus



Technische Universität Dresden
Fakultät Informatik
D-01062 Dresden
Germany

URL: http://www.inf.tu-dresden.de/

A Formalization of Hylomorphism Based Deforestation with an Application to an Extended Typed λ-Calculus

Claus Jürgensen*
Department of Computer Science
Dresden University of Technology
D-01062 Dresden, Germany
mailto:Claus@Orchid.Inf.TU-Dresden.DE
http://wwwtcs.inf.tu-dresden.de/~claus/

November 17, 2000

Abstract

We develop a new approach to prove the acid rain theorem (for cata-, ana- and hylomorphisms) and some generalizations, using the concept of concrete functors. Furthermore we introduce a categorical denotational semantics for an extended typed λ-calculus which is similar to Core Haskell, such that this semantics is a functor. This enables us to conveniently apply our generalized acid rain theorem to the language, to derive fusion-rules, which are more powerful than those of the well-known deforestation algorithms.

*This work was partially supported by the DFG project 'Specification of discrete processes and systems of processes by operational models and logics' (Graduiertenkolleg GRK 334/2)

Contents

1 Introduction

Using functional languages [Hud89], it is a good style of programming to define simple functions in order to combine them to more complicated functions and finally to the desired program. This is known as the compositional programming style [Hug89]. It leads to easy to read and maintain source-code. Unfortunately, these kinds of programs often are inefficient, because functions need time to calculate and memory to store the intermediate data, which is transfered from one function to another. This is a problem in particular for recursive functions and recursive data types where the intermediate values are trees.

In the case of a lazy language, *i.e.* a language with a call by need semantics, we still have a problem, since an unnecessary constructor-application is performed, *i.e.* an unnecessary encapsulation of the intermediate values into a recursive data structure. It is necessary to allocate memory for a cell to contain a reference of the actual value (*i.e.* a pointer to a subtree or a leaf) and thus time is needed for dereferentiation and memory allocation.

An automatic optimization of the interface between tree producing and tree consuming functions is called a **deforestation** algorithm [Wad90].

We focus on deforestation techniques using *fold/unfold* [GLP93, SF93] and in particular *catamorphism/anamorphism* [TM95], where a *fold* is a generalization of the well-known list function `foldr` for arbitrary recursive types, and a *catamorphism* is a mathematical abstraction of a *fold* in category theory.

The basis for deforestation using category theory is the following well-known *acid rain theorem* or *catamorphism-fusion* [TM95]

$$\frac{g : \forall A.(\mathsf{G}A \to A) \to B \to A}{(\!|\varphi|\!)_{\mathsf{G}} \cdot g\, in_{\mathsf{G}} = g\, \varphi}$$

where G is a functor and $(\!|\varphi|\!)_{\mathsf{G}}$ the catamorphism induced by the G-algebra φ and in_{G} is the G-constructor[1]. This is the category theory version of the *fold-build-rule* [LS95], in which $\mathtt{build}^{\mu\mathsf{G}}\, g = g\, in_{\mathsf{G}}$. In order to generalize the acid rain theorem, we will make it a little more abstract using a *concrete functor* H:

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}\,\mathsf{F}, |\,\centerdot\,|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}\,\mathsf{G}, |\,\centerdot\,|_{\mathsf{G}})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}$$

The explicit *forgetful functors* $|\,\centerdot\,|_{\mathsf{F}}$ and $|\,\centerdot\,|_{\mathsf{G}}$ of the *concrete categories* $(\boldsymbol{\mathcal{A}lg}\,\mathsf{F}, |\,\centerdot\,|_{\mathsf{F}})$ and $(\boldsymbol{\mathcal{A}lg}\,\mathsf{G}, |\,\centerdot\,|_{\mathsf{G}})$ enable us to derive generalizations like *e.g.*

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}\,\mathsf{F}, |\,\centerdot\,|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}\,\mathsf{G}, \mathsf{E} \cdot |\,\centerdot\,|_{\mathsf{G}})}{\mathsf{E}(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}$$

for faithful functors E. Hence we can handle fusions of catamorphisms, where something (*e.g.* the E above) is in between the composition.

By duality, our results are also true for anamorphisms and thus for specific hylomorphisms[2], since the latter are compositions of a cata- and an anamorphism.

Many publications on program transformations by means of category theory (*cf. e.g.* [HIT96, LS95, TM95]) use the intuition that typed functional languages are categories, where the programs are the morphisms and the types are objects of the category, without specifying a precise categorical denotational semantics.

[1] The reader who is unfamiliar with the above notions and/or the notations in these formulas, may ignore them: Later, they will be explained in detail.

[2] The reader who likes to see hylomorphism based deforestation in practice, can find an implementation at `http://www.ipl.t.u-tokyo.ac.jp/~onoue/hylo` by Yoshiyuki Onue. It is possible to upload Haskell programs and to download the result of the deforestation algorithm.

We will provide a categorical denotational semantics for an extended typed λ-calculus. This semantics is a functor

$$\mathsf{S} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\Lambda}$$

where $\boldsymbol{\mathcal{C}}$ is a category with specific properties, as we will see later, and $\boldsymbol{\Lambda}$ is the category of types and typed λ-terms, where the composition is defined by substitution.

This enables us to treat types (as objects) and terms (as morphisms) consistently, which is important, since types are critical to construct hylomorphisms (or cata-, anamorphisms as special cases) (*cf.* [LS95]).

The structure of this paper is as follows: In Section ...

2 we collect the necessary mathematical notions and notations. The basic ones (in particular *ω-complete (chain-complete) partial order (CPO)*) are collected in Subsection 2.1. The reader who is familiar with the notions *(quasi-)category, morphism, object, (full) subcategory, (faithful) functor, (natural) transformation, initial/final object, (co-)product, exponent, (initial/final) (co-)algebra, cata- and anamorphism* may skip Subsection 2.2.

3 we briefly describe *concrete category/functor, construct, enriched category, $\boldsymbol{\mathcal{V}}$-functor, involutory category.*

4 we state our main result (Theorem 4.1.1) which enables us to derive calculation-rules for various categorical constructs. One corollary is the *acid rain theorem* and generalizations thereof (Corollaries 4.3.2, 4.3.4 and Theorem 4.3.5). Since (co-)product functors occur first in Subsection 4.2, we introduce the notions *Cartesian (closed) category (CCC)* and *internal fixed point operator of a CCC* here.

5 we define the notions *pointed category* and *polynomial functor* and finally we define a *hylomorphism* on a $\boldsymbol{\mathcal{C}po}$*-category* as a specific fixed point operator. Furthermore we show the relation between hylo-, cata- and anamorphisms and obtain a hylomorphism-version of the *acid rain theorem.*

6 we introduce an extended typed λ-calculus, quite similar to Core Haskell [PH99], and an according operational semantics and develop a sound categorical denotational semantics for this language. This semantics is a functor and enables us to apply generalized acid rain theorems to the extended typed λ-calculus in such a way that we derive sound formal program transformations for deforestation. We demonstrate the power of our program transformations by some example Haskell programs.

7 we outline some ideas for further research.

2 Preliminaries

2.1 Basic Notions and Notations

We recall and collect some notations, basic definitions and terminology, which will be used throughout this paper.

2.1.1. Definition (inference rule). To write down a logical inference, we will often use a so called **inference rule**:

$$\frac{A}{B} \qquad \text{for} \qquad A \implies B.$$

If a statement is a conjunction, we omit the 'and'-symbols ($\wedge$) and simply write:

$$\frac{A_1 \cdots A_m}{B_1 \cdots B_n} \qquad \text{for} \qquad A_1 \wedge \cdots \wedge A_m \implies B_1 \wedge \cdots \wedge B_n.$$

A mathematical theorem may thus be written in the clear form:

$$\frac{\text{Precondition(s)}}{\text{Proposition(s)}}.$$

In order to make this notation even simpler, we will omit leading universal quantifiers ($\forall$), keeping in mind that it denotes its universal closure.

We will also use inference rules to notate definitions:

$$\frac{B}{A} \qquad \text{for} \qquad A :\iff B.$$

In particular, for the definition of relations we will use so called **inference rule systems**. We define a relation $R \subseteq M$ on a set M by

$$\frac{P_1(R)}{Q_1(R)} \quad \cdots \quad \frac{P_n(R)}{Q_n(R)}$$

where P_i, Q_i are predicates for all i. This is short for

$$R := \bigcap \Big\{ S \subseteq M \;\Big|\; \forall i \in \{1, \ldots, n\} :: \big(P_i(S) \implies Q_i(S)\big) \Big\},$$

i.e. R is the smallest relation w.r.t. $\subseteq$, which satisfies all rules.

2.1.2. Definition (partial order, (pre-)order, equivalence relation). Let M be a set. A subset $R \subseteq M \times M$ is called a (binary) **relation** on M.

The relation R is called:	provided that:
reflexive	$(a,a) \in R \quad \forall a \in M$
transitive	$\dfrac{(a,b) \in R \quad (b,c) \in R}{(a,c) \in R}$
symmetric	$\dfrac{(a,b) \in R}{(b,a) \in R}$
antisymmetric	$\dfrac{(a,b) \in R \quad (b,a) \in R}{a = b}$
total	$(a,b) \in R \vee (b,a) \in R \quad \forall a,b \in M$

A reflexive and transitive relation is called **preorder**. A symmetric preorder is called **equivalence relation**, and an antisymmetric preorder is called **partial order**[3].

Obviously, a preorder R, which is symmetric as well as antisymmetric, has to be $\Delta_M = \{(a,a) | a \in M\}$, which is called the **diagonal** of M.

The pair (M, R) consisting of a set M and a partial order $R \subseteq M \times M$ is called **partially ordered set (poset)** or just **partial order**. A partial order is called **order**[3], if it is total.

For a binary relation R and $a, b \in M$ we will write $a\,R\,b$ for $(a,b) \in R$.

[3]Particularly in elder literature one finds the denotations 'order' and 'total order' for 'partial order' and 'order', respectively.

2.1.3. Definition (equivalence class, factor set). Let M be a set and R be an equivalence relation on M. For every $a \in M$ the set

$$[a]_R := \{b \in M \mid (a,b) \in R\}$$

is called the **equivalence class** of a w.r.t. R in M. The set

$${}^{X}\!/_{R} := \{[a]_R \mid a \in M\}$$

is called the **factor set** of M by R.

2.1.4. Definition and Lemma (monotone function). Let $A = (|A|, \sqsubseteq_A)$ and $B = (|B|, \sqsubseteq_B)$ be posets. A function $f : |A| \leftarrow |B|$ with

$$\frac{a \sqsubseteq_B b}{fa \sqsubseteq_A fb}$$

is called **monotone** and we write:

$$f : A \leftarrow B.$$

The composition of monotone functions as well as the identity function are monotone.

2.1.5. Definition (supremum). Let $A = (|A|, \sqsubseteq)$ be a poset and $S \subseteq |A|$. An element $a \in |A|$ is called **lower bound** (or **bottom**) of S, if

$$a \sqsubseteq s \quad \forall\, s \in S.$$

If the lower bound is an element of S, it is called **minimum** of S. Since it is uniquely determined, due to the antisymmetry of $\sqsubseteq$, we denote the minimum of S by $\min_{\sqsubseteq} S$ or just $\min S$. We can define a partial function $\min : |A| \leftarrow\cdots\ \boldsymbol{\mathcal{P}ot}\,|A|$, which maps each subset of $|A|$ onto its minimum, if it exists.

Vice versa an element $a \in |A|$ is called an **upper bound** (or **top**) of S, if

$$s \sqsubseteq a \quad \forall\, s \in S,$$

which we will write $S \sqsubseteq a$. If S has an upper bound, the set of all upper bounds is nonempty and has itself a lower bound (any element of S). If this set has even a minimum, *i.e.* a least upper bound, this is called the **supremum** (or **join**) of S. We define the partial function $\bigsqcup$ ('supremum of') by

$$\bigsqcup : \begin{cases} |A| & \leftarrow\cdots\ \boldsymbol{\mathcal{P}ot}\,|A| \\ \min\{a \in |A| \mid S \sqsubseteq a\} & \longleftarrow\!\!\shortmid \quad S \qquad \forall\, S \subseteq |A|_{\substack{\text{for which the} \\ \text{min. on the left exists}}}. \end{cases}$$

2.1.6. Definition (ω-chain). For every poset $A = (|A|, \sqsubseteq)$ we define by

$$A^\omega := \{(a_i)_{i=0}^\infty \in |A|^{\mathbb{N}_0} \mid a_i \sqsubseteq a_{i+1} \quad \forall\, i \in \mathbb{N}_0\}$$

the set of all **ω-chains** of A. We define the supremum

$$\bigsqcup : |A| \leftarrow\cdots\ A^\omega$$

of an ω-chain $a = (a_i)_{i=0}^\infty \in A^\omega$, if it exists, by

$$\bigsqcup_{i=0}^{\infty} a_i := \bigsqcup_{i \in \mathbb{N}_0} a_i := \bigsqcup a := \bigsqcup\{a_i \mid i \in \mathbb{N}_0\}.$$

Let $B = (|B|, \sqsubseteq_B)$ be a poset. For any monotone function

$$f : A \leftarrow B$$

we define the function

$$f^\omega : \begin{cases} A^\omega & \leftarrow \quad B^\omega \\ (f b_i)_{i=0}^\infty & \leftarrow\!\!\shortmid \quad (b_i)_{i=0}^\infty \qquad \forall\, (b_i)_{i=0}^\infty \in B^\omega . \end{cases}$$

It is easy to see that the image of f^ω is indeed a subset of A^ω, because f is monotone.

2.1.7. Definition (complete partial order). A triple

$$A = (|A|, \sqsubseteq, \bot)$$

consisting of a set $|A|$, a relation $(\sqsubseteq) \subseteq |A| \times |A|$ and an element $\bot \in |A|$ is called **complete partial order (CPO)**, provided that

$$(|A|, \sqsubseteq) \text{ is a poset} \tag{poset}$$

$$\bot = \min |A| \tag{minimum}$$

$$\bigsqcup : |A| \leftarrow A^\omega \tag{completeness}$$

The element $\bot$ is called the **least element** of A.

2.1.8. Note. In the literature one can find differing or more general definitions for CPOs:

- Sometimes so called **directed** sets, which can be incountable, are used instead of ω-chains. In this context, what we call a CPO would be called an ω-CPO or **chain-complete partial order (cCPO)**. Since we will only treat suprema of sequences (*cf.* Definition 2.1.12 and Theorem 5.4.3), we can content ourselves with the countable ω-chains.
- Sometimes the existence of a least element (minimum) is not part of the definition. In this context, what we call a CPO would be called a **complete pointed partial order (CPPO)**. We will need the least element to demonstrate the existence of the least fixed points (*cf.* Definition 2.1.12) and finally the existence of the hylomorphisms (*cf.* Definition 5.3.1).

So in fact, we use CPO as an abbreviation for ω-chain-complete pointed partial order.

2.1.9. Definition (ω-continuous, strict). Let $A = (|A|, \sqsubseteq_A, \bot_A)$ and $B = (|B|, \sqsubseteq_B, \bot_B)$ be CPOs. A function

$$f : |A| \leftarrow |B|$$

is called **ω-continuous**, if it is *monotone* and

$$f\left(\bigsqcup\nolimits_B S\right) = \bigsqcup\nolimits_A (f^\omega S) \quad \forall\, S \in B^\omega .$$

Then we write

$$f : A \leftarrow B.$$

We define the set of all ω-continuous functions to A from B by

$$\boldsymbol{\mathcal{C}po}(A, B) := \{ f : |A| \leftarrow |B| \mid f \text{ ist } \omega\text{-continous} \} = \{ f \mid f : A \leftarrow B \}.$$

If in addition

$$f\bot_B = \bot_A$$

holds, then f is called **strict**. We define the set of all strict ω-continuous functions to A from B by

$$\boldsymbol{\mathcal{C}po}_\bot(A, B) := \{f \in \boldsymbol{\mathcal{C}po}(A, B) \mid f\bot_B = \bot_A\}.$$

2.1.10. Lemma. The composition of two ω-continuous (strict) functions and the identity function are ω-continuous (strict).

2.1.11. Definition ((least) fixed point). Let M be a set and $f : M \leftarrow M$ a function. An element $m \in M$ is called a **fixed point** of f, provided that

$$fm = m.$$

Let $(\sqsubseteq) \subseteq M \times M$ be a poset. If the set $\{n \in M \mid fn = n\}$ has a minimum, then it is called the **least fixed point** of f.

2.1.12. Theorem (Tarski's fixed point theorem [BdM97]). For a CPO $A = (|A|, \sqsubseteq, \bot)$ we define the **fixed point operator**

$$\mathrm{fix}_A : \begin{cases} |A| & \leftarrow \ \boldsymbol{\mathcal{C}po}(A, A) \\ \bigsqcup_{k=0}^{\infty} f^k\bot & \leftarrow\!\shortmid \quad f \qquad \forall f \in \boldsymbol{\mathcal{C}po}(A, A), \end{cases}$$

which we will denote simply by fix instead of fix_A.

Then the following holds: Every $f : A \leftarrow A$ has a least fixed point and its value is $\mathrm{fix}\, f$, *i.e.*

$$\mathrm{fix}_A\, f = \min\{a \in A \mid fa = a\} \quad \forall f : A \leftarrow A.$$

2.1.13. Definition and Lemma (product of CPOs). For every CPOs $A = (|A|, \sqsubseteq_A, \bot_A)$ and $B = (|B|, \sqsubseteq_B, \bot_B)$ we define the **product**

$$A \times B := (|A| \times |B|, \sqsubseteq, \bot)$$

by

$$\frac{a \sqsubseteq_A a' \qquad b \sqsubseteq_B b'}{(a, b) \sqsubseteq (a', b')}$$

and

$$\bot := (\bot_A, \bot_B).$$

The functions

$$\begin{array}{ccccc} |A| & \xleftarrow[\grave{\pi}]{} & |A \times B| & \xrightarrow[\acute{\pi}]{} & |B| \\ a & \longleftarrow\!\shortmid & (a, b) & \longmapsto & b \qquad \forall a \in |A| \quad \forall b \in |B| \end{array}$$

are called the **projections** of the product.

The product $A \times B$ is a CPO and the projections $\grave{\pi}$ and $\acute{\pi}$ are ω-continuous, *i.e.*

$$A \xleftarrow{\grave{\pi}} A \times B \xrightarrow{\acute{\pi}} B$$

2.1.14. Definition and Lemma (exponent of CPOs). For every CPOs $A = (|A|, \sqsubseteq_A, \perp_A)$ and $B = (|B|, \sqsubseteq_B, \perp_B)$ we define the **exponent**

$$A^B := (\boldsymbol{\mathcal{C}po}(A,B), \sqsubseteq, \perp)$$

by

$$\frac{fb \sqsubseteq_A gb \quad \forall\, b \in |B|}{f \sqsubseteq g}$$

and

$$\perp : \begin{cases} |A| & \leftarrow & |B| \\ \perp_A & \leftarrow\!\shortmid & b \quad \forall\, b \in |B|. \end{cases}$$

The function

$$apply : \begin{cases} |A| & \leftarrow & |A^B \times B| \\ fb & \leftarrow\!\shortmid & (f,b) \qquad \forall\, f \in \boldsymbol{\mathcal{C}po}(A,B) \quad \forall\, b \in B \end{cases}$$

is called **application**.

The exponent A^B is a CPO and the application *apply* is ω-continuous, *i.e.*

$$apply : A \leftarrow A^B \times B.$$

2.1.15. Lemma (ω-continuity of the composition). Let A, B and C be CPOs. The composition

$$\cdot : \boldsymbol{\mathcal{C}po}(A,C) \leftarrow \boldsymbol{\mathcal{C}po}(A,B) \times \boldsymbol{\mathcal{C}po}(B,C)$$

is ω-continuous, *i.e.*

$$\cdot : A^C \leftarrow A^B \times B^C.$$

2.1.16. Definition (sum). For every CPOs $A = (|A|, \sqsubseteq_A, \perp_A)$ and $B = (|B|, \sqsubseteq_B, \perp_B)$ we define the **sum**

$$A + B := \big(|A| \times \{\perp_B\} \cup \{\perp_A\} \times |B|, \sqsubseteq, \perp\big)$$

by

$$\frac{a \sqsubseteq_A a'}{(a, \perp_B) \sqsubseteq (a', \perp_B)} \quad \text{und} \quad \frac{b \sqsubseteq_B b'}{(\perp_A, b) \sqsubseteq (\perp_A, b')}$$

and

$$\perp := (\perp_A, \perp_B).$$

The functions

$$\begin{array}{ccc} |A| & \xrightarrow[\imath]{} & |A+B| \\ a & \longmapsto & (a, \perp_B) \end{array} \qquad \begin{array}{ccc} |A+B| & \xleftarrow[\acute{\imath}]{} & |B| \\ (\perp_A, b) & \longleftarrow\!\shortmid & b \end{array} \qquad \forall\, a \in |a| \quad \forall\, b \in |B|$$

are called the **injections** of the sum.

The sum $A + B$ is a CPO and the injections $\grave{\imath}$ and $\acute{\imath}$ are ω-continuous and strict, *i.e.*

$$A \xrightarrow[\mathcal{C}po_\perp]{\grave{\imath}} A + B \xleftarrow[\mathcal{C}po_\perp]{\acute{\imath}} B$$

2.2 Basic category theory

2.2.1. Note (set, class, quasi-class). The set of all sets does not exist due to Russel's Paradox. We use class-theory (*cf.* [Fel78]) and define a set to be a class, which is an element of some other class, *i.e.*: A is a set, iff A is a class and there exists a class C such that $A \in C$. Now we can introduce the class of all sets, which is not a set itself. For the same reasons we may not define the class of all classes, but we may iterate the above construction and conceive the quasi-class of all classes, which is not a class itself. In fact one could construct the classes of order n for $n \in \mathbb{N}_0$ where the collection of all classes of order n is a class of order $n+1$ which is not a class of order n itself. In that context sets, classes and quasi-classes would be classes of order 0, 1 and 2, respectively. We will not need classes of order greater than 2 in this paper.

2.2.2. Definition (category). A **category** $\mathcal{C} = (\mathrm{Ob}\,\mathcal{C}, \mathrm{Mor}\,\mathcal{C}, \mathrm{dom}, \mathrm{cod}, \cdot, id)$ consists of a class $\mathrm{Ob}\,\mathcal{C}$ of so called **objects**, a class $\mathrm{Mor}\,\mathcal{C}$ of so called **morphisms**, two functions dom, cod : $\mathrm{Ob}\,\mathcal{C} \leftarrow \mathrm{Mor}\,\mathcal{C}$ called **domain**- or **source**-function and **co-domain**- or **target**-function, a partial function $\cdot : \mathrm{Mor}\,\mathcal{C} \leftarrow\!\!\cdots \mathrm{Mor}\,\mathcal{C} \times \mathrm{Mor}\,\mathcal{C}$ called **composition** and a function $id_{\bullet} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Ob}\,\mathcal{C}$ called **identity**. Before we introduce the axioms, which a category has to satisfy we define the **hom-classes** by

$$\mathcal{C}(A,B) := \left\{ f \in \mathrm{Mor}\,\mathcal{C} \;\middle|\; A = \mathrm{cod}\,f \;\wedge\; \mathrm{dom}\,f = B \right\} \quad \forall\, A, B \in \mathrm{Ob}\,\mathcal{C}$$

and the ternary relation $(\bullet : \bullet \underset{\mathcal{C}}{\leftarrow} \bullet) \subseteq \mathrm{Mor}\,\mathcal{C} \times \mathrm{Ob}\,\mathcal{C} \times \mathrm{Ob}\,\mathcal{C}$ by

$$f : A \underset{\mathcal{C}}{\leftarrow} B \;:\Longleftrightarrow\; A \underset{\mathcal{C}}{\xleftarrow{f}} B \;:\Longleftrightarrow\; f \in \mathcal{C}(A,B) \quad \forall\, A, B \in \mathrm{Ob}\,\mathcal{C}$$

which we will write just

$$f : A \leftarrow B \qquad \text{or} \qquad A \xleftarrow{f} B$$

if the connection to the category $\mathcal{C}$ is obvious. The axioms are:

$$\frac{f, g, h \in \mathrm{Mor}\,\mathcal{C} \qquad f \cdot g = h}{\mathrm{dom}\,f = \mathrm{cod}\,g} \tag{composition}$$

$$\frac{f : A \leftarrow B \qquad g : B \leftarrow C}{f \cdot g : A \leftarrow C} \tag{typing}$$

$$\frac{f : A \leftarrow B \qquad g : B \leftarrow C \qquad h : C \leftarrow D}{(f \cdot g) \cdot h = f \cdot (g \cdot h)} \tag{associativity}$$

$$\frac{f : A \leftarrow B}{f \cdot id_B = f = id_A \cdot f} \tag{identity}$$

2.2.3. Definition (trivial category). The category $\mathbf{1}$ with $\mathrm{Ob}\,\mathbf{1} = \{1\}$ and $\mathrm{Mor}\,\mathbf{1} = \{id_1\}$ is called **trivial category**.

2.2.4. Definition (pre-category). A **pre-category** is defined by the same axioms as a category except that dom and cod are only *partial* functions.

2.2.5. Note. For every pre-category $\mathcal{C}$ we can construct a category $\mathcal{C}'$ by

$$\operatorname{Ob}\mathcal{C}' := \operatorname{Ob}\mathcal{C} \qquad \mathcal{C}'(A,B) := \bigl\{(A,f,B) \bigm| f : A \underset{\mathcal{C}}{\leftarrow} B\bigr\} \quad \forall\, A,B \in \operatorname{Ob}\mathcal{C}$$

with the obvious composition inherited from $\mathcal{C}$. Notice that the definition of the $\mathcal{C}'$-hom-classes also uniquely determines domains and co-domains. For convenience we will use the above construction implicitly and treat pre-categories as categories without any special notation.

2.2.6. Example ([AHS90])**.** We state some categories:

category	objects	morphisms
$\mathcal{Set}$	all sets	all set functions
$\mathcal{Poset}$	all posets	all monotone function
$\mathcal{Cpo}$	all CPOs	all ω-continuous functions
$\mathcal{Cpo}_{\perp}$	all CPOs	all ω-continuous, strict function

2.2.7. Definition and Theorem (duality principle). For every category $\mathcal{C}$ we define the **dual** (or **opposite**) category (to $\mathcal{C}$) by

$$\begin{aligned} \operatorname{Ob}\mathcal{C}^{\mathrm{op}} &:= \operatorname{Ob}\mathcal{C} \\ \mathcal{C}^{\mathrm{op}}(A,B) &:= \mathcal{C}(B,A) \\ f \cdot_{\mathcal{C}^{\mathrm{op}}} g &:= g \cdot_{\mathcal{C}} f \\ id_{\mathcal{C}^{\mathrm{op}}} &:= id_{\mathcal{C}}, \end{aligned}$$

i.e. $\mathcal{C}^{\mathrm{op}}$ has the same objects and morphisms as $\mathcal{C}$, but dom and cod are exchanged and the composition is commuted. The transformation from $\mathcal{C}$ to $\mathcal{C}^{\mathrm{op}}$ may be viewed as the reversion of all morphism arrows. If this is done twice, then we receive the original category:

$$(\mathcal{C}^{\mathrm{op}})^{\mathrm{op}} = \mathcal{C}.$$

We will use this interesting symmetry: For every predicate $A(\mathcal{C})$ about a category $\mathcal{C}$ we derive the **dual** predicate $A^{\mathrm{op}}(\mathcal{C}) :\iff A(\mathcal{C}^{\mathrm{op}})$ by reversing all morphism arrows. It holds:

$$\bigl(\text{for every category } \mathcal{C} ::\ A(\mathcal{C})\bigr) \quad\iff\quad \bigl(\text{for every category } \mathcal{C} ::\ A^{\mathrm{op}}(\mathcal{C})\bigr)$$

Proof.

$$\begin{aligned} & \forall\,\mathcal{C} ::\ A(\mathcal{C}) \\ \iff & \quad \{\text{ since } \forall\,\mathcal{C} ::\ \mathcal{C} = (\mathcal{C}^{\mathrm{op}})^{\mathrm{op}} \ \} \\ & \forall\,\mathcal{C} ::\ A(\mathcal{C}^{\mathrm{op}}) \\ \iff & \quad \{\text{ definition of the dual predicate }\} \\ & \forall\,\mathcal{C} ::\ A^{\mathrm{op}}(\mathcal{C}) \end{aligned}$$

∎

Thus a proof for a proposition about categories is also a proof for the dual proposition. Furthermore, for every term defined in category theory there is a dual term. We only need to investigate one of them and get the properties of the dual term by the duality principle.

2.2.8. Definition (isomorphism). Let $\mathcal{C}$ be a category. A morphism $f \in \operatorname{Mor}\mathcal{C}$ with

$$\exists\, g \in \operatorname{Mor}\mathcal{C} ::\ f \cdot g = id \ \wedge\ g \cdot f = id$$

is called an **isomorphism**. It is easy to see that in this case g is unique. We call it the **inverse** of f and denote it by f^{-1} Two objects $A, B \in \operatorname{Ob}\mathcal{C}$ such that

$$\exists \text{ isomorphism } f \in \operatorname{Mor}\mathcal{C} ::\ f : A \leftarrow B$$

are called **isomorphic**, we write

$$A \cong B$$

Obviously, the relation $\cong$ is an equivalence relation.

2.2.9. Definition (functor). Let $\boldsymbol{\mathcal{C}}$, $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{E}}$ be categories. A **(covariant) functor** F to $\boldsymbol{\mathcal{C}}$ from $\boldsymbol{\mathcal{D}}$ consists of two functions

$$\mathsf{F} : \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \leftarrow \mathrm{Ob}\,\boldsymbol{\mathcal{D}} \quad \text{and} \quad \mathsf{F} : \mathrm{Mor}\,\boldsymbol{\mathcal{C}} \leftarrow \mathrm{Mor}\,\boldsymbol{\mathcal{D}},$$

which suffice the following axioms:

$$\frac{f : A \underset{\boldsymbol{\mathcal{D}}}{\longleftarrow} B}{\mathsf{F}f : \mathsf{F}A \underset{\boldsymbol{\mathcal{C}}}{\longleftarrow} \mathsf{F}B} \qquad \text{(typing)}$$

$$\frac{f : A \underset{\boldsymbol{\mathcal{D}}}{\longleftarrow} B \qquad g : B \underset{\boldsymbol{\mathcal{D}}}{\longleftarrow} C}{\mathsf{F}(f \cdot g) = \mathsf{F}f \ \cdot \ \mathsf{F}g} \qquad \text{(multiplicativity)}$$

$$\frac{A \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}}{\mathsf{F}id_A = id_{\mathsf{F}A}} \qquad \text{(identity)}$$

We write:

$$\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$$

It is common, to denote the functor as well as the two underlying functions by the same symbol. In fact, a functor is already determined on objects by (identity), if it is defined on morphisms.

For every functor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ we define the **dual functor** $\mathsf{F}^{\mathrm{op}} : \boldsymbol{\mathcal{C}}^{\mathrm{op}} \leftarrow \boldsymbol{\mathcal{D}}^{\mathrm{op}}$, which is determined by the same underlying functions on objects and morphisms as F. It is easy to see that F^{op} is indeed a functor.

A functor $\mathsf{E} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$, which maps a category on itself, is called an **endofunctor**.

A functor $\mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}^{\mathrm{op}}$, *i.e.* a covariant functor from $\boldsymbol{\mathcal{D}}^{\mathrm{op}}$, is called a **contravariant functor**[4] from $\boldsymbol{\mathcal{D}}$ (*not* $\boldsymbol{\mathcal{D}}^{\mathrm{op}}$).

A functor $\mathsf{H} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}} \times \boldsymbol{\mathcal{D}}'$ (see Definition 2.2.11) is called a **bifunctor** from $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{D}}'$. For bifunctors we will also use the infix notation $\centerdot \otimes \centerdot : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}} \times \boldsymbol{\mathcal{D}}' : f \otimes g \leftarrowtail (f, g)$ and use the same operation symbol $\otimes$ for the pointwise operation, which maps two functors F and F' on a functor $\mathsf{F} \otimes \mathsf{F}'$:

$$\mathsf{F} \otimes \mathsf{F}' : \begin{cases} \boldsymbol{\mathcal{C}} & \leftarrow \boldsymbol{\mathcal{E}} \\ (\mathsf{F}A) \otimes (\mathsf{F}'A) & \leftarrowtail A \qquad \forall A \in \mathrm{Ob}\,\boldsymbol{\mathcal{E}} \\ (\mathsf{F}f) \otimes (\mathsf{F}'f) & \leftarrowtail f \qquad \forall f \in \mathrm{Mor}\,\boldsymbol{\mathcal{E}} \end{cases} \qquad \begin{array}{l} \forall \mathsf{F} : \boldsymbol{\mathcal{D}} \leftarrow \boldsymbol{\mathcal{E}} \\ \forall \mathsf{F}' : \boldsymbol{\mathcal{D}}' \leftarrow \boldsymbol{\mathcal{E}}. \end{array}$$

We define the **identity functor**

$$\mathsf{Id} : \begin{cases} \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \\ A \leftarrowtail A \qquad \forall A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \\ f \leftarrowtail f \qquad \forall f \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}} \end{cases}$$

[4] According to our definition any functor is covariant. We use the term 'contravariant functor' to emphasize that the functor maps from the dual category of some given category. Many authors use a different definition.

and for every $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ the **constant functor**

$$\mathsf{K}_A : \begin{cases} \boldsymbol{\mathcal{C}} & \leftarrow \boldsymbol{\mathcal{D}} \\ A & \leftarrow\!\shortmid B \qquad \forall\, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} \\ id_A & \leftarrow\!\shortmid f \qquad \forall\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{D}}. \end{cases}$$

where it is easy to see that these are indeed functors.

For simplicity we avoid notations like $\mathsf{Id}_{\boldsymbol{\mathcal{C}}}$ or $\mathsf{K}_A^{\boldsymbol{\mathcal{C}},\boldsymbol{\mathcal{D}}}$, if the connection to the categories is obvious.

2.2.10. Definition ((full) subcategory). Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories. The category $\boldsymbol{\mathcal{D}}$ is called a **subcategory** of $\boldsymbol{\mathcal{C}}$ provided that

$$\mathrm{Ob}\,\boldsymbol{\mathcal{D}} \subseteq \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \text{ und } \mathrm{Mor}\,\boldsymbol{\mathcal{D}} \subseteq \mathrm{Mor}\,\boldsymbol{\mathcal{C}}$$

holds. If in addition

$$\boldsymbol{\mathcal{D}}(A,B) = \boldsymbol{\mathcal{C}}(A,B) \quad \forall\, A,B \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}}$$

is true, then $\boldsymbol{\mathcal{D}}$ is called a **full** subcategory of $\boldsymbol{\mathcal{C}}$. The functor

$$\mathsf{E} : \begin{cases} \boldsymbol{\mathcal{C}} & \leftarrow \boldsymbol{\mathcal{D}} \\ A & \leftarrow\!\shortmid A \qquad \forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} \\ f & \leftarrow\!\shortmid f \qquad \forall\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{D}} \end{cases}$$

is called the **canonical embedding** of the subcategory $\boldsymbol{\mathcal{D}}$ in $\boldsymbol{\mathcal{C}}$.

2.2.11. Definition (product-category). For two categories $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ we define the **product-category** $\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{D}}$ by

$$\mathrm{Ob}(\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{D}}) := \Big\{ (A,A') \;\Big|\; A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \;\wedge\; A' \in \mathrm{Ob}\,\boldsymbol{\mathcal{D}} \Big\}$$

and

$$\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{D}}\big((A,A'),(B,B')\big) := \Big\{ (f,f') \;\Big|\; f : A \underset{\boldsymbol{\mathcal{C}}}{\leftarrow} B \;\wedge\; f' : A' \underset{\boldsymbol{\mathcal{D}}}{\leftarrow} B' \Big\}$$

with component wise composition. The functors

$$\begin{array}{ccccccc} \boldsymbol{\mathcal{C}} & \overset{\grave{\mathsf{P}}}{\longleftarrow} & \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{D}} & \overset{\acute{\mathsf{P}}}{\longrightarrow} & \boldsymbol{\mathcal{D}} & & \\ A & \longleftarrow\!\shortmid & (A,A') & \longmapsto & A' & \forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} & \forall\, A' \in \boldsymbol{\mathcal{D}} \\ f & \longleftarrow\!\shortmid & (f,f') & \longmapsto & f' & \forall\, f \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}} & \forall\, f' \in \mathrm{Mor}\,\boldsymbol{\mathcal{D}} \end{array}$$

are called the **projection-functors**.

2.2.12. Definition (quasi-category). A **quasi-category** is defined by the same axioms as a category except that the collections of objects and morphisms are quasi-classes and need not to be classes.

For every predicate P on categories, which uses the terms ‘set’ or ‘class’ explicitly, we use the convention that the predicate quasi-P denotes the predicate on quasi-categories, which we obtain from P by substituting ‘set’ by ‘class’ and ‘class’ by ‘quasi-class’.

2.2.13. Note. The quasi-category of all quasi-categories does not exist for the same reason that the category of all categories does not exist, but we can define the following:

2.2.14. Definition (quasi-categories of all classes and of all categories). We denote the quasi-category of all classes with all class-functions as morphisms by $\boldsymbol{CLASS}$ and the quasi-category of all categories with all functors as morphisms by $\boldsymbol{CAT}$.

2.2.15. Note. every statement on categories, which does not make use of the fact that the collections of objects and morphisms are classes is also true for quasi-categories.

2.2.16. Definition (locally small category, hom-functor). A category $\boldsymbol{\mathcal{C}}$ is called **locally small** if the hom-classes of $\boldsymbol{\mathcal{C}}$ are sets, *i.e.*

$$\boldsymbol{\mathcal{C}}(A,B) \in \operatorname{Ob} \boldsymbol{\mathcal{S}et} \quad \forall\, A,B \in \operatorname{Ob} \boldsymbol{\mathcal{C}}.$$

The above sets $\boldsymbol{\mathcal{C}}(A,B)$ are called the **hom-sets** of $\boldsymbol{\mathcal{C}}$. In this case we can introduce the **hom-functor** of $\boldsymbol{\mathcal{C}}$

$$\boldsymbol{\mathcal{C}}(\,\centerdot\,,\centerdot\,) : \begin{cases} \quad \boldsymbol{\mathcal{S}et} & \leftarrow \ \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}^{\mathrm{op}} \\ \ \ \boldsymbol{\mathcal{C}}(A,B) & \leftarrow\!\shortmid \ (A,B) \qquad \forall\, A,B \in \operatorname{Ob} \boldsymbol{\mathcal{C}} \\ (f \cdot p \cdot g \leftarrow\!\shortmid p) & \leftarrow\!\shortmid \ \ (f,g) \qquad \forall\, f \in \operatorname{Mor} \boldsymbol{\mathcal{C}} \quad \forall\, g \in \operatorname{Mor} \boldsymbol{\mathcal{C}}^{\mathrm{op}}, \end{cases}$$

and as a special case the so called **covariant hom-functors**

$$\boldsymbol{\mathcal{C}}(\,\centerdot\,,B) : \boldsymbol{\mathcal{S}et} \leftarrow \boldsymbol{\mathcal{C}} \quad \forall\, B \in \operatorname{Ob} \boldsymbol{\mathcal{C}}.$$

2.2.17. Definition ((natural) transformation). Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories and $\mathsf{F},\mathsf{G} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be functors. A function

$$\tau = (\tau_A)_{A \in \operatorname{Ob} \boldsymbol{\mathcal{D}}} \in (\operatorname{Mor} \boldsymbol{\mathcal{C}})^{\operatorname{Ob} \boldsymbol{\mathcal{D}}}$$

with

$$\tau_A : \mathsf{F}A \underset{\boldsymbol{\mathcal{C}}}{\longleftarrow} \mathsf{G}A \quad \forall\, A \in \operatorname{Ob} \boldsymbol{\mathcal{D}}$$

is called a **transformation** to F from G, we write:

$$\tau : \mathsf{F} \leftarrow \mathsf{G}.$$

If in addition τ satisfies the so called **naturalness condition**

$$\frac{h : A \underset{\boldsymbol{\mathcal{D}}}{\longleftarrow} B}{\mathsf{F}h \cdot \tau_B = \tau_A \cdot \mathsf{G}h}$$

then it is called a **natural** transformation to F from G, and we write:

$$\tau : \mathsf{F} \mathrel{\dot\leftarrow} \mathsf{G}.$$

2.2.18. Definition and Lemma (composition of (natural) transformations). Let $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be categories and $\mathsf{F},\mathsf{G},\mathsf{H} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ be functors and $\sigma : \mathsf{F} \leftarrow \mathsf{G}$ and $\tau : \mathsf{G} \leftarrow \mathsf{H}$ be transformations. We define the composition of transformations component wise by

$$(\sigma \cdot \tau)_A := \sigma_A \cdot \tau_A \quad \forall\, A \in \operatorname{Ob} \boldsymbol{\mathcal{D}}$$

It holds: The composition of (natural) transformations is a (natural) transformation, *i.e.*:

$$\frac{\sigma : \mathsf{F} \leftarrow \mathsf{G} \qquad \tau : \mathsf{G} \leftarrow \mathsf{H}}{\sigma \cdot \tau : \mathsf{F} \leftarrow \mathsf{H}} \quad \text{and} \quad \frac{\sigma : \mathsf{F} \mathrel{\dot\leftarrow} \mathsf{G} \qquad \tau : \mathsf{G} \mathrel{\dot\leftarrow} \mathsf{H}}{\sigma \cdot \tau : \mathsf{F} \mathrel{\dot\leftarrow} \mathsf{H}}$$

Proof. Immediately with Definition 2.2.17. ∎

2.2.19. Definition and Lemma (functor-category). Let $\mathcal{C}$ and $\mathcal{D}$ be categories. We define the quasi-category $\mathcal{C}^{\mathcal{D}}$ by

$$\operatorname{Ob}\mathcal{C}^{\mathcal{D}} := \{\mathsf{F} \mid \mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}\}$$

and for every $\mathsf{F}, \mathsf{G} \in \operatorname{Ob}\mathcal{C}^{\mathcal{D}}$:

$$\mathcal{C}^{\mathcal{D}}(\mathsf{F}, \mathsf{G}) := \{\tau \mid \tau : \mathsf{F} \stackrel{.}{\leftarrow} \mathsf{G}\}$$

with the composition of natural transformations.

If $\mathcal{D}$ is a *small* category, then $\mathcal{C}^{\mathcal{D}}$ is a locally small category.

Proof. See [Mac72] for the proof that $\operatorname{Ob}\mathcal{C}^{\mathcal{D}}$ and $\operatorname{Mor}(\mathcal{C}^{\mathcal{D}})$ are classes and that $\mathcal{C}^{\mathcal{D}}$ is locally small. The category axioms are obvious by the Definitions 2.2.17 and 2.2.18. ∎

2.2.20. Definition and Lemma (functors respect naturalness). Let $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$ be categories and $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{D}$ be functors. For a natural transformation

$$\tau : \mathsf{F} \stackrel{.}{\leftarrow} \mathsf{G}$$

we have:

(i) The composition (as class-functions) of a functor $\mathsf{H} : \mathcal{E} \leftarrow \mathcal{C}$ and τ defined by

$$(\mathsf{H}\tau)_X := \mathsf{H}(\tau_X) \quad \forall\, X \in \operatorname{Ob}\mathcal{D}$$

is a natural transformation:

$$\mathsf{H}\tau : \mathsf{H} \cdot \mathsf{F} \stackrel{.}{\leftarrow} \mathsf{H} \cdot \mathsf{G}.$$

(ii) The composition (as class-functions) of τ and a functor $\mathsf{H} : \mathcal{D} \leftarrow \mathcal{E}$ defined by

$$(\tau\mathsf{H})_X := \tau_{\mathsf{H}X} \quad \forall\, X \in \operatorname{Ob}\mathcal{E}$$

is also a natural transformation:

$$\tau\mathsf{H} : \mathsf{F} \cdot \mathsf{H} \stackrel{.}{\leftarrow} \mathsf{G} \cdot \mathsf{H}.$$

Proof. Immediately by the Definitions 2.2.9 and 2.2.17. ∎

2.2.21. Definition (hom-set restricted functor). Let $\mathcal{C}$ and $\mathcal{D}$ be locally small categories, $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ be a functor and $A, B \in \operatorname{Ob}\mathcal{C}$. We denote the set-function, which we obtain from the class-function

$$\mathsf{F} : \operatorname{Mor}\mathcal{C} \xleftarrow[\boldsymbol{CLASS}]{} \operatorname{Mor}\mathcal{D}$$

by restricting the domain of F to the hom-set $\mathcal{D}(A, B) \subseteq \operatorname{Mor}\mathcal{D}$ and the co-domain of F to the hom-set $\mathcal{C}(\mathsf{F}A, \mathsf{F}B) \subseteq \operatorname{Mor}\mathcal{D}$ by

$$\mathsf{F}_{(A,B)} : \mathcal{C}(\mathsf{F}A, \mathsf{F}B) \xleftarrow[\mathcal{S}et]{} \mathcal{D}(A, B)$$

and we call it the **hom-set restriction** of F or the **localization** of F by A, B. Using the multiplicativity of the functor F, it is easy to see that we have got a natural transformation

$$\mathsf{F}_{(\bullet,\bullet)} : \mathcal{C}(\mathsf{F}\bullet, \mathsf{F}\bullet) \stackrel{.}{\leftarrow} \mathcal{D}(\bullet, \bullet).$$

2.2.22. Definition (embedding, faithful functor). Let $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}$ be a functor. F is called an **embedding** provided that F is injective on morphisms. F is called **faithful** provided that all its hom-set restrictions $\mathsf{F}_{(A,B)}$ are injective for every $A, B \in \mathcal{D}$.

2.2.23. Note. A functor is an embedding iff it is faithful and injective on objects.

2.2.24. Definition (initial/final object, null-object). Let $\boldsymbol{\mathcal{C}}$ be a category. An object $0 \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$ is called an **initial object** of $\boldsymbol{\mathcal{C}}$, if for every $\boldsymbol{\mathcal{C}}$-object A there exists a unique $\boldsymbol{\mathcal{C}}$-morphism to A from 0, *i.e.*:

$$\#\big(\boldsymbol{\mathcal{C}}(A,0)\big) = 1 \quad \forall\, A \in \operatorname{Ob}\boldsymbol{\mathcal{C}}.$$

or equivalently

$$\forall\, A \in \operatorname{Ob}\boldsymbol{\mathcal{C}} :: \ \exists!\, f \in \operatorname{Mor}\boldsymbol{\mathcal{C}} :: \ f : A \leftarrow 0$$

The above statement has the form '$\forall \cdots \exists! \cdots$' and can thus be used to define a function

$$\text{¡}_{(\,.\,)\leftarrow 0} : \operatorname{Mor}\boldsymbol{\mathcal{C}} \leftarrow \operatorname{Ob}\boldsymbol{\mathcal{C}}$$

by

$$f = \text{¡}_{A\leftarrow 0} \iff f : A \leftarrow 0 \quad \forall\, f \in \operatorname{Mor}\boldsymbol{\mathcal{C}}. \tag{UP}$$

The function ¡ is called the **unique mediating morphism** or **co-mediator** of 0. The defining property (UP) is called the **universal property**. If the connection to the initial object $0 \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$ is obvious, we simply write ¡_A for $\text{¡}_{A\leftarrow 0}$.

Dually an object $1 \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$ with

$$\#\big(\boldsymbol{\mathcal{C}}(1,A)\big) = 1 \quad \forall\, A \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$$

is called **final object** (and sometimes **terminal object**) of $\boldsymbol{\mathcal{C}}$ and the function

$$!_{(\,.\,)\rightarrow 1} : \operatorname{Mor}\boldsymbol{\mathcal{C}} \leftarrow \operatorname{Ob}\boldsymbol{\mathcal{C}}$$

with

$$f = !_{A\rightarrow 1} \iff f : 1 \leftarrow A \quad \forall\, f \in \operatorname{Mor}\boldsymbol{\mathcal{C}} \tag{UP$^{\mathrm{op}}$}$$

is called the **unique mediating morphism** or **mediator** of 1. If the connection to the final object $1 \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$ is obvious, we simply write $!_A$ for $!_{A\rightarrow 1}$.

A $\boldsymbol{\mathcal{C}}$-object that is initial and final as well is called a **null-object** of $\boldsymbol{\mathcal{C}}$.

2.2.25. Lemma (essential uniqueness of initial objects). Let $\boldsymbol{\mathcal{C}}$ be a category. every two initial objects of $\boldsymbol{\mathcal{C}}$ are isomorphic.

Proof. Let 0 and $0'$ be initial objects with co-mediators $\text{¡}_{(\,.\,)\leftarrow 0}$ and $\text{¡}_{(\,.\,)\leftarrow 0'}$, respectively.

$$\begin{aligned}
\Longrightarrow &\quad \{ \text{ (typing) } \}\\
&\text{¡}_{0\leftarrow 0'} \cdot \text{¡}_{0'\leftarrow 0} : 0 \leftarrow 0\\
\Longrightarrow &\quad \{ \text{ (UP), (composition) \& (identity) } \}\\
&\text{¡}_{0\leftarrow 0'} \cdot \text{¡}_{0'\leftarrow 0} = \text{¡}_{0\leftarrow 0} = id_0.
\end{aligned}$$

Dually follows: $\text{¡}_{0'\leftarrow 0} \cdot \text{¡}_{0\leftarrow 0'} = id_{0'}$. Hence $\text{¡}_{0\leftarrow 0'} : 0 \leftarrow 0'$ is an isomorphism and thus $0 \cong 0'$.

Since $\#\big(\boldsymbol{\mathcal{C}}(0,0')\big) = 1$ the morphism $\text{¡}_{0\leftarrow 0'}$ is even the only isomorphism to 0 from $0'$. ∎

2.2.26. Lemma (laws for initial/final objects). Let $\boldsymbol{\mathcal{C}}$ be a category with initial object $0 \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$. The following rules hold:

laws for co-mediators	
UP	$f = \text{¡}_A \iff f : A \leftarrow 0$
reflection	$\text{¡}_0 = id_0$
fusion	$f : A \leftarrow B \implies f \cdot \text{¡}_B = \text{¡}_A$
where $f \in \operatorname{Mor}\boldsymbol{\mathcal{C}}$ and $A, B \in \operatorname{Ob}\boldsymbol{\mathcal{C}}$	

Proof. Analogically to Lemma 2.2.25. ∎

Obviously the dual rules hold for a category $\mathcal{C}$ with a final object $1 \in \mathrm{Ob}\,\mathcal{C}$:

laws for mediators	
UP	$f = !_A \iff f : 1 \leftarrow A$
reflection	$!_1 = id_1$
fusion	$f : A \leftarrow B \implies !_A \cdot f = !_B$
where $f \in \mathrm{Mor}\,\mathcal{C}$ and $A, B \in \mathrm{Ob}\,\mathcal{C}$	

2.2.27. Corollary. Let $\mathcal{C}$ be a category with initial object $0 \in \mathrm{Ob}\,\mathcal{C}$. The function $¡$ yielding the unique mediating morphism is a natural transformation, *i.e.*:

$$¡ : \mathsf{Id} \dot{\leftarrow} \mathsf{K}_0.$$

Proof. Since

$$¡_A : A \leftarrow 0 \quad \forall A \in \mathrm{Ob}\,\mathcal{C}$$

it is a transformation

$$¡ : \mathsf{Id} \leftarrow \mathsf{K}_0.$$

The fusion-rule from Lemma 2.2.26 is equivalent to the naturalness condition in Definition 2.2.17. ∎

2.2.28. Example. We state the initial and initial/final objects for some categories:

category	**initial object**	**final object**
$\mathcal{Set}$	$\emptyset$	every $\mathbf{1} \in \mathrm{Ob}\,\mathcal{Set}$ with $\#(\mathbf{1}) = 1$
$\mathcal{Poset}$	$\emptyset$	every $\mathbf{1} \in \mathrm{Ob}\,\mathcal{Poset}$ with $\#(\mathbf{1}) = 1$
$\mathcal{Cpo}$	—*none*—	$\{\perp\}$
$\mathcal{Cpo}_\perp$	$\{\perp\}$	$\{\perp\}$

The following definitions are constructed analogically to those of initial and final object. With Theorem 4.1.1 we will see that the following definitions are just instances of specific initial and final objects in some concrete category build upon the default category (this will be explained later in Section 3 and Section 4). We worked this out in detail for algebras in Definition 2.2.33, which the reader should compare with Definition 3.1.1. We also give the concrete category for exponents in Definition and Lemma 2.2.32. For (co-)products (Definitions 2.2.29 and 2.2.31) we leave this to the reader.

2.2.29. Definition and Lemma (product). An object $A \times B \in \mathrm{Ob}\,\mathcal{C}$ of a category $\mathcal{C}$ together with two morphisms

$$A \xleftarrow{\grave{\pi}} A \times B \xrightarrow{\acute{\pi}} B$$

is called *a* **product** of $A \in \mathrm{Ob}\,\mathcal{C}$ and $B \in \mathrm{Ob}\,\mathcal{C}$ with **projections** $\grave{\pi}$ and $\acute{\pi}$, provided that for every object $C \in \mathrm{Ob}\,\mathcal{C}$ and every two morphisms

$$A \xleftarrow{f} C \xrightarrow{g} B$$

there exists a unique morphism

$$\langle f, g \rangle : A \times B \leftarrow C,$$

which is called **split** or **pairing**, such that the following diagram commutes:

$$\begin{tikzcd}
A & A \times B \arrow[l, "\grave{\pi}"'] \arrow[r, "\acute{\pi}"] & B \\
& C \arrow[ul, "f"] \arrow[u, "\langle f, g\rangle"'] \arrow[ur, "g"'] &
\end{tikzcd}$$

If a category $\mathcal{C}$ has a final object $1 \in \operatorname{Ob}\mathcal{C}$ and for every two objects $A, B \in \operatorname{Ob}\mathcal{C}$ has a product $A \times B \in \operatorname{Ob}\mathcal{C}$, then we say that the category $\mathcal{C}$ has **finite products**.

We define the so called **diagonal** $\Delta : \mathsf{Id} \times \mathsf{Id} \dot{\leftarrow} \mathsf{Id}$ by

$$\Delta_A := \langle id_A, id_A \rangle \quad \forall A \in \operatorname{Ob}\mathcal{C}.$$

It is easy to see that we find laws analogue to Lemma 2.2.26.

laws for $\langle\centerdot,\centerdot\rangle$	
UP	$h = \langle f, g\rangle \iff \grave{\pi} \cdot h = f \ \wedge \ \acute{\pi} \cdot h = g$
reflection	$\langle \grave{\pi}, \acute{\pi}\rangle = id_{A\times B}$
fusion	$\langle f, g\rangle \cdot h = \langle f \cdot h, g \cdot h\rangle$
cancelation	$\grave{\pi} \cdot \langle f, g\rangle = f \ \wedge \ \acute{\pi} \cdot \langle f, g\rangle = g$
where $h \in \operatorname{Mor}\mathcal{C}$ and $f : A \leftarrow C$ and $g : B \leftarrow C$	

Proof. The UP is equivalent to the definition of the product. The remaining rules follow analogically to Lemma 2.2.25 and Lemma 2.2.26. ∎

2.2.30. Lemma. In a category $\mathcal{C}$ with finite products we have the following isomorphisms:

$$\begin{aligned} A \times B &\cong B \times A \\ A &\cong 1 \times A \\ A \times (B \times C) &\cong (A \times B) \times C \end{aligned}$$

Proof. Using the laws for $\langle\centerdot,\centerdot\rangle$ from Definition 2.2.29 and for $!$ from Lemma 2.2.26 it is easy to calculate that

$$\begin{array}{rcccccc} swap_{(B,A)} & := & \langle \acute{\pi}_{(B,A)}, \grave{\pi}_{(B,A)} \rangle & : & A \times B & \leftarrow & B \times A \\ unit_A & := & \acute{\pi}_{(1,A)} & : & A & \leftarrow & 1 \times A \\ assocr & := & \big\langle \grave{\pi} \cdot \grave{\pi}, \langle \acute{\pi} \cdot \grave{\pi}, \acute{\pi} \rangle \big\rangle & : & A \times (B \times C) & \leftarrow & (A \times B) \times C \end{array}$$

are natural isomorphisms with inverses

$$\begin{array}{rcccccc} swap_{(B,A)}^{-1} & = & swap_{(A,B)} & : & B \times A & \leftarrow & A \times B \\ unit_A^{-1} & = & \langle !_A, id_A \rangle & : & 1 \times A & \leftarrow & A \\ assocr^{-1} & = & assocl := \big\langle \langle \grave{\pi}, \grave{\pi} \cdot \acute{\pi} \rangle, \acute{\pi} \cdot \acute{\pi} \big\rangle & : & (A \times B) \times C & \leftarrow & A \times (B \times C). \end{array}$$

∎

2.2.31. Definition and Lemma (sum). The dual concept to 'product' is the following: An object $A + B \in \operatorname{Ob}\mathcal{C}$ of a category $\mathcal{C}$ together with to morphisms

$$A \xrightarrow{\grave{\iota}} A + B \xleftarrow{\acute{\iota}} B$$

is called *a* **sum** or **co-product** of $A \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$ and $B \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$ with **injections** $\grave{\imath}$ and $\acute{\imath}$, provided that for every object $C \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$ and every two morphisms

$$A \xrightarrow{f} C \xleftarrow{g} B$$

there exists a unique morphism

$$[f, g] : C \leftarrow A + B,$$

which is called **case**, such that the following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "\grave{\imath}"] \arrow[dr, "f"'] & A + B \arrow[d, "{[f,g]}"] & B \arrow[l, "\acute{\imath}"'] \arrow[dl, "g"] \\
& C &
\end{tikzcd}$$

We get the following laws:

laws for $[\cdot, \cdot]$	
UP	$h = [f, g] \iff h \cdot \grave{\imath} = f \ \wedge\ h \cdot \acute{\imath} = g$
reflection	$[\grave{\imath}, \acute{\imath}] = id_{A+B}$
fusion	$h \cdot [f, g] = [h \cdot f, h \cdot g]$
cancelation	$[f, g] \cdot \grave{\imath} = f \ \wedge\ [f, g] \cdot \acute{\imath} = g$
where $h \in \operatorname{Mor} \boldsymbol{\mathcal{C}}$ and $f : C \leftarrow A$ and $g : C \leftarrow A$	

If a category $\boldsymbol{\mathcal{C}}$ has an initial object $0 \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$ and has a sum $A \times B \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$ for every two objects $A, B \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$, we say that the category $\boldsymbol{\mathcal{C}}$ has **finite sums**.

Proof. Dually to Definition 2.2.29. ■

2.2.32. Definition and Lemma (exponent, application, abstraction, currying). Let $\boldsymbol{\mathcal{C}}$ be a category with finite products. For every to objects $A, B \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$ we define the category $\boldsymbol{\mathcal{E}xp}_{\boldsymbol{\mathcal{C}}}(A,B)$ by

$$\operatorname{Ob}\big(\boldsymbol{\mathcal{E}xp}_{\boldsymbol{\mathcal{C}}}(A,B)\big) := \bigcup_{C \in \operatorname{Ob} \boldsymbol{\mathcal{C}}} \boldsymbol{\mathcal{C}}(A, C \times B)$$

and $\forall\, f, g \in \operatorname{Ob}\big(\boldsymbol{\mathcal{E}xp}_{\boldsymbol{\mathcal{C}}}(A,B)\big)$ where $f : A \leftarrow C \times B$ and $g : A \leftarrow D \times B$ with $C, D \in \operatorname{Ob} \boldsymbol{\mathcal{C}}$:

$$\boldsymbol{\mathcal{E}xp}_{\boldsymbol{\mathcal{C}}}(A,B)(f, g) := \big\{ h \in \boldsymbol{\mathcal{C}}(C, D) \ \big|\ f \cdot (h \times id_B) = g \big\}.$$

If it has a final object, we denote it by

$$apply_{(A,B)} = 1_{\boldsymbol{\mathcal{E}xp}_{\boldsymbol{\mathcal{C}}}(A,B)} : A \underset{\boldsymbol{\mathcal{C}}}{\leftarrow} A^B \times B$$

and call the underlying $\boldsymbol{\mathcal{C}}$-object A^B an **exponent** of A and B with **application** $apply_{(A,B)}$. The unique morphism

$$\operatorname{curry} f := \mathbf{!}_f : A^B \underset{\boldsymbol{\mathcal{C}}}{\leftarrow} C \quad (\text{where } C \in \operatorname{Ob} \boldsymbol{\mathcal{C}} \text{ such that } f : A \leftarrow C \times B)$$

for each $f \in \mathrm{Ob}\big(\mathcal{E}\boldsymbol{xp}_{\mathcal{C}}(A,B)\big)$ is called the **abstraction** or **currying**[5]. The function

$$\mathrm{uncurry} : \mathrm{Mor}\,\mathcal{C} \leftarrow \mathrm{Ob}\big(\mathcal{E}\boldsymbol{xp}_{\mathcal{C}}(A,B)\big)$$

defined by

$$\mathrm{uncurry}\, h := \mathit{apply}_{(A,B)} \cdot (h \times \mathit{id}_B) \quad \forall\, h \in \mathrm{Ob}\big(\mathcal{E}\boldsymbol{xp}_{\mathcal{C}}(A,B)\big),$$

is the inverse of curry and we have the following laws:

laws for currying	
UP	$h = \mathrm{curry}\, f \iff \mathit{apply}_{(A,B)} \cdot (h \times \mathit{id}_B) = f$
reflection	$\mathrm{curry}\, \mathit{apply}_{(A,B)} = \mathit{id}_{A^B}$
fusion	$\mathrm{curry}\, f \;\cdot\; h = \mathrm{curry}\big(f \cdot (h \times \mathit{id}_B)\big)$
cancelation	$\mathit{apply}_{(A,B)} \cdot (\mathrm{curry}\, f \times \mathit{id}_B) = f$
where $h \in \mathrm{Mor}\,\mathcal{C}$ and $f \in \mathrm{Ob}\big(\mathcal{E}\boldsymbol{xp}_{\mathcal{C}}(A,B)\big)$	

Proof. Immediately by Definition 2.2.24. ∎

2.2.33. Definition and Lemma (initial/final (co-)algebra, cata-, anamorphism). Let $\mathcal{C}$ be a category and $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ an endofunctor. The category $\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}$, defined by:

$$\mathrm{Ob}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}) := \big\{\varphi \in \mathrm{Mor}\,\mathcal{C} \;\big|\; \exists\, A \in \mathrm{Ob}\,\mathcal{C} :: \; \varphi : A \leftarrow \mathsf{F}A\big\}$$

and $\forall\, \varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F})$ where $\varphi : A \leftarrow \mathsf{F}A$ and $\varphi' : B \leftarrow \mathsf{F}B$ with $A, B \in \mathrm{Ob}\,\mathcal{C}$:

$$\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}\,(\varphi, \varphi') := \big\{f \in \mathcal{C}(A, B) \;\big|\; f \cdot \varphi' = \varphi \cdot \mathsf{F}f\big\}$$

is called the category of **F-algebras** (of $\mathcal{C}$). The according **forgetful functor** (*cf.* Definition 3.1.1) $|\,\centerdot\,| : \mathcal{C} \leftarrow \mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}$ is defined by

$$|\varphi| := \mathrm{cod}_{\mathcal{C}}\, \varphi \quad \forall\, \varphi \in \mathrm{Ob}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F})$$

and

$$|f| := f \quad \forall\, f \in \mathrm{Mor}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}).$$

If $\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}$ has an initial object, then it is denoted by

$$\mathit{in}_{\mathsf{F}} = 0_{\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}} : \mu\mathsf{F} \underset{\mathcal{C}}{\leftarrow} \mathsf{F}(\mu\mathsf{F})$$

and is called the **constructor** of F. The image of the co-mediator

$$\mathsf{i}_{\varphi} : \varphi \underset{\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}}{\longleftarrow} \mathit{in}_{\mathsf{F}} \quad \forall\, \varphi \in \mathrm{Ob}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F})$$

under $|\,\centerdot\,|$, *i.e.*

$$(\!|\varphi|\!)_{\mathsf{F}} := |\mathsf{i}_{\varphi}| : \mathrm{cod}_{\mathcal{C}}\, \varphi \underset{\mathcal{C}}{\leftarrow} \mu\mathsf{F}$$

is called the **catamorphism** generated by φ (w.r.t. F) (from Greek $\kappa\alpha\tau\alpha$, downwards). Note that constructor and catamorphism are determined only unique up to F-algebra-isomorphism, and hence $\mu\mathsf{F}$ is determined only unique up to $\mathcal{C}$-isomorphism. The following laws follow immediately by Lemma 2.2.26:

[5] Named after Haskell B. Curry.

laws for catamorphisms	
UP	$f = (\!\vert\varphi\vert\!)_{\mathsf{F}} \iff f \cdot in_{\mathsf{F}} = \varphi \cdot \mathsf{F}f$
reflection	$(\!\vert in_{\mathsf{F}}\vert\!)_{\mathsf{F}} = id_{\mu\mathsf{F}}$
fusion	$f \cdot \varphi' = \varphi \cdot \mathsf{F}f \implies f \cdot (\!\vert\varphi'\vert\!)_{\mathsf{F}} = (\!\vert\varphi\vert\!)_{\mathsf{F}}$
where $f \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}$ and $\varphi, \varphi' \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F})$	

The following equivalent definition is more descriptive: $in_{\mathsf{F}} : \mu\mathsf{F} \leftarrow \mathsf{F}(\mu\mathsf{F})$ is a constructor of F, provided that for every $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and every $\varphi : A \leftarrow \mathsf{F}A$ there exists a unique morphism $(\!\vert\varphi\vert\!)_{\mathsf{F}}$ such that the following diagram commutes:

$$\begin{tikzcd}
\mu\mathsf{F} \arrow[d, "(\!\vert\varphi\vert\!)_{\mathsf{F}}"'] & \mathsf{F}(\mu\mathsf{F}) \arrow[l, "in_{\mathsf{F}}"'] \arrow[d, "\mathsf{F}(\!\vert\varphi\vert\!)_{\mathsf{F}}"] \\
A & \mathsf{F}A \arrow[l, "\varphi"]
\end{tikzcd}$$

Dually to the above we define the category $\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F}$ of **F-co-algebras**, whose objects are the $\boldsymbol{\mathcal{C}}$-morphisms $\mathsf{F}A \leftarrow A$ for every $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$, with the **forgetful functor** $|\centerdot| : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F}$ defined by

$$|\psi| := \mathrm{dom}_{\boldsymbol{\mathcal{C}}}\,\psi \quad \forall\,\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F})$$

and

$$|f| := f \quad \forall\,f \in \mathrm{Mor}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F}).$$

If the category $\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F}$ has a final object, we denote it by

$$out_{\mathsf{F}} = 1_{\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F}} : \mathsf{F}(\nu\mathsf{F}) \underset{\boldsymbol{\mathcal{C}}}{\longleftarrow} \nu\mathsf{F}$$

and call it the **destructor** of F. The image of the mediator under $|\centerdot|$, *i.e.*

$$[\!(\psi)\!]_{\mathsf{F}} := |!_{\psi}| : \nu\mathsf{F} \underset{\boldsymbol{\mathcal{C}}}{\longleftarrow} \mathrm{dom}_{\boldsymbol{\mathcal{C}}}\,\psi$$

is called the **anamorphism** generated by ψ (w.r.t. F) (from Greek $'\alpha\nu\alpha$, upwards). Note that destructor and anamorphism are determined only unique up to F-co-algebra-isomorphism, and hence $\nu\mathsf{F}$ is determined only unique up to $\boldsymbol{\mathcal{C}}$-isomorphism. We have the following laws from Lemma 2.2.26:

laws for anamorphisms	
UP	$f = [\!(\psi)\!]_{\mathsf{F}} \iff out_{\mathsf{F}} \cdot f = \mathsf{F}f \cdot \psi$
reflection	$[\!(out_{\mathsf{F}})\!]_{\mathsf{F}} = id_{\nu\mathsf{F}}$
fusion	$\psi \cdot f = \mathsf{F}f \cdot \psi' \implies [\!(\psi)\!]_{\mathsf{F}} \cdot f = [\!(\psi')\!]_{\mathsf{F}}$
where $f \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}$ and $\psi, \psi' \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F})$	

Again we give a more descriptive equivalent definition: $out_{\mathsf{F}} : \mathsf{F}(\nu\mathsf{F}) \leftarrow \nu\mathsf{F}$ is a destructor of F, provided that for every $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and every $\psi : \mathsf{F}A \leftarrow A$ there exists a unique morphism $[\!(\psi)\!]_{\mathsf{F}}$, such that the following diagram commutes:

$$\begin{array}{ccc} B & \xrightarrow{\psi} & \mathsf{F}B \\ {\scriptstyle [\![\psi]\!]_{\mathsf{F}}}\downarrow & & \downarrow{\scriptstyle \mathsf{F}[\![\psi]\!]_{\mathsf{F}}} \\ \nu\mathsf{F} & \xrightarrow[out_{\mathsf{F}}]{} & \mathsf{F}(\nu\mathsf{F}) \end{array}$$

Proof. Immediately by Lemma 2.2.26. ∎

2.2.34. Note. In order to simplify the above definition, we have introduced $\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\,\mathsf{F}$ and $\boldsymbol{\mathcal{C}o\mathcal{A}lg_{\mathcal{C}}}\,\mathsf{F}$ as pre-categories. Actually we had to specify domain and co-domain for F-algebra-morphisms: $(\varphi, f, \varphi') \in \boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\,\mathsf{F}(\varphi, \varphi')$. This information will be forgotten by the forgetful functor $|(\varphi, f, \varphi')| = f$. We did this according to Definition 2.2.4. This is the case for $\boldsymbol{\mathcal{C}o\mathcal{A}lg_{\mathcal{C}}}\,\mathsf{F}$ as well.

2.2.35. Example. For the endofunctor $\mathsf{F} := \mathsf{Id} \times \mathsf{Id}$ in the category $\boldsymbol{\mathcal{S}et}$, *i.e.*

$$\mathsf{F} : \begin{cases} \boldsymbol{\mathcal{S}et} & \leftarrow \boldsymbol{\mathcal{S}et} \\ A \times A & \leftarrow\!\shortmid\ A \qquad \forall\, A \in \operatorname{Ob} \boldsymbol{\mathcal{S}et} \\ f \times f & \leftarrow\!\shortmid\ f \qquad \forall\, f \in \operatorname{Mor} \boldsymbol{\mathcal{S}et} \end{cases}$$

we consider the category $\boldsymbol{\mathcal{A}lg_{\mathcal{S}et}}\,\mathsf{F}$. An F-Algebra $\varphi \in \boldsymbol{\mathcal{A}lg_{\mathcal{S}et}}\,\mathsf{F}$ is a function

$$\varphi : A \xleftarrow[\boldsymbol{\mathcal{S}et}]{} A \times A$$

with an $A \in \operatorname{Ob} \boldsymbol{\mathcal{S}et}$. On the other hand an F-Algebra φ is a set $A := |\varphi|$ together with a binary operation $\star$, where

$$a \star b := \varphi(a, b) \quad \forall\, a, b \in A.$$

Let $\varphi, \varphi' \in \boldsymbol{\mathcal{A}lg_{\mathcal{S}et}}\,\mathsf{F}$ where $\varphi : A \leftarrow A \times A$ and $\varphi' : A' \leftarrow A' \times A'$. An F-algebra-morphism $f : \varphi \xleftarrow[\boldsymbol{\mathcal{A}lg_{\mathcal{S}et}}\,\mathsf{F}]{} \varphi$ is a function

$$f : A \xleftarrow[\boldsymbol{\mathcal{S}et}]{} A'$$

such that

$$f \cdot \varphi' = \varphi \cdot \mathsf{F}f,$$

i.e.

$$f\big(\varphi'(a, b)\big) = \varphi(fa, fb) \quad \forall\, a, b \in A.$$

If we denote the functions φ and φ' by the binary operations $\star$ and $\star'$, respectively, as above, we may write the last equation as follows:

$$f(a \star' b) = fa \star fb.$$

This is the well-known homomorphism property of f for algebras (with one binary function-symbol).

2.2.36. Lemma (Lambek's Lemma [BdM97]**).** Constructor and destructor are isomorphisms. In more detail: Let $\boldsymbol{\mathcal{C}}$ be a category and $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ be an endofunctor. If the category $\boldsymbol{\mathcal{A}lg_{\mathcal{C}}}\,\mathsf{F}$ has an initial object in_{F}, this is an isomorphism in $\boldsymbol{\mathcal{C}}$.

Proof.

$$
\begin{aligned}
& in_{\mathsf{F}} \cdot (\![\mathsf{F} in_{\mathsf{F}}]\!)_{\mathsf{F}} \\
= \quad & \{ \text{ fusion 2.2.33 } \} \\
& (\![in_{\mathsf{F}}]\!)_{\mathsf{F}} \\
= \quad & \{ \text{ reflection 2.2.33 } \} \\
& id
\end{aligned}
$$

and on the other hand

$$
\begin{aligned}
& (\![\mathsf{F} in_{\mathsf{F}}]\!)_{\mathsf{F}} \cdot in_{\mathsf{F}} \\
= \quad & \{ \text{ (UP) } \} \\
& \mathsf{F} in_{\mathsf{F}} \cdot \mathsf{F}(\![\mathsf{F} in_{\mathsf{F}}]\!)_{\mathsf{F}} \\
= \quad & \{ \; \mathsf{F} \text{ functor } \} \\
& \mathsf{F}\big(in_{\mathsf{F}} \cdot (\![\mathsf{F} in_{\mathsf{F}}]\!)_{\mathsf{F}}\big) \\
= \quad & \{ \text{ see above } \} \\
& \mathsf{F} id \\
= \quad & \{ \; \mathsf{F} \text{ functor } \} \\
& id.
\end{aligned}
$$

Thus, the constructor in_{F} is a $\mathcal{C}$-isomorphism and dually the destructor. ∎

3 Concrete categories

3.1 Concrete categories and constructs

3.1.1. Definition (concrete category, forgetful functor [AHS90]**).** Let $\mathcal{C}$ and $\mathcal{C}'$ be locally small categories and $\mathsf{U} : \mathcal{C} \leftarrow \mathcal{C}'$ be a faithful functor. The pair $(\mathcal{C}', \mathsf{U})$ is called the **concrete category** built upon $\mathcal{C}$ by U. The functor U is called the **forgetful functor** of $(\mathcal{C}', \mathsf{U})$. As a convention we will write most of the forgetful functors just $|\,\centerdot\,|$ if the connection to their concrete category is obvious. We say $\mathcal{C}'$ is built upon $\mathcal{C}$ to indicate that $(\mathcal{C}', |\,\centerdot\,|)$ is a concrete category built upon $\mathcal{C}$. For each $\mathcal{C}'$-object A we call $|A|$ the **underlying $\mathcal{C}$-object** of A and for each $\mathcal{C}'$-morphism f we call $|f|$ the **underlying $\mathcal{C}$-morphism** of f.

3.1.2. Definition (concrete functor [AHS90]**).** Let $\mathcal{C}$ be a locally small category and $(\mathcal{D}, \mathsf{U})$ and $(\mathcal{D}', \mathsf{U}')$ be concrete categories built upon $\mathcal{C}$. A functor $\mathsf{F} : \mathcal{D} \leftarrow \mathcal{D}'$ is called a **concrete functor** provided that

$$\mathsf{U} \cdot \mathsf{F} = \mathsf{U}'.$$

In this case we write

$$\mathsf{F} : (\mathcal{D}, \mathsf{U}) \leftarrow (\mathcal{D}', \mathsf{U}').$$

We denote the quasi-category of all concrete categories built upon $\mathcal{C}$ with all concrete functors as morphisms by $\boldsymbol{cCAT}\ \mathcal{C}$.

3.1.3. Example. The category $\boldsymbol{Cpo}$ is built upon $\boldsymbol{Set}$. The quasi-category $\boldsymbol{cCAT}\ \mathcal{C}$ is a quasi-concrete quasi-category built upon $\boldsymbol{CAT}$ by $|\,\centerdot\,| : \boldsymbol{CAT} \leftarrow \boldsymbol{cCAT}\ \mathcal{C} : \mathcal{C}' \hookleftarrow (\mathcal{C}', |\,\centerdot\,|)$.

3.1.4. Definition (construct, carrier-set, underlying set-function [AHS90]**).** A concrete category built upon $\boldsymbol{Set}$ is called a **construct**. We say the category $\mathcal{C}$ is a construct to indicate that $(\mathcal{C}, |\,\centerdot\,|)$ is a construct. For each $\mathcal{C}$-object A we call $|A|$ the **carrier-set** of A and for each $\mathcal{C}$-morphism f we call $|f|$ the **underlying set-function** of f.

3.1.5. Note. Let $(\mathcal{C}, \mathsf{U}_1)$ be a construct and $(\mathcal{D}, \mathsf{U}_2)$ be a concrete category built upon $\mathcal{C}$. Then $(\mathcal{D}, \mathsf{U}_2 \cdot \mathsf{U}_1)$ is also a construct. Since $\mathcal{C}$ is a construct, the $\mathcal{C}$-objects are sets with some additional structure. The forgetful functor U_1 forgets the additional structure of the $\mathcal{C}$-objects.

Since the forgetful functor is faithful, its hom-set restrictions are injections. Hence the hom-set-restrictions encode the information, which of the $\boldsymbol{Set}$-morphisms are $\mathcal{C}$-morphism.

The above is of course also true for the construct $(\mathcal{D}, \mathsf{U}_2 \cdot \mathsf{U}_1)$ and the concrete category $(\mathcal{D}, \mathsf{U}_2)$, respectively.

The concrete category $(\mathcal{D}, \mathsf{U}_2)$ is a view on the category $\mathcal{D}$ focusing on the additional structure, which is the difference between $\mathcal{C}$ and $\mathcal{D}$.

3.2 Enriched categories

3.2.1. Definition (enriched category). Let $\mathcal{C}$ be a locally small category, $(\mathcal{V}, \mathsf{U})$ be a construct and $\mathsf{H} : \mathcal{V} \leftarrow \mathcal{C} \times \mathcal{C}^{\mathrm{op}}$ a bi-functor. The pair $(\mathcal{C}, \mathsf{H})$ is called a **$(\mathcal{V}, \mathsf{U})$-enriched category** if

$$\mathsf{U} \cdot \mathsf{H} = \mathcal{C}(\,\centerdot\,, \centerdot\,).$$

Note that the hom-set restrictions of H are uniquely determined by the above equation, since U is faithful. As a convention we will write the infix symbol $\multimapinv$ for most of the bi-functors of enriched categories, if the connection to the enriched category is obvious. If the forgetful functor of the construct $\mathcal{V}$ is also obvious, we say $\mathcal{C}$ is a $\mathcal{V}$-category to indicate that $(\mathcal{C}, \multimapinv)$ is a $(\mathcal{V}, |\,\centerdot\,|)$-enriched category. In other words the category $\mathcal{C}$ is $\mathcal{V}$-enriched, if the $\mathcal{C}$-hom-sets are the carrier-sets of the $\mathcal{V}$-objects, and the hom-set restrictions of the composition in $\mathcal{C}$ are the underlying set-functions of $\mathcal{V}$-morphisms.

3.2.2. Example. Any locally small category is a $\boldsymbol{\mathcal{S}et}$-category. The categories $\boldsymbol{\mathcal{P}art}$ and $\boldsymbol{\mathcal{C}po}$ are $\boldsymbol{\mathcal{C}po}$-categories.

3.2.3. Definition. Let $\boldsymbol{\mathcal{V}}$ be a construct and $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{\mathcal{D}}$ be $\boldsymbol{\mathcal{V}}$-enriched categories. A functor $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{D}}$ is called a **$\boldsymbol{\mathcal{V}}$-functor**, if there exists a natural transformation

$$\tau : (\mathsf{F} \leftarrow\!\circ\, \mathsf{F}) \;\dot{\leftarrow}\; (\leftarrow\!\circ)$$

such that $|\tau| = \mathsf{F}_{(\,\centerdot\,,\,\centerdot\,)}$. Notice that in this case τ is uniquely determined because $|\,\centerdot\,|$ is faithful. In other words a functor is a $\boldsymbol{\mathcal{V}}$-functor, if its hom-set restrictions are the underlying set-functions of $\boldsymbol{\mathcal{V}}$-morphisms. We define for any $\varphi : A \leftarrow \mathsf{F}A$ and $\psi : \mathsf{F}B \leftarrow B$ the morphism

$$\varphi \overset{\mathsf{F}}{\leftarrow\!\circ}\, \psi := (\varphi \leftarrow\!\circ\, \psi) \cdot \tau \;\;:\;\; A \leftarrow\!\circ\, B \;\;\leftarrow\;\; A \leftarrow\!\circ\, B.$$

Notice that $|\varphi \overset{\mathsf{F}}{\leftarrow\!\circ}\, \psi| h = \varphi \cdot \mathsf{F}h \cdot \psi \quad \forall\, h \in \boldsymbol{\mathcal{V}}(A, B)$.

3.3 Involutory categories

3.3.1. Definition (self-dual category). A category which is isomorphic to its dual category is called **self-dual**.

3.3.2. Definition (involutory category [FP94]). A concrete category $(\boldsymbol{\mathcal{C}}, \mathsf{I})$ build upon $\boldsymbol{\mathcal{C}}^{\mathrm{op}}$ where

$$\mathsf{I}^{\mathrm{op}} \cdot \mathsf{I} = \mathsf{Id}_{\boldsymbol{\mathcal{C}}} \quad \text{and} \quad \mathsf{I} \cdot \mathsf{I}^{\mathrm{op}} = \mathsf{Id}_{\boldsymbol{\mathcal{C}}^{\mathrm{op}}}$$

is called an **involutory category** and the functor I is called an **involution**. Notice that in this case $\boldsymbol{\mathcal{C}}$ is self-dual because the involution I is an isomorphism.

3.3.3. Definition (Involutory closure [FP94]). Let $\boldsymbol{\mathcal{C}}$ be a category. We define the category $\breve{\boldsymbol{\mathcal{C}}}$ by

$$\begin{aligned} \mathrm{Ob}\,\breve{\boldsymbol{\mathcal{C}}} &:= \mathrm{Ob}\,\boldsymbol{\mathcal{C}} = \mathrm{Ob}\,\boldsymbol{\mathcal{C}}^{\mathrm{op}} \\ \breve{\boldsymbol{\mathcal{C}}}(A, B) &:= \boldsymbol{\mathcal{C}}(A, B) \times \boldsymbol{\mathcal{C}}(B, A) \quad \forall\, A, B \in \mathrm{Ob}\,\breve{\boldsymbol{\mathcal{C}}} \end{aligned}$$

and the functors $\grave{\mathsf{P}}$ and $(\,\centerdot\,)^*$ by

$$\grave{\mathsf{P}} : \begin{cases} \boldsymbol{\mathcal{C}} \;\leftarrow\; \breve{\boldsymbol{\mathcal{C}}} & \\ A \;\leftarrow\!\shortmid\; A & \forall\, A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} = \mathrm{Ob}\,\breve{\boldsymbol{\mathcal{C}}} \\ f \;\leftarrow\!\shortmid\; (f, g) & \forall\, (f, g) \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}} \times \mathrm{Mor}\,\boldsymbol{\mathcal{C}}^{\mathrm{op}} = \mathrm{Mor}(\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}^{\mathrm{op}}) = \mathrm{Mor}\,\breve{\boldsymbol{\mathcal{C}}} \end{cases}$$

and

$$(\,\centerdot\,)^* : \begin{cases} \breve{\boldsymbol{\mathcal{C}}}^{\mathrm{op}} \;\leftarrow\; \breve{\boldsymbol{\mathcal{C}}} & \\ A \;\leftarrow\!\shortmid\; A & \forall\, A \in \mathrm{Ob}\,\breve{\boldsymbol{\mathcal{C}}} \\ (g, f) \;\leftarrow\!\shortmid\; (f, g) & \forall\, (f, g) \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}} \times \mathrm{Mor}\,\boldsymbol{\mathcal{C}}^{\mathrm{op}} = \mathrm{Mor}(\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}^{\mathrm{op}}) = \mathrm{Mor}\,\breve{\boldsymbol{\mathcal{C}}} \end{cases}$$

where $\breve{\boldsymbol{\mathcal{C}}}^{\mathrm{op}} := (\breve{\boldsymbol{\mathcal{C}}})^{\mathrm{op}} = (\boldsymbol{\mathcal{C}}^{\mathrm{op}})\breve{\;}$. Obviously $(\breve{\boldsymbol{\mathcal{C}}}, {}^*)$ is an involutory category. It is called the **involutory closure** of $\boldsymbol{\mathcal{C}}$.

For each functor

$$\mathsf{H} : \boldsymbol{\mathcal{C}} \leftarrow \breve{\boldsymbol{\mathcal{C}}}$$

we define the functor $\breve{\mathsf{H}}$ by

$$\breve{\mathsf{H}} : \begin{cases} \breve{\boldsymbol{\mathcal{C}}} \;\leftarrow\; \breve{\boldsymbol{\mathcal{C}}} & \\ \mathsf{H}A \;\leftarrow\!\shortmid\; A & \forall\, A \in \mathrm{Ob}\,\breve{\boldsymbol{\mathcal{C}}} \\ \big(\mathsf{H}\,h, \mathsf{H}^{\mathrm{op}}h^*\big) \;\leftarrow\!\shortmid\; h & \forall\, h \in \mathrm{Mor}\,\breve{\boldsymbol{\mathcal{C}}}. \end{cases}$$

3.3.4. Definition. Let $\mathcal{C}$ be a category. We continue the functor $(\,.\,)^* : \breve{\mathcal{C}}^{\mathrm{op}} \leftarrow \breve{\mathcal{C}}$ component wise on the finite products

$$\begin{aligned} \breve{\mathcal{C}}^{0} &:= \mathbf{1} \\ \breve{\mathcal{C}}^{n+1} &:= \breve{\mathcal{C}}^{n} \times \breve{\mathcal{C}} \quad \forall\, n \in \mathbb{N}_0 \end{aligned}$$

by

$$\begin{aligned} (\,.\,)^*_{\mathbf{1}} &:= \mathsf{Id}_{\mathbf{1}} \\ (f,g)^*_{\breve{\mathcal{C}}^{n+1}} &:= (f^*_{\breve{\mathcal{C}}^{n}}, g^*) \quad \forall\, (f,g) \in \mathrm{Mor}\, \breve{\mathcal{C}}^{n} \times \mathrm{Mor}\, \breve{\mathcal{C}} = \mathrm{Mor}\, \breve{\mathcal{C}}^{n+1}. \end{aligned}$$

We will simply write * for all these functors.

4 Calculating with initial and final objects

In this chapter we will develop our tools to calculate with the morphisms of a category. The basic idea is to use the laws for initial and final objects (*cf.* Lemma 2.2.26). We consider a concrete category with an initial/final object and investigate the laws in the underlying category.

4.1 Deriving functors by initiality/finality

4.1.1. Theorem (deriving natural transformations and functors from initiality). Let $\mathcal{C}$ and $\mathcal{D}$ be categories and

$$\boldsymbol{CAT} \xleftarrow{|\,\boldsymbol{.}\,|} \boldsymbol{cCAT}\ \mathcal{C} \xleftarrow{\mathsf{E}} \mathcal{D}$$

where $(\boldsymbol{cCAT}\ \mathcal{C}, |\,\boldsymbol{.}\,|)$ is the default quasi-concrete quasi-category and $\mathsf{E} : \boldsymbol{cCAT}\ \mathcal{C} \leftarrow \mathcal{D}$ is a functor. Further let $0_D \in |\mathsf{E}D|$ be an initial object of the category $|\mathsf{E}D|$ where

$$¡_A^D : A \underset{|\mathsf{E}D|}{\longleftarrow} 0_D$$

is the unique mediating morphism for all $A \in \mathrm{Ob}\,|\mathsf{E}D|$ and all $D \in \mathrm{Ob}\,\mathcal{D}$. For each $D \in \mathrm{Ob}\,\mathcal{D}$ we denote the forgetful-functor by U_D where $(|\mathsf{E}D|, \mathsf{U}_D) = \mathsf{E}D$. We define the class-function τ by

$$\tau_A^D := \mathsf{U}_D ¡_A^D \quad \forall\, A \in \mathrm{Ob}\,|\mathsf{E}D| \quad \forall\, D \in \mathrm{Ob}\,\mathcal{D}$$

and the class-function F on objects by

$$\mathsf{F}D := \mathsf{U}_D 0_D \quad \forall\, D \in \mathrm{Ob}\,\mathcal{D}$$

and on morphisms by

$$\mathsf{F}d := \tau^D_{|\mathsf{E}d|0_E} \quad \forall\, d \in \mathcal{D}(D, D') \quad \forall\, D, D' \in \mathrm{Ob}\,\mathcal{D}.$$

Then the following holds:

(i) $\tau^D : \mathsf{U}_D \dot{\leftarrow} \mathsf{K}_{\mathsf{F}D}$ is a natural transformation for all $D \in \mathrm{Ob}\,\mathcal{D}$ and

(ii) $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{D}^{\mathrm{op}}$ is a functor.

Proof. (i) Immediately by Corollary 2.2.27 and Lemma 2.2.20 (i) pointwise for all $\mathcal{D}$-objects.

(ii) F respects typing since $\mathsf{E}d$ is a concrete functor, *i.e.* $\mathsf{U}_D \cdot |\mathsf{E}d| = \mathsf{U}_{D'}$ $\forall\, d : D \underset{\mathcal{D}}{\longleftarrow} D'$. That F respects composition is just an instance of the naturalness condition of τ^D for all $D \in \mathrm{Ob}\,\mathcal{D}$ which holds due to (i).

∎

Notice that the dual proposition (for finality) is also true.

4.1.2. Corollary. With the preconditions and notations from above we get immediately the following laws:

laws for F	
reflection	$\mathsf{F}id_D = id_{\mathsf{F}D}$
fusion (i)	$\tau_A^{D'} \cdot \mathsf{F}d = \tau^D_{|\mathsf{E}d|A}$
fusion (ii)	$\mathsf{F}e \cdot \mathsf{F}d = \mathsf{F}(e \cdot d)$
where $D, D' \in \mathrm{Ob}\,\mathcal{D}$ and $e, d \in \mathrm{Mor}\,\mathcal{D}$	

Theorem 4.1.1 enables us to construct functors for (co-)products, exponents and initial/final (co-)algebras. Notice that these functors are determined uniquely only up to isomorphism. We can conveniently translate the laws (reflection, fusion and cancelation) for (co-)mediators to rules for the according functors. In the case of initial/final (co-)algebras this directly leads to the acid rain theorem or to generalizations thereof.

4.2 The functors for (co-)products and exponents

4.2.1. Corollary (coproduct functor). On a category $\boldsymbol{\mathcal{C}}$ with finite sums we have the **sum-functor**, given by

$$\bullet + \bullet : \begin{cases} \boldsymbol{\mathcal{C}} & \leftarrow \ \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}} \\ A + B & \leftarrowtail \ (A, B) \quad \forall\, A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \\ [\grave{\imath} \cdot f, \acute{\imath} \cdot g] & \leftarrowtail \ (f, g) \quad \forall\, f, g \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}. \end{cases}$$

We declare that the binary operation symbol $+$ has lower precedence than the composition. This functor which satisfies the following laws

laws for +	
reflection	$id_A + id_B = id_{A+B}$
fusion (i)	$[h, i] \cdot (f + g) = [h \cdot f, i \cdot g]$
fusion (ii)	$(h + i) \cdot (f + g) = h \cdot f + i \cdot g$
cancelation	$(f + g) \cdot \grave{\imath} = \grave{\imath} \cdot f \ \wedge \ (f + g) \cdot \acute{\imath} = \acute{\imath} \cdot g$
where $A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and $f, g, h, i \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}$	

The injections are natural transformations in $\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}$:

$$\grave{\imath} : (+) \dot{\leftarrow} \grave{\mathsf{P}} \qquad \acute{\imath} : (+) \dot{\leftarrow} \acute{\mathsf{P}}.$$

4.2.2. Corollary (product functor). On a category $\boldsymbol{\mathcal{C}}$ with finite products we have the **product-functor** given by

$$\bullet \times \bullet : \begin{cases} \boldsymbol{\mathcal{C}} & \leftarrow \ \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}} \\ A \times B & \leftarrowtail \ (A, B) \quad \forall\, A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \\ \langle f \cdot \grave{\pi}, g \cdot \acute{\pi} \rangle & \leftarrowtail \ (f, g) \quad \forall\, f, g \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}, \end{cases}$$

We declare that the binary operation symbol $\times$ has higher precedence than $+$ but lower precedence than the composition. This functor satisfies the following laws:

laws for ×	
reflection	$id_A \times id_B = id_{A \times B}$
fusion (i)	$(f \times g) \cdot \langle h, i \rangle = \langle f \cdot h, g \cdot i \rangle$
fusion (ii)	$(f \times g) \cdot (h \times i) = f \cdot h \times g \cdot i$
cancelation	$\grave{\pi} \cdot (f \times g) = f \cdot \grave{\pi} \ \wedge \ \acute{\pi} \cdot (f \times g) = g \cdot \acute{\pi}$
where $A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and $f, g, h, i \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}$	

The projections are natural transformations in $\boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}$:

$$\grave{\pi} : \grave{\mathsf{P}} \dot{\leftarrow} (\times) \qquad \acute{\pi} : \acute{\mathsf{P}} \dot{\leftarrow} (\times).$$

4.2.3. Definition (Cartesian category). Let $\boldsymbol{\mathcal{C}}$ be a category with terminal object $1 \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and $\times : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}$ be a product functor. The triple $(\boldsymbol{\mathcal{C}}, \times, 1)$ is called a **Cartesian category**[6].

4.2.4. Corollary (exponent functor). On a Cartesian category $(\boldsymbol{\mathcal{C}}, \times, 1)$ with exponents for pairs of $\boldsymbol{\mathcal{C}}$-objects we have the **exponent-functor** $\Leftarrow : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}^{\mathrm{op}}$, given by

$$f \Leftarrow g := \mathrm{curry}\big(f \cdot \mathit{apply}_{(B,C)} \cdot (id_{B^C} \times g)\big)$$

for all $f : A \leftarrow B$ and $g : C \leftarrow D$. We declare that the binary operation symbol $\Leftarrow$ has higher precedence than $\times$ but lower precedence than the composition. The following laws hold:

[6] This is a special case of a so-called **monoidal category** $(\boldsymbol{\mathcal{C}}, \otimes, I)$.

laws for $\Leftarrow$	
reflection	$id_A \Leftarrow id_B = id_{A^B}$
fusion (i)	$(f \Leftarrow id) \cdot \text{curry}\, g = \text{curry}(f \cdot g)$
fusion (ii)	$(f \Leftarrow g) \cdot (h \Leftarrow i) = f \cdot h \Leftarrow i \cdot g$
cancelation	$apply \cdot ((f \Leftarrow id) \times id) = f \cdot apply$
where $A, B \in \text{Ob}\,\mathcal{C}$ and $f, g, h, i \in \text{Mor}\,\mathcal{C}$	

The application is a natural transformation:

$$apply \ : \ \mathsf{Id} \ \dot{\leftarrow} \ (\mathsf{Id} \Leftarrow \mathsf{K}_B \ \times \ \mathsf{K}_C).$$

4.2.5. Definition (Cartesian closed category). Let $(\mathcal{C}, \times, 1)$ be a Cartesian category and $\Leftarrow : \mathcal{C} \leftarrow \mathcal{C} \times \mathcal{C}^{\text{op}}$ an exponent functor w.r.t. $\times$. The 4-tuple $(\mathcal{C}, \times, 1, \Leftarrow)$ is called a **Cartesian closed category** or **CCC**.

4.2.6. Definition (internal fixed point operator). Let $\mathcal{C}$ be a CCC and $A \in \text{Ob}\,\mathcal{C}$. A morphism $Y_A : A \leftarrow A^A$ such that

$$Y_A = apply_{(A,A)} \cdot (id_{A^A} \times Y_A) \cdot \Delta_{A^A}$$

is called an **internal fixed point operator** for the object A in $\mathcal{C}$.

4.2.7. Lemma. A $\boldsymbol{\mathcal{C}po}$-CCC has an internal fixed point operator given by

$$Y = \text{uncurry}\Big(\text{fix}\big((apply \Leftarrow id) \leftarrow\!\!\circ\, id\big)\Big) \cdot unit^{-}1.$$

Proof. See [Bra97]. ∎

4.3 Generalized acid rain theorems

4.3.1. Corollary (generalized acid rain theorem). Let $\mathcal{C}$ be a category and $A \in \text{Ob}\,\mathcal{C}$. Let (D, U_D) and $(D', \mathsf{U}_{D'})$ be concrete categories such that the underlying categories D and D' have initial objects. With the notations from Theorem 4.1.1 we get

$$\frac{\mathsf{H} : (D, |\centerdot|_D) \leftarrow (D', |\centerdot|_{D'})}{\tau_A^{D'} \ \cdot \ \tau_{\mathsf{H}0_{D'}}^{D} = \tau_{\mathsf{H}A}^{D}}$$

4.3.2. Corollary (catamorphism fusion or acid rain theorem [TM95]**).** Let $\mathcal{C}$ be a category and $\mathsf{F}, G : \mathcal{C} \leftarrow \mathcal{C}$ endofunctors such that the categories $\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F}$ and $\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}$ have initial objects. Let $\varphi \in \text{Ob}(\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F})$. Then the following holds:

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F}, |\centerdot|_{\mathsf{F}}) \xleftarrow[\mathcal{D}]{} (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}, |\centerdot|_{\mathsf{G}})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}.$$

4.3.3. Note. It is worth mentioning that the above is just a corollary of Theorem 4.1.1. The classical acid rain theorem (*cf.* [TM95] Theorem 3.2) requires a naturalness precondition. This needs the **parametricity** property or **free theorem**, *i.e.* polymorphic functions of the language will be mapped onto *natural* transformations by the categorical semantics (*cf.* [Wad89, dB89]). Our new approach is entirely different, since neither language nor semantics occured — it is pure category theory.

4.3.4. Corollary (functorial acid rain theorem). With the preconditions and notations from above the class-function defined by

$$\mathsf{MH} := (\!|\mathsf{H}in_{\mathsf{G}}|\!)_{\mathsf{F}} \quad \forall\, \mathsf{H} \in \boldsymbol{\mathcal{D}}\big((Alg_{\mathcal{C}}\mathsf{F}, |\centerdot|_{\mathsf{F}}), (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}, |\centerdot|_{\mathsf{G}})\big)$$

is a functor $\mathsf{M} : \mathcal{C} \leftarrow \mathcal{D}^{\mathrm{op}}$, *i.e.*:

$$\frac{(\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}_1, |\,.\,|_{\mathsf{F}_1}) \xleftarrow[\mathcal{D}]{\mathsf{H}} (\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}_2, |\,.\,|_{\mathsf{F}_2}) \xleftarrow[\mathcal{D}]{\mathsf{H}'} (\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}_3, |\,.\,|_{\mathsf{F}_3})}{(\!|\mathsf{H}' in_{\mathsf{F}_3}|\!)_{\mathsf{F}_2} \cdot (\!|\mathsf{H} in_{\mathsf{F}_2}|\!)_{\mathsf{F}_1} = (\!|(\mathsf{H}\cdot\mathsf{H}') in_{\mathsf{F}_3}|\!)_{\mathsf{F}_1}}.$$

Using our proof technique, the acid rain theorem Corollary 4.3.2 may be generalized to:

4.3.5. Theorem (generalized acid rain theorem II). For a faithful functor $\mathsf{E} : \mathcal{C}_\perp \leftarrow \mathcal{C}_\perp$; $\mathsf{F}, \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol}\,\mathcal{C})$ and $\varphi \in \mathrm{Ob}(\mathcal{A}lg\,\mathsf{F})$ holds:

$$\frac{\mathsf{H} : (\mathcal{A}lg\,\mathsf{F}, |\,.\,|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg\,\mathsf{G}, \mathsf{E}\cdot|\,.\,|_{\mathsf{G}})}{\mathsf{E}(\!|\varphi|\!)_{\mathsf{G}} \cdot (\!|\mathsf{H} in_{\mathsf{G}}|\!)_{\mathsf{F}} = (\!|\mathsf{H}\varphi|\!)_{\mathsf{F}}}$$

and for a faithful bifunctor $\otimes : \mathcal{C}_\perp \leftarrow \mathcal{C}_\perp \times \mathcal{C}_\perp$; $\mathsf{F}\otimes\mathsf{F}', \mathsf{G}, \mathsf{G}' \in \mathrm{Ob}(\boldsymbol{Pol}\,\mathcal{C})$ and $\varphi \in \mathrm{Ob}(\mathcal{A}lg\,\mathsf{G})$, $\varphi' \in \mathrm{Ob}(\mathcal{A}lg\,\mathsf{G}')$ we have:

$$\frac{\mathsf{H} : \big(\mathcal{A}lg(\mathsf{F}\otimes\mathsf{F}'), |\,.\,|_{\mathsf{F}\otimes\mathsf{F}'}\big) \leftarrow (\mathcal{A}lg\,\mathsf{G}\times\mathcal{A}lg\,\mathsf{G}', |\,.\,|_{\mathsf{G}}\otimes|\,.\,|_{\mathsf{G}'})}{\big((\!|\varphi|\!)_{\mathsf{G}} \otimes (\!|\varphi'|\!)_{\mathsf{G}'}\big) \cdot (\!|\mathsf{H}(in_{\mathsf{G}}, in_{\mathsf{G}'})|\!)_{\mathsf{F}\otimes\mathsf{F}'} = (\!|\mathsf{H}(\varphi,\varphi')|\!)_{\mathsf{F}\otimes\mathsf{F}'}}$$

These two fusion laws (and their duals) enable us to fuse functions, which are not composed directly, *i.e.*, there is a third function, which is a projection, injection, *etc.*, in between the composition. The second law can be instantiated *e.g.* with $\otimes = \times$ to calculate fusions of the form $(\!|\,.\,|\!)\cdot\check{\pi}\cdot(\!|\,.\,|\!) = \check{\pi}\cdot(\!|\,.\,|\!)$. It turns out that we are able to conveniently derive the above functors H, for both of the rules, by means of a substitution operator, analogically to Lemma 6.4.11. In fact we just need to substitute the constructor, like we did before.

4.4 Characterization of concrete functors

4.4.1. Lemma (characterization of concrete functors on categories of algebras). Let $\mathcal{C}$ be a locally small category, $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{C}$ be endo-functors and $\mathsf{H} : \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}) \leftarrow \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G})$ a class-function. The function H can be uniquely continued on $\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G}$-morphisms to a concrete functor

$$\mathsf{H} : (\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}, |\,.\,|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G}, |\,.\,|_{\mathsf{G}})$$

iff it satisfies the following condition:

$$\frac{\varphi\cdot\mathsf{G}f = f\cdot\varphi'}{\mathsf{H}\varphi\cdot\mathsf{F}f = f\cdot\mathsf{H}\varphi'} \qquad (*)$$

for all $\varphi, \varphi' \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G})$ and all $f : |\varphi|_{\mathsf{G}} \xleftarrow[\mathcal{C}]{} |\varphi'|_{\mathsf{G}}$. Notice the different usage of functors F, G on morphisms and H on objects in the above equations.

Proof. Since we want the functor H to be concrete, *i.e.* $|\,.\,|_{\mathsf{F}}\cdot\mathsf{H} = |\,.\,|_{\mathsf{G}}$ and since $|\,.\,|_{\mathsf{F}}$ and $|\,.\,|_{\mathsf{G}}$ are faithful, the only continuation of H on morphisms (respecting typing) is

$$\mathsf{H}(\varphi, f, \varphi') := (\mathsf{H}\varphi, f, \mathsf{H}\varphi') \quad \forall\,(\varphi, f, \varphi') \in \mathcal{A}lg_{\mathcal{C}}\,\mathsf{G}(\varphi,\varphi') \quad \forall\,\varphi,\varphi' \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}).$$

Since this function operates trivially on the underlying $\mathcal{C}$-morphisms it obviously respects composition in $\mathcal{C}$ and thus also in $\mathrm{Mor}(\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G})$. In order to see that it is a functor

$$\mathsf{H} : (\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}, |\,.\,|_{\mathsf{F}}) \leftarrow (\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G}, |\,.\,|_{\mathsf{G}})$$

we have to check that $\mathcal{A}lg_{\mathcal{C}}\,\mathsf{G}$-morphisms will be mapped onto $\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F}$-morphism. But this property is nothing else then the above condition $(*)$. ∎

4.4.2. Corollary (characterization of concrete functors on categories of algebras). With the above preconditions we get

$$\boldsymbol{cCAT}\ \big((\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F}, |\centerdot|_{\mathsf{F}}), (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}, |\centerdot|_{\mathsf{G}})\big) \cong \mathcal{C}^{\mathcal{C}}(|\centerdot|_{\mathsf{G}}, \mathsf{F}\cdot|\centerdot|_{\mathsf{G}}),$$

because the naturalness condition for the natural transformations $\tau : |\centerdot|_{\mathsf{G}} \dot{\leftarrow} \mathsf{F}\cdot|\centerdot|_{\mathsf{G}}$ is just the property $(*)$ from Lemma 4.4.1 if we set $\mathsf{H}\varphi := \tau_\varphi$ for all $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G})$.

4.4.3. Corollary. With the above preconditions, each of the following definitions yields a $\mathsf{H} : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F}, |\centerdot|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}, |\centerdot|_{\mathsf{G}})$. For all $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G})$:

(i) $\mathsf{H}\varphi := \varphi\cdot\mathsf{G}\varphi$ where $\mathsf{F} = \mathsf{G}\cdot\mathsf{G}$,

(ii) $\mathsf{H}\varphi := \varphi\cdot\tau$ where $\tau : \mathsf{G} \dot{\leftarrow} \mathsf{F}$,

(iii) $\mathsf{H}\varphi := \tau$ where $\tau : \mathsf{Id} \dot{\leftarrow} \mathsf{F}$,

(iv) $\mathsf{H}\varphi := [\mathsf{H}_1\varphi, \mathsf{H}_2\varphi]$, where $\mathsf{H}_1 : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F}_1, |\centerdot|_{\mathsf{F}_1}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}, |\centerdot|_{\mathsf{G}})$ and $\mathsf{H}_2 : (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{F}_2, |\centerdot|_{\mathsf{F}_2}) \leftarrow (\boldsymbol{\mathcal{A}lg}_{\mathcal{C}}\,\mathsf{G}, |\centerdot|_{\mathsf{G}})$ and $\mathsf{F} = \mathsf{F}_1 + \mathsf{F}_2$,

5 Hylomorphisms

5.1 Pointed Categories

5.1.1. Definition (pointed category). Let $\mathcal{C}$ be a category. A $\mathcal{C}$-morphism $f : A \leftarrow B$ is called a **zero-morphism** if

$$f \cdot r = f \cdot s \quad \forall\, r, s : B \leftarrow B' \quad \forall\, B' \in \mathrm{Ob}\,\mathcal{C}$$

and

$$p \cdot f = q \cdot f \quad \forall\, p, q : A' \leftarrow A \quad \forall\, A' \in \mathrm{Ob}\,\mathcal{C}$$

hold. If every $\mathcal{C}$-morphism-class contains a zero-morphism, then $\mathcal{C}$ is called **pointed**.

5.1.2. Lemma. (i) Let $\mathcal{C}$ be a category with zero-object $0 \in \mathrm{Ob}\,\mathcal{C}$. A $\mathcal{C}$-morphism $f : A \leftarrow B$ is a zero-morphism iff $f = ¡_A \cdot !_B$.

(ii) A category is pointed iff it is a full subcategory of a category with a zero-object.

Proof. (i) Immediately by $¡_A \cdot \underbrace{!_A \cdot f \cdot ¡_B}_{=id_0} \cdot !_B = f$.

(ii) See [AHS90].

■

5.1.3. Definition. Let $\mathcal{C}$ be a category with a final object $1 \in \mathrm{Ob}\,\mathcal{C}$ and let $\perp : \mathsf{Id} \leftarrow \mathsf{K}_1$ be a (*not* necessarily natural) transformation. A morphism $f : A \underset{\mathcal{C}}{\leftarrow} B$ is called **strict** w.r.t. $\perp$, provided that

$$f \cdot \perp_B = \perp_A \quad \forall\, C \in \mathrm{Ob}\,\mathcal{C}.$$

We define the category $\mathcal{C}_\perp$ by

$$\begin{array}{rcl} \mathrm{Ob}\,\mathcal{C}_\perp & := & \mathrm{Ob}\,\mathcal{C} \\ \mathrm{Mor}\,\mathcal{C} & := & \{ f \in \mathrm{Mor}\,\mathcal{C} \mid f \text{ is strict w.r.t. } \perp \}. \end{array}$$

It is easy to see that $\mathcal{C}_\perp$ is indeed a category with the composition inherited from $\mathcal{C}$.

5.1.4. Lemma. Let $\mathcal{C}$ be a category with a final object $1 \in \mathrm{Ob}\,\mathcal{C}$ and let $\perp : \mathsf{Id} \leftarrow \mathsf{K}_1$ be a (*not* necessarily natural) transformation.

(i) $\perp_A$ and $!_A$ are strict w.r.t. $\perp$ for every $A \in \mathrm{Ob}\,\mathcal{C}$ and $\perp : \mathsf{Id}_{\mathcal{C}_\perp} \dot{\leftarrow} \mathsf{K}_1^{\mathcal{C}_\perp}$ is a *natural* transformation in $\mathcal{C}_\perp$.

(ii) 1 is a zero-object in $\mathcal{C}_\perp$ and $\mathcal{C}_\perp$ is pointed.

(iii) $\mathcal{C}_\perp = \mathcal{C}$ iff $\perp$ is a *natural* transformation in $\mathcal{C}$.

Proof. Immediately with $\perp_1 = !_1 = id_1$. ■

5.1.5. Lemma. Let $\mathcal{C}$ be a $\mathcal{C}po$-category and, for every $\mathcal{C}$-objects A and B, let $\perp_{A,B}$ denote the least morphism of $\mathcal{C}(A, B)$, *i.e.* $|\perp_{A,B}| = \min(A \leftarrow\!\circ\, B)$. By $\perp_A := \perp_{A,1} \quad \forall\, A \in \mathrm{Ob}\,\mathcal{C}$ we get a transformation $\perp : \mathsf{Id} \leftarrow \mathsf{K}_1$.

If $\mathcal{C} = \mathcal{C}po$ then $\mathcal{C}$ is not pointed and $\perp$ is not natural. Furthermore the notation $\mathcal{C}po_\perp$ for the category of all strict ω-continuous functions is consistent with the notation $\mathcal{C}_\perp$ for the category of all strict $\mathcal{C}$-morphisms w.r.t. $\perp$.

Proof. With the above definition of the transformation $\perp$ it is easy to see that the strict CPO-homomorphisms precisely are the strict $\boldsymbol{\mathcal{C}po}$-morphisms and thus indeed $\boldsymbol{\mathcal{C}}_\perp = \boldsymbol{\mathcal{C}po}_\perp$. The category $\boldsymbol{\mathcal{C}po}$ is not pointed, since it has no initial object. $\boldsymbol{\mathcal{C}po}_\perp$ is a proper subcategory of $\boldsymbol{\mathcal{C}po}$, hence $\perp$ can not be natural. ∎

5.1.6. Lemma. Every pointed CCC is trivial.

Proof. See [AHS90] 27.A. ∎

5.2 Polynomial Functors

In this subsection, let $\boldsymbol{\mathcal{C}}$ be a category with a final object $1 \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ and let $\perp : \mathsf{Id} \leftarrow \mathsf{K}_1$ be a transformation, such that $(\boldsymbol{\mathcal{C}}, \times, 1, \Leftarrow)$ is a CCC and $({\boldsymbol{\mathcal{C}}_\perp}^{\mathrm{op}}, +^{\mathrm{op}}, 1)$ is Cartesian.

5.2.1. Definition (polynomial functor). The category $\boldsymbol{Pol\,\mathcal{C}}$ of **polynomial functors** is defined to be the full subcategory of the quasi-category $\boldsymbol{\mathcal{C}}^{\boldsymbol{\mathcal{C}}_\perp}$ where the object-class is given by the following inference-rules:

$$\frac{}{\mathsf{Id}_{\boldsymbol{\mathcal{C}}_\perp} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})} \qquad \frac{A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}}{\mathsf{K}_A^{\boldsymbol{\mathcal{C}}_\perp} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})}$$

$$\frac{\mathsf{F}, \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})}{\mathsf{F} \times \mathsf{G}, \mathsf{F} + \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})} \qquad \frac{\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}}) \quad B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}}{\mathsf{F} \Leftarrow \mathsf{K}_B \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})}.$$

5.2.2. Lemma. We immediately can state several morphisms from $\mathrm{Mor}(\boldsymbol{Pol\,\mathcal{C}})$. Let $\mathsf{F}, \mathsf{G}, \mathsf{H} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ and $B, C \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$.

$$\begin{array}{rcccc} \grave{\imath} & : & \mathsf{F} + \mathsf{G} & \dot{\leftarrow} & \mathsf{F} \\ \acute{\imath} & : & \mathsf{F} + \mathsf{G} & \dot{\leftarrow} & \mathsf{G} \\ \grave{\pi} & : & \mathsf{F} & \dot{\leftarrow} & \mathsf{F} \times \mathsf{G} \\ \acute{\pi} & : & \mathsf{G} & \dot{\leftarrow} & \mathsf{F} \times \mathsf{G} \\ \mathit{apply} & : & \mathsf{F} & \dot{\leftarrow} & \mathsf{F} \Leftarrow \mathsf{K}_B \times \mathsf{K}_{\boldsymbol{\mathcal{C}}} \end{array}$$

and with the natural transformations from Lemma 2.2.30:

$$\begin{array}{rcccc} \mathit{unit} & : & \mathsf{F} & \dot{\leftarrow} & \mathsf{K}_1 \times \mathsf{F} \\ \mathit{swap} & : & \mathsf{F} \times \mathsf{G} & \dot{\leftarrow} & \mathsf{G} \times \mathsf{F} \\ \mathit{assocr} & : & \mathsf{F} \times (\mathsf{G} \times \mathsf{H}) & \dot{\leftarrow} & (\mathsf{F} \times \mathsf{G}) \times \mathsf{H} \\ \mathit{assocl} & : & (\mathsf{F} \times \mathsf{G}) \times \mathsf{H} & \dot{\leftarrow} & \mathsf{F} \times (\mathsf{G} \times \mathsf{H}). \end{array}$$

Proof. By means of the diagonal functor

$$\mathsf{D} : \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} : (A, A) \leftarrow\!\shortmid A : (f, f) \leftarrow\!\shortmid f$$

we can use the notation from Lemma 2.2.20 (ii) and write

$$\grave{\imath} = \grave{\imath}(\mathsf{F}, \mathsf{G})\mathsf{D} = \grave{\imath}\big((\mathsf{F}, \mathsf{G}) \cdot \mathsf{D}\big).$$

Since $(+) \cdot (\mathsf{F}, \mathsf{G}) \cdot \mathsf{D} = \mathsf{F} + \mathsf{G}$ and $\grave{\mathsf{P}} \cdot (\mathsf{F}, \mathsf{G}) \cdot \mathsf{D} = \mathsf{F}$ the above proposition (i) follows, using Lemma 2.2.20 (ii), from $\grave{\imath} : (+) \dot{\leftarrow} \grave{\mathsf{P}}$ from Definition and Lemma 2.2.31. The remaining propositions can be shown analogically. ∎

5.2.3. Lemma. Let $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ be a polynomial functor. The category $\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}_\perp}\,\mathsf{F}$ has an initial and the category $\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F}$ has a final object.

Proof. See [AHS90] 13.4 and [Jür99] or [Tüf98]. ∎

5.2.4. Lemma. Polynomial functors are faithful.

Proof. Since $\mathcal{C}_\perp$ is pointed and a full subcategory of $\mathcal{C}$, the projections $\grave{\pi}$, $\acute{\pi}$ are retractions, *i.e.* have a right inverse:

$$\grave{\pi}_{(A,B)} \cdot \langle id_A, \perp_{(B,A)}\rangle = id_A \quad \text{and} \quad \acute{\pi}_{(A,B)} \cdot \langle \perp_{(A,B)}, id_B\rangle = id_B \quad \forall\, A, B \in \mathrm{Ob}\,\mathcal{C}.$$

Together with the cancelation rule it follows directly from the definition of the product functor $\times$ that it is faithful. Analogically, this holds also for the functors $\times$ and $\Leftarrow$, and thus by structural induction for polynomial functors. ∎

5.3 Hylomorphisms

5.3.1. Definition (hylomorphism [TM95]**).** Let $\mathcal{C}$ be a $\mathcal{C}po$-category, $\mathsf{F} : \mathcal{C} \leftarrow \mathcal{C}$ a $\mathcal{C}po$-functor, $\varphi \in \mathrm{Ob}(\mathcal{A}lg_{\mathcal{C}_\perp}\,\mathsf{F})$ and $\psi \in \mathrm{Ob}(\mathcal{C}o\mathcal{A}lg_{\mathcal{C}}\,\mathsf{F})$. The morphism

$$[\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}} := \mathrm{fix}(\varphi \overset{\mathsf{F}}{\hookleftarrow} \psi)$$

is called the **hylomorphism**[7] (from Greek 'ὑλη, substance) generated by φ and ψ w.r.t. F. Where fix denotes the fixed point operator on CPOs according to Definition 2.1.12 and $\varphi \overset{\mathsf{F}}{\hookleftarrow} \psi$ as defined in Definition 3.2.3. Notice that because of Definition 3.2.3, $\varphi \overset{\mathsf{F}}{\hookleftarrow} \psi$ is a CPO-endomorphism on the CPO $\mathcal{C}(|\varphi|_{\mathsf{F}}, |\psi|_{\mathsf{F}})$ and thus $\mathrm{fix}(\varphi \overset{\mathsf{F}}{\hookleftarrow} \psi)$ is defined (*cf.* Definition 2.1.12).

5.3.2. Lemma (hylo-shift [TM95]**).** Let $\mathcal{C}$ be a $\mathcal{C}po$-category, $\mathsf{F}, \mathsf{G} : \mathcal{C} \leftarrow \mathcal{C}$ be $\mathcal{C}po$-functors, $\varphi : A \leftarrow \mathsf{F}A$ strict, $\psi : \mathsf{G}B \leftarrow B$ and $A, B \in \mathrm{Ob}\,\mathcal{C}$. Then the following holds:

$$\frac{\tau : \mathsf{F} \dot{\leftarrow} \mathsf{G} \quad \text{in } \mathcal{C}_\perp}{[\![\, \varphi \cdot \tau_A,\, \psi \,]\!]_{\mathsf{G}} = [\![\, \varphi,\, \tau_B \cdot \psi \,]\!]_{\mathsf{F}}}$$

Proof. We need the naturalness of τ:

$$\begin{array}{ll}
 & \tau : \mathsf{F} \dot{\leftarrow} \mathsf{G} \quad \text{in } \mathcal{C}_\perp \\
\Longrightarrow & \quad \{\text{ Definition 2.2.17 }\} \\
 & \tau_A \cdot \mathsf{G}h = \mathsf{F}h \cdot \tau_B \quad \forall\, h : A \leftarrow B \\
\Longrightarrow & \quad \{\text{ The composition ist a function. }\} \\
 & \varphi \cdot \tau_A \cdot \mathsf{G}h \cdot \psi = \varphi \cdot \mathsf{F}h \cdot \tau_B \cdot \psi \quad \forall\, h : A \leftarrow B \\
\Longrightarrow & \quad \{\text{ definition of } \hookleftarrow \text{ in 3.2.3 }\} \\
 & \varphi \cdot \tau_A \overset{\mathsf{G}}{\hookleftarrow} \psi = \varphi \overset{\mathsf{F}}{\hookleftarrow} \tau_B \cdot \psi \\
\Longrightarrow & \quad \{\text{ definition of } [\![\, \cdot\,,\, \cdot \,]\!] \text{ in 5.3.1 }\} \\
 & [\![\, \varphi \cdot \tau_A,\, \psi \,]\!]_{\mathsf{G}} = \mathrm{fix}(\varphi \cdot \tau_A \overset{\mathsf{G}}{\hookleftarrow} \psi) = \mathrm{fix}(\varphi \overset{\mathsf{F}}{\hookleftarrow} \tau_B \cdot \psi) = [\![\, \varphi,\, \tau_B \cdot \psi \,]\!]_{\mathsf{F}}
\end{array}$$ ∎

5.4 Hylomorphism-Fusion

From now on to the end of the paper we fix a category $\mathcal{C}$ which satisfies

- $(\mathcal{C}, \times, 1, \Leftarrow)$ is a CCC.
- $\mathcal{C}$ is a $\mathcal{C}po$-category and we define the transformation $\perp : \mathsf{Id} \leftarrow \mathsf{K}_1$ as in Lemma 5.1.5.

[7] The double-brackets $[\![\, \cdot\,,\, \cdot \,]\!]$ are commonly used to write hylomorphisms. Unfortunately, they look like the 'semantic-brackets' used in computer science. Therefore, we will use braces and brackets $\{\![\, \cdot \,]\!\}$ to denote the semantics in Section 6.

- $(\boldsymbol{\mathcal{C}_\perp}^{\mathrm{op}}, +^{\mathrm{op}}, 1)$ is Cartesian.
- $\boldsymbol{\mathcal{C}}$ has final co-algebras for polynomial functors.
- $\boldsymbol{\mathcal{C}_\perp}$ has initial algebras for polynomial functors.

We have already seen that the simplification $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{C}_\perp}$ implies that $\boldsymbol{\mathcal{C}}$ is trivial and that the (nontrivial) category $\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{C}po}$ satisfies all the above conditions. In the preceding sections we prepared the tools to work with such a category $\boldsymbol{\mathcal{C}}$:

- The fixed point operator yielding the least fixed point of CPO-homomorphisms.
- The unique mediating morphisms of specific initial or final objects and the according laws:
  - reflection
  - fusion
  - cancelation
- The natural isomorphisms and their properties.

For every functor $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ we define the abbreviation

$$\boldsymbol{\mathcal{A}lg}\,\mathsf{F} := \boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}_\perp}}\,\mathsf{F} \qquad \text{and} \qquad \boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F} := \boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F},$$

i.e. F-algebras are always assumed to be strict but F-co-algebras are not.

5.4.1. Lemma. Let $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ be a polynomial functor.

(i) F is a $\boldsymbol{\mathcal{C}po}$-functor.

(ii) F respects the strictness of morphisms, *i.e.* $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}_\perp})$.

Proof. (i) Let $A, B, C \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$. We show at first

$$\langle\,\centerdot\,,\,\centerdot\,\rangle : \boldsymbol{\mathcal{C}}(A \times B, C) \xleftarrow[\boldsymbol{\mathcal{C}po}]{} \boldsymbol{\mathcal{C}}(A, C) \times \boldsymbol{\mathcal{C}}(B, C) \tag{$*$}$$

We only need to demonstrate that $\langle\,\centerdot\,,\,\centerdot\,\rangle$ respects the supremum of ω-chains: Let

$$(f_i)_i^\infty \in \boldsymbol{\mathcal{C}}(A, C)^\omega \qquad (g_i)_i^\infty \in \boldsymbol{\mathcal{C}}(B, C)^\omega.$$

We set

$$f := \bigsqcup_{i=0}^{\infty} f_i \qquad g := \bigsqcup_{i=0}^{\infty} g_i \qquad h := \bigsqcup_{i=0}^{\infty} \langle f_i, g_i\rangle$$

and have to show $h = \langle f, g\rangle$:

$$
\begin{aligned}
& \grave{\pi} \cdot h \\
= \quad & \{ \text{ see above } \} \\
& \grave{\pi} \cdot \bigsqcup_{i=0}^{\infty} \langle f_i, g_i\rangle \\
= \quad & \{ \text{ the composition in } \boldsymbol{\mathcal{C}} \text{ is } \omega\text{-continuous. } \} \\
& \bigsqcup_{i=0}^{\infty} \grave{\pi} \cdot \langle f_i, g_i\rangle \\
= \quad & \{ \text{ cancelation } \} \\
& \bigsqcup_{i=0}^{\infty} f_i \\
= \quad & \{ \text{ see above } \} \\
& f
\end{aligned}
$$

Analogically, $\acute{\pi} \cdot h = g$ holds. With the UP we get $h = \langle f, g \rangle$ and thus $(*)$. Now let $D, E \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$. From the definition of the product functor in Corollary 4.2.2 and the ω-continuity of the $\boldsymbol{\mathcal{C}}$-composition we get:

$$\bullet \times \bullet : \boldsymbol{\mathcal{C}}(A \times B, D \times E) \xleftarrow[\boldsymbol{\mathcal{C}po}]{} \boldsymbol{\mathcal{C}}(A, D) \times \boldsymbol{\mathcal{C}}(B, E)$$

Since $A, B, D, E \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ had been chosen arbitrarily

$$\bullet \times \bullet : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}}$$

is a $\boldsymbol{\mathcal{C}po}$-functor. Analogically, one can show that

$$\bullet + \bullet : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \times \boldsymbol{\mathcal{C}} \quad \text{and} \quad \bullet \Leftarrow id_B : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}} \quad \forall B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$$

are $\boldsymbol{\mathcal{C}po}$-functors. Since Id and K are also obviously $\boldsymbol{\mathcal{C}po}$-functors, the proposition follows, using the Definition 5.2.1.

(ii) Analogically.

∎

5.4.2. Note (cata- and anamorphisms are least fixed points). Let $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol}\,\boldsymbol{\mathcal{C}})$, $A, B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$, $\varphi : A \leftarrow \mathsf{F}A$ strict and $\psi : \mathsf{F}B \leftarrow B$. The UP of the catamorphism is a fixed point equation:

$$h = (\!|\varphi|\!)_{\mathsf{F}} \iff h = (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} in_{\mathsf{F}}^{-1})h \quad \forall\, h : A \leftarrow B.$$

Thus $(\!|\varphi|\!)_{\mathsf{F}}$ is the unique fixed point of $\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} in_{\mathsf{F}}^{-1}$ and hence in particular is also the least fixed point:

$$(\!|\varphi|\!)_{\mathsf{F}} = \mathrm{fix}(\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} in_{\mathsf{F}}^{-1}) = [\![\, \varphi,\, in_{\mathsf{F}}^{-1} \,]\!]_{\mathsf{F}}.$$

For the anamorphism the dual statement is true:

$$[\!(\psi)\!]_{\mathsf{F}} = \mathrm{fix}(out_{\mathsf{F}}^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi) = [\![\, out_{\mathsf{F}}^{-1},\, \psi \,]\!]_{\mathsf{F}}.$$

5.4.3. Theorem (hylo-fusion I). Let $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol}\,\boldsymbol{\mathcal{C}})$, $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{F})$, $\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F})$ and $\alpha \in \mathrm{Mor}\,\boldsymbol{\mathcal{C}}$ be an isomorphism such that $\alpha \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F})$ (and hence $\alpha^{-1} \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{F})$). Then the following holds:

$$[\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \cdot [\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}} = [\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}}.$$

Proof. At first we show by induction:

$$\forall\, n, m \in \mathbb{N}_0 :: \ (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^n \bot \ \cdot \ (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^m \bot = (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{\min\{n,m\}} \bot. \qquad (*)$$

Induction base $n = 0 \ \vee \ m = 0$:

$$\begin{aligned} n = 0 &\implies \forall\, m \in \mathbb{N}_0 :: \ \bot \cdot (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^m \bot = \bot, \\ m = 0 &\implies \forall\, n \in \mathbb{N}_0 :: \ (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^n \bot \cdot \bot = \bot. \end{aligned}$$

Induction step, $(n,m) \mapsto (n+1, m+1)$:

$$
\begin{aligned}
& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n+1}\bot \;\cdot\; (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m+1}\bot \\
= \;& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)\big((\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot\big) \;\cdot\; (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)\big((\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot\big) \\
= \;& \quad \{ \text{ definition of } \leftarrow\!\circ \text{ in 3.2.3 } \} \\
& \varphi \cdot \mathsf{F}\big((\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot\big) \cdot \alpha \;\cdot\; \alpha^{-1} \cdot \mathsf{F}\big((\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot\big) \cdot \psi \\
= \;& \varphi \cdot \mathsf{F}\big((\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot\big) \;\cdot\; \mathsf{F}\big((\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot\big) \cdot \psi \\
= \;& \quad \{ \text{ (multiplicativity) } \} \\
& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)\big((\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot \;\cdot\; (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot\big) \\
= \;& \quad \{ \text{ induction premise } \} \\
& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)\big((\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{\min\{n,m\}}\bot\big) \\
= \;& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{1+\min\{n,m\}}\bot \\
= \;& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{\min\{n+1,m+1\}}\bot
\end{aligned}
$$

According to Definition 5.3.1 and Definition 2.1.12 the hylomorphism is the supremum

$$
[\![\varphi, \psi]\!]_{\mathsf{F}} = \mathrm{fix}(\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi) = \bigsqcup_{k \in \mathbb{N}_0} (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{k}\bot. \tag{$**$}
$$

Therewith we will show the actual proposition in two parts:

'$\sqsupseteq$': It holds $\forall\, n, m \in \mathbb{N}_0$:

$$
\begin{aligned}
& [\![\varphi, \psi]\!]_{\mathsf{F}} \\
\sqsupseteq \;& \quad \{ \; (**) \; \} \\
& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{\min\{n,m\}} \\
= \;& \quad \{ \; (*) \; \} \\
& (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot \;\cdot\; (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot
\end{aligned}
$$

thus

$$
\begin{aligned}
& [\![\varphi, \psi]\!]_{\mathsf{F}} \\
\sqsupseteq \;& \quad \{ \text{ see above } \} \\
& \bigsqcup_{n,m \in \mathbb{N}_0} \big((\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot \;\cdot\; (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot\big) \\
= \;& \quad \{ \text{ Lemma 2.1.15 } \} \\
& \bigsqcup_{n \in \mathbb{N}_0} (\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \alpha)^{n}\bot \;\cdot\; \bigsqcup_{m \in \mathbb{N}_0} (\alpha^{-1} \overset{\mathsf{F}}{\leftarrow\!\circ} \psi)^{m}\bot \\
= \;& \quad \{ \; (**) \; \} \\
& [\![\varphi, \alpha]\!]_{\mathsf{F}} \cdot [\![\alpha^{-1}, \psi]\!]_{\mathsf{F}}.
\end{aligned}
$$

'$\sqsubseteq$': On the other hand it is easy to see that $[\![\varphi, \alpha]\!]_{\mathsf{F}} \cdot [\![\alpha^{-1}, \psi]\!]_{\mathsf{F}}$ is a fixed point of $\varphi \overset{\mathsf{F}}{\leftarrow\!\circ} \psi$, because of

$$
\begin{array}{cl}
& (\varphi \overset{\mathsf{F}}{\hookleftarrow} \psi)\big([\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \cdot [\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}}\big) \\
= & \quad \{ \text{ definition of } \hookleftarrow \text{ in 3.2.3 } \} \\
& \varphi \cdot \mathsf{F}[\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \ \cdot \ \mathsf{F}[\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}} \cdot \psi \\
= & \quad \{ \text{ since } \alpha \text{ is an isomorphism } \} \\
& \varphi \cdot \mathsf{F}[\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \cdot \alpha \ \cdot \ \alpha^{-1} \cdot \mathsf{F}[\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}} \cdot \psi \\
= & \quad \{ \text{ definition of } \hookleftarrow \text{ in 3.2.3 } \} \\
& (\varphi \overset{\mathsf{F}}{\hookleftarrow} \alpha)[\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \ \cdot \ (\alpha^{-1} \overset{\mathsf{F}}{\hookleftarrow} \psi)[\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}} \\
= & \quad \{ \text{ definition of } [\![\, \centerdot,\, \centerdot \,]\!] \text{ in 5.3.1 } \} \\
& [\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \cdot [\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}}.
\end{array}
$$

Since $[\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}}$ however is the least fixed point, we get this time

$$[\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}} \sqsubseteq [\![\, \varphi,\, \alpha \,]\!]_{\mathsf{F}} \cdot [\![\, \alpha^{-1},\, \psi \,]\!]_{\mathsf{F}}.$$

■

5.4.4. Corollary. Let $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$. The initial F-algebra in_{F} is isomorphic to the inverse out_{F}^{-1} of the final F-co-algebra out_{F}. More precisely, the hylomorphism

$$[\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} : \mu\mathsf{F} \leftarrow \nu\mathsf{F}$$

is an isomorphism with

$$[\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \in \boldsymbol{\mathcal{A}lg}\,\mathsf{F}(in_{\mathsf{F}}, out_{\mathsf{F}}^{-1}) \cap \in \boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F}(in_{\mathsf{F}}^{-1}, out_{\mathsf{F}}).$$

Proof.

$$
\begin{array}{cl}
& [\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \cdot [\![\, out_{\mathsf{F}}^{-1},\, in_{\mathsf{F}}^{-1} \,]\!]_{\mathsf{F}} \\
= & \quad \{ \text{ Theorem 5.4.3 } \} \\
& [\![\, in_{\mathsf{F}},\, in_{\mathsf{F}}^{-1} \,]\!]_{\mathsf{F}} \\
= & \quad \{ \text{ Note 5.4.2 } \} \\
& (\![in_{\mathsf{F}}]\!)_{\mathsf{F}} \\
= & \quad \{ \text{ Definition 2.2.33 and reflection } \} \\
& id_{\mu\mathsf{F}}.
\end{array}
$$

And dually:

$$[\![\, out_{\mathsf{F}}^{-1},\, in_{\mathsf{F}}^{-1} \,]\!]_{\mathsf{F}} \cdot [\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} = id_{\nu\mathsf{F}}.$$

Thus $[\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} : \mu\mathsf{F} \leftarrow \nu\mathsf{F}$ is an isomorphism.

$$
\begin{array}{cl}
\Longrightarrow & \quad \{ \text{ Definition 5.3.1 } \} \\
& [\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} = \mathrm{fix}(in_{\mathsf{F}} \overset{\mathsf{F}}{\hookleftarrow} out_{\mathsf{F}}) \\
\Longrightarrow & \quad \{ \text{ Definitions 2.1.12 \& 3.2.3 } \} \\
& [\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} = in_{\mathsf{F}} \cdot \mathsf{F}[\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \cdot out_{\mathsf{F}} \\
\Longrightarrow & \quad \{ \text{ Lemma 2.2.36 } \} \\
& [\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \cdot out_{\mathsf{F}}^{-1} = in_{\mathsf{F}} \cdot \mathsf{F}[\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \\
\Longrightarrow & \quad \{ \text{ Definition 2.2.33 } \} \\
& [\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \in \boldsymbol{\mathcal{A}lg}\,\mathsf{F}(in_{\mathsf{F}}, out_{\mathsf{F}}^{-1}).
\end{array}
$$

And dually: $[\![\, in_{\mathsf{F}},\, out_{\mathsf{F}} \,]\!]_{\mathsf{F}} \in \boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F}(in_{\mathsf{F}}^{-1}, out_{\mathsf{F}})$. ■

5.4.5. Corollary. Under the preconditions from Corollary 5.4.4 we may, without loss of generality, assume that $\llbracket\, in_{\mathsf{F}},\, out_{\mathsf{F}}\,\rrbracket_{\mathsf{F}} = id_{\mu\mathsf{F}}$ and thus $in_{\mathsf{F}} = out_{\mathsf{F}}^{-1}$, since in_{F} and out_{F} are only determined up to F-algebra-isomorphism anyway.

5.4.6. Theorem. Let $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ and $\mathsf{H} := \mathsf{F} \cdot \grave{\mathsf{P}} : \boldsymbol{\mathcal{C}}_{\perp} \leftarrow \breve{\boldsymbol{\mathcal{C}}}_{\perp}$ where $\breve{\boldsymbol{\mathcal{C}}}_{\perp} := (\boldsymbol{\mathcal{C}}_{\perp})\breve{}$ denotes the involutory closure of $\boldsymbol{\mathcal{C}}_{\perp}$ as defined in Definition 3.3.3. If $\boldsymbol{\mathcal{A}lg}\,\mathsf{F} = \boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}_{\perp}}\,\mathsf{F}$ has an initial object, then $\boldsymbol{\mathcal{A}lg}_{\breve{\boldsymbol{\mathcal{C}}}_{\perp}}\,\breve{\mathsf{H}}$ has an initial object, such that

$$\grave{\mathsf{P}}\, in_{\breve{\mathsf{H}}} = in_{\mathsf{F}} \quad \text{and} \quad \grave{\mathsf{P}}\, in_{\breve{\mathsf{H}}}^{*} = out_{\mathsf{F}}.$$

Proof. See [FP94]. ■

5.4.7. Definition and Lemma (type-functor). Under the preconditions and assumptions from Corollary 5.4.5 we define:

$$\mathsf{TF} := \mu\mathsf{F} = \nu\mathsf{F} \quad \forall\, \mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}}).$$

We may complete T to a functor $\mathsf{T} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{Pol\,\mathcal{C}}$ by:

$$\mathsf{T}\tau := (\!|in_{\mathsf{F}} \cdot \tau_{\mathsf{TF}}|\!)_{\mathsf{G}} = \llbracket(\tau_{\mathsf{TG}} \cdot out_{\mathsf{G}})\rrbracket_{\mathsf{F}} \quad \forall\, \mathsf{F}, \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}}) \quad \forall\, \tau : \mathsf{F} \dot{\leftarrow} \mathsf{G}.$$

The functor T is called the **type-functor**. In Section 6 we will view the objects of a category as types of a typed language. TF is the recursive type defined by F. T is an analogue at the type level to the fixed point operator fix at the program level.

Proof. Immediately by Theorem 4.1.1. ■

5.4.8. Corollary (cata · ana = hylo). For $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$, $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{F})$ and $\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F})$ holds:

$$(\!|\varphi|\!)_{\mathsf{F}} \cdot \llbracket(\psi)\rrbracket_{\mathsf{F}} = \llbracket\, \varphi,\, \psi\,\rrbracket_{\mathsf{F}}.$$

This allows us to visualize the connection of cata-, ana-, and hylomorphism, constructor and destructor as well as the type-functor in one commuting diagram:

$$\begin{tikzcd}
B \arrow[rrr, "\psi"] \arrow[dd, "\llbracket\, \varphi,\, \psi\,\rrbracket_{\mathsf{F}}"'] \arrow[dr, "\llbracket(\psi)\rrbracket_{\mathsf{F}}"] & & & \mathsf{F}B \arrow[dd, "\mathsf{F}\llbracket\, \varphi,\, \psi\,\rrbracket_{\mathsf{F}}"] \arrow[dl, "\mathsf{F}\llbracket(\psi)\rrbracket_{\mathsf{F}}"'] \\
& \mathsf{TF} \arrow[r, shift left, "out_{\mathsf{F}}"] \arrow[dl, "(\!|\varphi|\!)_{\mathsf{F}}"] & \mathsf{F}(\mathsf{TF}) \arrow[l, shift left, "in_{\mathsf{F}}"] \arrow[dr, "\mathsf{F}(\!|\varphi|\!)_{\mathsf{F}}"'] & \\
A & & & \mathsf{F}A \arrow[lll, "\varphi"]
\end{tikzcd}$$

5.4.9. Lemma.

$$\mathsf{T}\,\mathsf{Id} = 1 \text{ and } \mathsf{T}\,\mathsf{K}_{A} = A \quad \forall\, A \in \boldsymbol{\mathcal{C}}$$

Without proof.

5.4.10. Theorem (hylo-fusion II). Let $\mathsf{F}, \mathsf{G}, \mathsf{H} \in \mathrm{Ob}(\boldsymbol{Pol}\,\boldsymbol{\mathcal{C}})$, $\sigma : \mathsf{F} \dot{\leftarrow} \mathsf{G}$, $\tau : \mathsf{G} \dot{\leftarrow} \mathsf{H}$, $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{F})$, $\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F})$, $\varphi' \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{H})$ and $\psi' \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{H})$. Then the following holds:

$$\frac{\mathrm{dom}\,\psi = \mathrm{cod}\,\varphi' =: B \qquad \psi \cdot \varphi' = \sigma_B \cdot \tau_B}{[\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}} \cdot [\![\, \varphi',\, \psi' \,]\!]_{\mathsf{H}} = [\![\, \varphi \cdot \sigma,\, \tau \cdot \psi' \,]\!]_{\mathsf{G}}}.$$

Proof. We infer:

$$\begin{array}{ll}
 & \psi \cdot \varphi' = \sigma_B \cdot \tau_B \\
\Longrightarrow & \quad \{\ \text{UP} \ \&\ \sigma \cdot \tau : \mathsf{F} \dot{\leftarrow} \mathsf{H}\ \} \\
 & [\![\psi]\!]_{\mathsf{F}} \cdot \varphi' = in_{\mathsf{F}} \cdot \mathsf{F}[\![\psi]\!]_{\mathsf{F}} \cdot \psi \cdot \varphi' = in_{\mathsf{F}} \cdot \mathsf{F}[\![\psi]\!]_{\mathsf{F}} \cdot \sigma \cdot \tau = (in_{\mathsf{F}} \cdot \sigma \cdot \tau) \cdot \mathsf{H}[\![\psi]\!]_{\mathsf{F}} \\
\Longrightarrow & \quad \{\ \text{fusion}\ \} \\
 & [\![\psi]\!]_{\mathsf{F}} \cdot (\!|\varphi'|\!)_{\mathsf{H}} = (\!|in_{\mathsf{F}} \cdot \sigma \cdot \tau|\!)_{\mathsf{H}} \\
\Longrightarrow & \quad \{\ \text{Corollary 5.4.8 \& Theorem 5.4.3}\ \} \\
 & [\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}} \cdot [\![\, \varphi',\, \psi' \,]\!]_{\mathsf{H}} = (\!|\varphi|\!)_{\mathsf{F}} \cdot [\![\psi]\!]_{\mathsf{F}} \cdot (\!|\varphi'|\!)_{\mathsf{H}} \cdot [\![\psi']\!]_{\mathsf{H}} = [\![\, \varphi \cdot \sigma,\, \tau \cdot \psi' \,]\!]_{\mathsf{G}} \quad \blacksquare
\end{array}$$

5.4.11. Theorem (acid rain theorem (hylo-version) [TM95]). For $\mathsf{F}, \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol}\,\boldsymbol{\mathcal{C}})$, $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{G})$ and $\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F})$ we have:

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}\,\mathsf{F}, |\,\centerdot\,|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}\,\mathsf{G}, |\,\centerdot\,|_{\mathsf{G}})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot [\![\, \mathsf{H}in_{\mathsf{G}},\, \psi \,]\!]_{\mathsf{F}} = [\![\, \mathsf{H}\varphi,\, \psi \,]\!]_{\mathsf{F}}}$$

and

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{G}, |\,\centerdot\,|_{\mathsf{G}}) \leftarrow (\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F}, |\,\centerdot\,|_{\mathsf{F}})}{[\![\, \varphi,\, \mathsf{H}out_{\mathsf{F}} \,]\!]_{\mathsf{G}} \cdot [\![\psi]\!]_{\mathsf{F}} = [\![\, \varphi,\, \mathsf{H}\psi \,]\!]_{\mathsf{G}}}$$

Proof. The first proposition follows from Corollary 4.3.2 and 5.4.8. The second one follows from the dual statement to Corollary 4.3.2, which refers to the anamorphism, and Corollary 5.4.8 as well. ∎

6 Categorical semantics

This section connects the category theory with the theory of functional programming languages. We will define a denotational semantics for a functional language, which maps programs onto morphisms and types onto objects. As we will see, a functional language itself may be viewed as a category, such that the mentioned semantics even is a functor. This functor allows us to apply Theorem 5.4.11 to our functional language.

6.1 Extended typed λ-calculus

We want to apply our results to the functional programming language Haskell [PH99], which is based on Core Haskell, which itself is an extended typed λ-calculus.

We will define the syntax and operational semantics of an extended typed λ-calculus, which is based on [Gun92] 7.4 and is quite similar to Core Haskell.

6.1.1. Definition (syntax of the extended typed λ-calculus). Let *Var* and *Tvar* be two countably infinite disjoint sets and ε the empty word. We give the syntax of our extended typed λ-calculus in EBNF by

$$
\begin{array}{lrcl}
\textbf{type-variables} & A & ::= & \mathit{Tvar} \\
\textbf{variables} & V & ::= & \mathit{Var} \\
\textbf{type-expressions} & T & ::= & 1 \;\big|\; A \;\big|\; (T \times T) \;\big|\; (T + T) \;\big|\; T^T \;\big|\; (\boldsymbol{\mu} A.\, T) \\
\textbf{type-schemes} & \mathit{Ts} & ::= & T \;\big|\; (\forall A.\, \mathit{Ts}) \\
\textbf{type-environments} & \mathit{Ct} & ::= & \varepsilon \;\big|\; \mathit{Ct}\ A \\
\textbf{environments} & \mathit{Cx} & ::= & \varepsilon \;\big|\; \mathit{Cx}\,(V : T) \\
\textbf{terms} & X & ::= & () \;\big|\; V \;\big|\; (\lambda V \colon T.\, X) \;\big|\; (X X) \;\big|\; (\Lambda A.\, X) \;\big|\; X_T \;\big|\; (\mu V \colon T.\, X) \\
 & & & \;\big|\; (X, X) \;\big|\; \mathtt{fst}\, X \;\big|\; \mathtt{snd}\, X \\
 & & & \;\big|\; \mathtt{inl}_{T+T}\, X \;\big|\; \mathtt{inr}_{T+T}\, X \;\big|\; [X, X] X \\
 & & & \;\big|\; \mathtt{intro}_{(\forall A.\, T)}\, X \;\big|\; \mathtt{elim}_{(\forall A.\, T)}\, X
\end{array}
$$

6.1.2. Note (simplified notation). In order to avoid parentheses in terms, we declare the following rules, where P, Q and R are terms and $x \in \mathit{Var}$:

- We view the application, *i.e.* the concatenation of terms as a left-associative operation, *i.e.* we write just PQR instead of $(PQ)R$.
- Partial terms directly following a point need not to be enclosed in parentheses, and thus we write *e.g.* just $(\lambda x \colon \tau.\, PQ)$ instead of $(\lambda x \colon \tau.\, (PQ))$.
- We omit the outer parentheses of a term, *except* the parentheses for pairing (P, Q) or the constant $()$.

6.1.3. Definition (type-substitution). For every type-scheme ϱ and every type-variable $b \in \mathit{Var}$ we define the substitution operator $[\varrho/b]$ on type schemes for every $a \in \mathit{Tvar}$ and every type-schemes σ, τ as follows:

$$
\begin{array}{lll}
[\varrho/b]\ 1 & := & 1 \\
[\varrho/b]\ a & := & \begin{cases} \varrho & : a = b, \\ a & : a \neq b \end{cases} \\
[\varrho/b]\ (\sigma \times \tau) & := & [\varrho/b]\,\sigma \times [\varrho/b]\,\tau \\
[\varrho/b]\ (\sigma + \tau) & := & [\varrho/b]\,\sigma + [\varrho/b]\,\tau \\
[\varrho/b]\ (\sigma^\tau) & := & \big([\varrho/b]\,\sigma\big)^{[\varrho/b]\,\tau} \\
[\varrho/b]\ (\forall a.\, \tau) & := & \begin{cases} (\forall a.\, \tau) & : a = b, \\ (\forall a.\ [\varrho/b]\,\tau) & : a \neq b \end{cases} \\
[\varrho/b]\ (\boldsymbol{\mu} a.\, \tau) & := & \begin{cases} \boldsymbol{\mu} a.\, \tau & : a = b, \\ \boldsymbol{\mu} a.\ [\varrho/b]\,\tau & : a \neq b \end{cases}
\end{array}
$$

In the style of the application of terms (as we will see later in Definition 6.2.3) in addition we define the functional notation

$$(\forall a.\, \tau)\sigma := [\sigma/a]\,\tau \quad \forall\, a \in \mathit{Tvar} \quad \forall \text{ type-schemes } \sigma, \tau.$$

This is just another notation for the previously defined substitution. Such an application of type-schemes on type-schemes is *not* part of our language. Anyway, it will turn out later that type-schemes of the form $(\forall a.\, \tau)$ will be mapped onto functors by the categorical semantics, which we will define in Definition 6.3.1. The matching definition of an application of $(\forall a.\, \tau)$ on terms will be given in Definition 6.4.1. Apart from this, the notation $(\forall a.\, \tau)\sigma$ is much more intuitive in connection with `intro` and `elim` than $[\sigma/a]\,\tau$.

6.1.4. Definition (some syntactic sugar). Our language is very simple and abstract, to simplify the following semantics and proofs. Unfortunately, this implies that even small programs are unreadable for humans. Therefore we introduce some notations in the style of Haskell, to abbreviate some specific terms. Let $x, y, f \in \mathit{Var}$, $a \in \mathit{Tvar}$ and P, Q, R be terms and σ, τ, ϱ be type-schemes.

$$\begin{array}{lcl}
\mathtt{let}\; x = P \;\mathtt{in}\; Q & := & (\lambda x\colon \tau.\, Q)P \\
\mathtt{letrec}\; x = P \;\mathtt{in}\; Q & := & (\lambda x\colon \tau.\, Q)(\mu x\colon \tau.\, P) \\
\mathtt{case}\; R \;\mathtt{of}\; \mathtt{inl}_{\sigma+\tau} x \;\rightarrow\; P \;\mathtt{or}\; \mathtt{inr}_{\sigma+\tau} y \;\rightarrow\; Q & := & [\lambda x\colon \sigma.\, P, \lambda y\colon \tau.\, Q]R \\
\mathtt{if}\; R \;\mathtt{then}\; P \;\mathtt{else}\; Q & := & \mathtt{case}\; R \;\mathtt{of}\; \mathtt{inl}_{1+1} x \;\rightarrow\; P \;\mathtt{or}\; \mathtt{inr}_{1+1} y \;\rightarrow\; Q \\
P\ \grave{}Q\grave{}\ R & := & (QP)R \\
\mathtt{id}_\tau & := & \lambda x\colon \tau.\, x
\end{array}$$

In order to get `case`-terms with more than two cases, one simply has to nest several `case`-terms. How to construct a Haskell-like `case` with connections to Haskell constructors names, will be demonstrated in Note 6.1.8.

6.1.5. Definition (well-typed term). We will introduce a 4-ary relation $\centerdot, \centerdot \vdash \centerdot : \centerdot$ on type-environments, environments, terms and type-schemes such that intuitively:

$$\Theta, \Gamma \vdash P : \tau$$
$$\Longleftrightarrow$$

'P is a **well-typed** term of type τ in the environment Γ and type-environment Θ.'

and a binary relation $\centerdot \vdash \centerdot$ such that intuitively:

$$\Theta \vdash \Gamma \quad\Longleftrightarrow\quad \text{‘}\Gamma \text{ is an environment in the type-environment } \Theta.\text{’}$$

or

$$\Theta \vdash \tau \quad\Longleftrightarrow\quad \text{‘}\tau \text{ is a type-scheme in the type-environment } \Theta.\text{’},$$

respectively. But first we define the function dom to *Tvar* from the set of type-environments recursively by

$$\begin{array}{rcl}
\operatorname{dom} \varepsilon & := & \emptyset \\
\operatorname{dom}\big(\Theta a\big) & := & \operatorname{dom} \Theta \cup \{a\} \\
& & \forall \text{type-environments } \Theta \quad \forall\, a \in \mathit{Tvar}
\end{array}$$

and analogically the function dom to *Var* from environments recursively by

$$\begin{array}{rcl}
\operatorname{dom} \varepsilon & := & \emptyset \\
\operatorname{dom}\big(\Gamma(x : \tau)\big) & := & \operatorname{dom} \Gamma \cup \{x\} \\
& & \forall \text{environments } \Gamma \quad \forall\, x \in \mathit{Var} \quad \forall \text{type-schemes } \tau.
\end{array}$$

Thus $\operatorname{dom}\Gamma$ is precisely the set of all those variables, to which a type-scheme is assigned by Γ, and $\operatorname{dom}\Theta$ is the set of all variables occuring in Θ.

Now we define the relation $\mathbf{.}, \mathbf{.} \vdash \mathbf{.} : \mathbf{.}$ by the inference rules in Table 1 together with the auxiliary technical rule

$$\frac{\Theta, \Gamma \vdash P : \tau}{\Theta \vdash \Gamma \qquad \Theta \vdash \tau}.$$

6.1.6. Definition (open and closed terms and type-schemes). A type-scheme τ is called **closed**, if it is a type-scheme in the trivial type-environment, *i.e.* $\varepsilon \vdash \tau$, and is called **open** otherwise. A term P is called **closed** if it is a well-typed term in the trivial environment and the trivial type-environment, *i.e.* $\varepsilon, \varepsilon \vdash P$, else it is called **open**.

6.1.7. Definition (type, program, value). A closed type-expression (*not* type-scheme) is called a **type**. A closed well-typed term is called a **program**. A program with the syntax given in EBNF (see 6.1.1) by

$$\begin{aligned} \mathit{Vl} ::= {} & () \mid V \mid (\lambda V : T.\, X) \mid (\Lambda A.\, T) \\ & \mid (X, X) \\ & \mid \mathtt{inl}_{T+T}\, X \mid \mathtt{inr}_{T+T}\, X \\ & \mid \mathtt{intro}_T\, X \end{aligned}$$

is called a **value**. We define the following sets:

$$\begin{array}{lcl} \mathit{Type}^{\Theta} & := & \{\tau \mid \Theta \vdash \tau \text{ and } \tau \text{ is a type-expression}\} \\ \mathit{Type} & := & \mathit{Type}^{\varepsilon} \\ \mathit{Prog}^{\Theta,\Gamma}\,\tau & := & \{P \mid \Theta, \Gamma \vdash P : \tau\} \\ \mathit{Prog}\,\tau & := & \mathit{Prog}^{\varepsilon,\varepsilon}\,\tau \\ \mathit{Prog} & := & \bigcup_{\tau \in \mathit{Type}} \mathit{Prog}\,\tau \\ \mathit{Val}^{\Theta,\Gamma}\,\tau & := & \{C \mid C \in \mathit{Prog}^{\Theta,\Gamma}\,\tau \text{ and } C \text{ has the above syntax } \mathit{Vl}\} \\ \mathit{Val}\,\tau & := & \mathit{Val}^{\varepsilon,\varepsilon}\,\tau \\ \mathit{Val} & := & \bigcup_{\tau \in \mathit{Type}} \mathit{Val}\,\tau \end{array}$$

Thus *Type* is the set of all types, *Prog* the set of all programs and *Val* the set of all values.

6.1.8. Note (constructors). In Definition 6.1.4 we saw how to translate the most important Haskell-constructs into our language. However, there is a difference in the treatment of constructors (and destructors) that we have to investigate a little further.

In our language there is only one constructor `intro` used for every recursive data type in contrast to Haskell, where each component of each recursive data type has its own constructor with its own name. But it is possible to express the Haskell-constructors using `intro` and the injections `inl` and `inr`.

We illustrate this by the following example Haskell-program:

```
data Nat = Zero | Succ Nat

dup          :: Nat -> Nat
dup Zero     =  Zero
dup (Succ n) =  Succ (Succ (dup n))
```

First we merge the case analysis into a single `case`-expression and then we abstract the whole expression from its parameter:

type inference rules (definition of $\_,\_\vdash\_:\_$)	
$\dfrac{}{\varepsilon,\varepsilon \vdash () : 1}$ $\qquad$ $\dfrac{x \in \mathit{Var} \qquad \Theta \vdash \tau}{\Theta, (x:\tau) \vdash x : \tau}$	constant & variable
$\dfrac{a \in \mathit{Tvar}}{a \vdash a}$ $\qquad$ $\dfrac{\Theta a \vdash \tau}{\Theta \vdash \boldsymbol{\mu} a.\,\tau}$	type-variable & type fixed point
$\dfrac{\Theta,\Gamma \vdash P:\tau \qquad a \in \mathit{Tvar} \setminus \operatorname{dom}\Theta}{\Theta a, \Gamma \vdash P : \tau}$	extension of the type-environment
$\dfrac{\Theta \vdash \tau \qquad \Theta,\Gamma \vdash P:\sigma \qquad x \in \mathit{Var} \setminus \operatorname{dom}\Gamma}{\Theta, \Gamma(x:\tau) \vdash P : \sigma}$	extension of the environment
$\dfrac{\Theta,\Gamma(x:\tau) \vdash P:\sigma}{\Theta,\Gamma \vdash (\lambda x{:}\,\tau.\,P) : \sigma^\tau}$	abstraction
$\dfrac{\Theta,\Gamma \vdash P:\sigma^\tau \qquad \Theta,\Gamma \vdash Q:\tau}{\Theta,\Gamma \vdash PQ : \sigma}$	application
$\dfrac{\Theta,\Gamma \vdash P:\sigma \qquad \Theta,\Gamma \vdash Q:\tau}{\Theta,\Gamma \vdash (P,Q) : \sigma \times \tau}$	pairing
$\dfrac{\Theta,\Gamma \vdash P:\sigma\times\tau}{\Theta,\Gamma \vdash \mathtt{fst}\,P : \sigma \qquad \Theta,\Gamma \vdash \mathtt{snd}\,P : \tau}$	projections
$\dfrac{\Theta,\Gamma \vdash P:\sigma}{\Theta,\Gamma \vdash \mathtt{inl}_{\sigma+\tau}\,P : \sigma+\tau}$ $\quad$ $\dfrac{\Theta,\Gamma \vdash Q:\tau}{\Theta,\Gamma \vdash \mathtt{inr}_{\sigma+\tau}\,Q : \sigma+\tau}$	injections
$\dfrac{\Theta,\Gamma \vdash R:\sigma+\tau \quad \Theta,\Gamma(x:\sigma) \vdash P:\varrho \quad \Theta,\Gamma(y:\tau) \vdash Q:\varrho}{\Theta,\Gamma \vdash \mathtt{case}\ R\ \mathtt{of}\ \mathtt{inl}_{\sigma+\tau}x \ \to\ P\ \mathtt{or}\ \mathtt{inr}_{\sigma+\tau}y \ \to\ Q : \varrho}$	case analysis
$\dfrac{\Theta,\Gamma(x:\tau) \vdash P:\tau}{\Theta,\Gamma \vdash (\mu x{:}\,\tau.\,P) : \tau}$	fixed point
$\dfrac{\Theta \vdash \Gamma \qquad \Theta a,\Gamma \vdash P:\tau}{\Theta,\Gamma \vdash \Lambda a.\,P : (\forall a.\,\tau)}$	type-abstraction
$\dfrac{\Theta,\Gamma \vdash P:(\forall a.\,\tau) \qquad \Theta \vdash \sigma}{\Theta,\Gamma \vdash P_\sigma : (\forall a.\,\tau)\sigma}$	type-application
$\dfrac{\Theta,\Gamma \vdash P:\big(\forall a.\,\tau\big)\big(\boldsymbol{\mu} a.\,\tau\big)}{\Theta,\Gamma \vdash \mathtt{intro}_{(\forall a.\,\tau)}\,P : \boldsymbol{\mu} a.\,\tau}$	constructor
$\dfrac{\Theta,\Gamma \vdash P:\boldsymbol{\mu} a.\,\tau}{\Theta,\Gamma \vdash \mathtt{elim}_{(\forall a.\,\tau)}\,P : \big(\forall a.\,\tau\big)\big(\boldsymbol{\mu} a.\,\tau\big)}$	destructor

Table 1: Type inference rules.

```
dup = \n ->  case n of Zero   -> Zero
                       Succ y -> Succ (Succ (dup y))
```

This is a recursive definition, which we want to make more explicit using the fixed point operator `fix` where `fix f = f (fix f)`:

```
dup = fix (\f -> \n -> case n of Zero   -> Zero
                                 Succ y -> Succ (Succ (f y))
          )
```

Now we can translate the program in our language: We start with the recursive data type `Nat`. By abstraction from the parameter `Nat` in the definition `data Nat = Zero | Succ Nat` and ignoring the constructors names we obtain a type-scheme $(\forall a.\, 1 + a)$. The formal fixed point thereof is the recursive data type:

$$Nat := \boldsymbol{\mu} a.\, 1 + a$$

Now we translate the definition of `dup` into a program *dup* of our language. In order to do so, we substitute the constructor `Zero` by

$$Zero := \lambda x\colon 1.\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inl}_{1+Nat}\, x)$$

and `Succ` by

$$Succ := \lambda y\colon Nat.\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}\, y)$$

and we get:

$$\begin{aligned} dup \ = \ & \mu f\colon Nat^{Nat}.\ \lambda n\colon Nat.\ \mathtt{case\ elim}_{(\forall a.\, 1+a)}\, n\ \mathtt{of\ inl}_{1+Nat} x \ \rightarrow\ Zero\, x \\ & \quad \mathtt{or\ inr}_{1+Nat} y \ \rightarrow\ Succ(Succ(fy)) \end{aligned}$$

and finally:

$$\begin{aligned} dup \ = \ & \mu f\colon Nat^{Nat}.\ \lambda n\colon Nat.\ \mathtt{case\ elim}_{(\forall a.\, 1+a)}\, n \\ & \quad \mathtt{of\ inl}_{1+Nat} x \ \rightarrow\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inl}_{1+Nat}\, x) \\ & \quad \mathtt{or\ inr}_{1+Nat} y \ \rightarrow\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}(\mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}(fy)))). \end{aligned}$$

It is worth mentioning that the Haskell-destructors are implicitly contained in the `case`-expressions, so that we have to translate a '`case`' from Haskell into a '`case elim`' of our language. Finally we substitute the `case`-expression according to Definition 6.1.4 by $[\centerdot, \centerdot]\centerdot$:

$$\begin{aligned} dup \ = \ & \mu f\colon Nat^{Nat}.\ \lambda n\colon Nat. \\ & \quad [\lambda x\colon 1.\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inl}_{1+Nat}\, x), \\ & \quad\ \ \lambda y\colon Nat.\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}(\mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}(fy)))) \\ & \quad](\mathtt{elim}_{(\forall a.\, 1+a)}\, n). \end{aligned}$$

The reasons for the different handling of constructors (and destructors) are the following:

- Haskell style:
  - The definition of names for the constructors components within the definition of a recursive data type increases the programs transparency and readability.
  - The destructor normally yields a sum-type, which is the input for a `case`-construct, hence it is obviously a good idea to combine destructor and `case` to a single program construct. In particular in the context of our above example, we find that

    ```
    \n -> case n of Zero   -> Zero
                    Succ y -> Succ y
    ```

is the product of constructor and destructor and hence the identical function on `Nat`.

- Our style:
  - Our language has explicit constructors and destructors `intro` and `elim`, because we want to make the wrapping and unwraping of type fixed points explicit.
  - The categorical semantics, which we will define in Definition 6.3.1, maps `intro` and `elim` on an initial F-algebra or a final F-co-algebra, respectively, in a natural way.
  - In order to apply the hylomorphism-fusion on our language, we will have to use a program transformation (*cf.* Corollary 6.4.12), which is a formal substitution of the constructor `intro`.

For analogous reasons one does not use an explicit fixed point operator (`fix`) in Haskell, but we do in our language (μ).

6.2 Operational semantics

6.2.1. Definition (substitution). For every well-typed term S and every variable $z \in \mathit{Var}$ we define the **substitution operator** $[S/z]$ by recursion on the inference rules from Definition 6.1.5:

$$
\begin{array}{llll}
[S/z]\ () & := () & & \text{constant} \\[1em]
[S/z]\ x & := \begin{cases} S & : z = x, \\ x & : z \neq x \end{cases} & \forall\, x \in \mathit{Var} & \text{variable} \\[1em]
[S/z]\ (\xi x\colon \tau.\, P) & := \begin{cases} \xi x\colon \tau.\, P & : z = x, \\ \xi x\colon \tau.\, ([S/z]\, P) & : z \neq x \end{cases} & \forall\, \xi \in \{\lambda, \mu\} & \begin{array}{l}\text{abstraction} \\ \text{and fixed point}\end{array}
\end{array}
$$

The remaining inference rules have the general form

$$\frac{\Theta_i, \Gamma_i \vdash P_i : \tau_i \quad \forall\, i \in \{1 \ldots n\}}{\Theta, \Gamma \vdash f(P_1, \ldots, P_n) : \tau},$$

with $n \in \{1, 2\}$ and we define for all these rules:

$$[S/z]\big(f(P_1, \ldots, P_n)\big) \ := \ f([S/z]\, P_1, \ldots, [S/z]\, P_n)$$

6.2.2. Lemma (substitution lemma). The substitution operator maps well-typed terms on well-typed terms with the same type, *i.e.*

$$\frac{\Theta, \Gamma \vdash S : \sigma \qquad \Theta, \Gamma(z : \sigma) \vdash P : \tau}{\Theta, \Gamma \vdash [S/z]\, P : \tau}$$

or written in a slightly different way:

$$\frac{\Theta, \Gamma \vdash S : \sigma \qquad z \in \mathit{Var}}{[S/z] : \mathit{Prog}^{\Theta,\Gamma}\, \tau \xleftarrow[\mathcal{S}et]{} \mathit{Prog}^{\Theta,\Gamma(z:\sigma)}\, \tau}.$$

Proof. Structural induction on the recursive definition of $[\,.\,/\,.\,]$ using the type inference rules from Definition 6.1.5. ∎

6.2.3. Definition (operational semantics). We introduce a lazy[8] big-step[9] operational semantics for our λ-calculus. We define the binary **reduction relation** $\Downarrow \subseteq \mathit{Prog} \times \mathit{Val}$ such that intuitively

$$P \Downarrow C \iff \text{`The program } P \text{ calculates the value } C\text{.'}$$

by the inference rules in Table 2. The reduction relation $\Downarrow$ is obviously deterministic, *i.e.* a partial function $\mathit{Prog} \dashrightarrow \mathit{Val}$.

6.2.4. Definition (termination). A program P **terminates** w.r.t. the operational semantics, we write $P\Downarrow$, if it reduces to a value, *i.e.*:

$$P\Downarrow :\iff \exists\, C :: \; P \Downarrow C.$$

6.2.5. Definition and Lemma (α-congruence). The **α-congruence** describes the consistent renaming of 'local variables' (These are those, which are not bound by λ, μ, Λ, $\boldsymbol{\mu}$ or $\forall$.) in terms and type-schemes, without changing the semantics. At first, we define the binary relation $\equiv_\alpha$ on terms, where we use the preconditions and environments (also for the same symbols with a prime$'$) as in Definition 6.1.5, by

$$\frac{}{() \equiv_\alpha ()} \qquad \frac{x \in \mathit{Var}}{x \equiv_\alpha x} \quad \text{constant and variable}$$

$$\frac{P \equiv_\alpha P' \qquad z \in \mathit{Var} \setminus \operatorname{dom}\Gamma}{\lambda x\colon \tau.\; P \equiv_\alpha \lambda z\colon \tau.\; [z/x]\, P'} \quad \text{abstraction}$$

$$\frac{P \equiv_\alpha P' \qquad z \in \mathit{Var} \setminus \operatorname{dom}\Gamma}{\mu x\colon \tau.\; P \equiv_\alpha \mu z\colon \tau.\; [z/x]\, P'} \quad \text{fixed point}$$

and according to

$$\frac{P_i \equiv_\alpha P_i' \quad \forall\, i}{f(P_1, \ldots, P_n) \equiv_\alpha f(P_1', \ldots, P_n')}$$

for all remaining rules. Likewise, we define also a binary relation $\equiv_\alpha$ on type-schemes in the obvious way.

The relation $\equiv_\alpha$ is an equivalence relation and leaves the type of a term invariant.

Proof. Structural induction on the recursive definition of $\equiv_\alpha$ using Definition 6.1.5. ■

6.2.6. Definition (operational semantic equivalence). Two programs P and P' with the same (up to α-congruence) type are called **(operationally) semantically equivalent**, if they have the same (up to α-congruence) semantics, we write $P \equiv^{op} P'$. This means in more detail that P and P' both do terminate and reduce to the same (mod $\equiv_\alpha$) value:

$$P \equiv^{op} P' \;:\iff\; \exists\, C, C' \in \mathit{Val} :: \; \big(P \Downarrow C \;\wedge\; P' \Downarrow C' \;\wedge\; C \equiv_\alpha C'\big).$$

The semantic equivalence obviously is an equivalence relation, because $\Downarrow$ is a partial function and $\equiv_\alpha$ is an equivalence relation. The $\equiv^{op}$-equivalence-classes are the $\Downarrow$-inverse-images of the $\equiv_\alpha$-equivalence-classes. Notice that $P \equiv^{op} P'$ implies $P\Downarrow$ as well as $P'\Downarrow$.

[8] *I.e.* partial terms are evaluated be need

[9] *I.e.* a term is directly mapped onto a value by a deterministic reduction-rule, which is in fact a recursively defined partial function.

operational semantics (definition of $\Downarrow$)	
$\dfrac{}{() \Downarrow ()}$	constant
$\dfrac{}{\lambda x\colon \tau.\, P \Downarrow \lambda x\colon \tau.\, P}$	abstraction
$\dfrac{P \Downarrow \lambda x\colon \tau.\, R \qquad [Q/x]\, R \Downarrow C}{PQ \Downarrow C}$	application (β-reduction)
$\dfrac{}{(P,Q) \Downarrow (P,Q)}$	pairing
$\dfrac{P \Downarrow (Q,R) \qquad Q \Downarrow C}{\mathtt{fst}\, P \Downarrow C}$	projection to the left
$\dfrac{P \Downarrow (Q,R) \qquad R \Downarrow C}{\mathtt{snd}\, P \Downarrow C}$	projection to the right
$\dfrac{}{\mathtt{inl}_{\sigma+\tau}\, P \Downarrow \mathtt{inl}_{\sigma+\tau}\, P}$	injection to the left
$\dfrac{}{\mathtt{inr}_{\sigma+\tau}\, Q \Downarrow \mathtt{inr}_{\sigma+\tau}\, Q}$	injection to the right
$\dfrac{R \Downarrow \mathtt{inl}_{\sigma+\tau}\, S \qquad [S/x]\, P \Downarrow C}{\mathtt{case}\ R\ \mathtt{of}\ \mathtt{inl}_{\sigma+\tau} x \to P\ \mathtt{or}\ \mathtt{inr}_{\sigma+\tau} y \to Q \Downarrow C}$	case analysis (1. case)
$\dfrac{R \Downarrow \mathtt{inr}_{\sigma+\tau}\, S \qquad [S/y]\, Q \Downarrow C}{\mathtt{case}\ R\ \mathtt{of}\ \mathtt{inl}_{\sigma+\tau} x \to P\ \mathtt{or}\ \mathtt{inr}_{\sigma+\tau} y \to Q \Downarrow C}$	case analysis (2. case)
$\dfrac{[\mu x\colon \tau.\, P/x]\, P \Downarrow C}{\mu x\colon \tau.\, P \Downarrow C}$	fixed point
$\dfrac{}{\Lambda a.\, P \Downarrow \Lambda a.\, P}$	type-abstraction
$\dfrac{P \Downarrow \Lambda a.\, Q \qquad [\sigma/a]\, Q \Downarrow C}{P_\sigma \Downarrow C}$	type-application
$\dfrac{}{\mathtt{intro}_{(\forall a.\, \tau)}\, P \Downarrow \mathtt{intro}_{(\forall a.\, \tau)}\, P}$	constructor
$\dfrac{P \Downarrow \mathtt{intro}_{(\forall a.\, \tau)}\, Q \qquad Q \Downarrow C}{\mathtt{elim}_{(\forall a.\, \tau)}\, P \Downarrow C}$	destructor

Table 2: Operational semantics.

6.2.7. Lemma. The α-congruence leaves the operational semantics invariant (So the α-congruence is indeed a *congruence* relation.), *i.e.* in more detail:

$$\frac{P \equiv_\alpha P' \qquad P \Downarrow C \qquad P' \Downarrow C'}{C \equiv_\alpha C'}$$

and hence:

$$\frac{P \equiv_\alpha Q}{P \equiv^{op} Q}.$$

Proof. Structural induction on the recursive Definition 6.2.3. ∎

6.3 Categorical denotational semantics

Die Mathematiker sind eine Art Franzosen,
redet man zu ihnen, so übersetzen sie es in ihre Sprache
und dann ist es alsobald ganz etwas anderes.[10]
— GOETHE, Maxims and Reflexions (1829)

Finally we are going to apply the category theory from Section 3 and Section 4. We will map the defined above language into a category. As we will see, each language construct can be mapped in a natural way onto a categorical construct. We will show that this mapping is a sound (w.r.t.to the operational semantics) denotational semantics.

The following definition is inspired by [FP94].

6.3.1. Definition (denotational semantics). We use the categories $\mathcal{C}$ and $\mathcal{C}_\perp$, which we fixed in Subsection 5.4 together with the according involutory category $\breve{\mathcal{C}_\perp} := (\mathcal{C}_\perp)\breve{}$ (see Definition 3.3.3 and Theorem 5.4.6). Furthermore we will need the internal fixed point operator defined in Definition 4.2.6, which exists due to Lemma 4.2.7.

We define the function $\{\![\,.\,]\!\}$[11] on type-environments, environments, type-expressions, type-schemes and terms. We call this function the **denotational semantics** or, to emphasize that it maps into a category, the **categorical semantics**.

▶ **semantics of type-environments:** every type-environment is mapped onto an according category.

semantics of type-environments		
$\{\![\varepsilon]\!\}$	$:=$	$\mathbf{1}$
$\{\![\Theta a]\!\}$	$:=$	$\{\![\Theta]\!\} \times \breve{\mathcal{C}_\perp}$

▶ **semantics of environments:** every environment is mapped onto a functor.

$$\frac{\Theta \vdash \Gamma}{\{\![\Gamma]\!\}^\Theta : \mathcal{C} \leftarrow \{\![\Theta]\!\}}$$

semantics of environments		
For every $h \in \mathrm{Mor}\,\{\![\Theta]\!\}$:		
$\{\![\varepsilon]\!\}^\Theta h$	$:=$	$1_{\mathcal{C}}$
$\{\![\Gamma(x : \tau)]\!\}^\Theta h$	$:=$	$\{\![\Gamma]\!\}^\Theta h \times \{\![\tau]\!\}^\Theta h$

[10] Mathematicians are like Frenchmen: whenever you say something to them, they translate it into their own language, and at once it is something entirely different.

[11] See the footnote in Definition 5.3.1 about braces and brackets $\{\![\,.\,]\!\}$ versus double-brackets $[\![\,.\,,\,.\,]\!]$.

► **semantics of type-expressions and type-schemes:** every type-scheme is mapped onto a functor.

$$\frac{\Theta \vdash \tau}{\{\![\tau]\!\}^{\Theta} : \boldsymbol{\mathcal{C}} \leftarrow \{\![\Theta]\!\}}$$

semantics of type-expressions and type-schemes					
For every $h \in \mathrm{Mor}\,\{\![\Theta]\!\}$:					
$\{\![1]\!\}^{\Theta} h$	$:=$		$id_{1_{\boldsymbol{\mathcal{C}}}}$		
$\{\![a]\!\}^{a} h$	$:=$		$\grave{\mathsf{P}}\, h$		
$\{\![\tau]\!\}^{\Theta a}(h, h')$	$:=$		$\{\![\tau]\!\}^{\Theta} h$		
$\{\![\sigma + \tau]\!\}^{\Theta} h$	$:=$	$\{\![\sigma]\!\}^{\Theta} h$	$+$	$\{\![\tau]\!\}^{\Theta} h$	
$\{\![\sigma \times \tau]\!\}^{\Theta} h$	$:=$	$\{\![\sigma]\!\}^{\Theta} h$	$\times$	$\{\![\tau]\!\}^{\Theta} h$	
$\{\![\sigma^{\tau}]\!\}^{\Theta} h$	$:=$	$\{\![\sigma]\!\}^{\Theta} h$	$\Leftarrow$	$\big(\{\![\tau]\!\}^{\Theta}\big)^{\mathrm{op}} h^{*}$	
$\big(\{\![\forall a.\, \tau]\!\}^{\Theta} h\big) h'$	$:=$		$\{\![\tau]\!\}^{\Theta a}(h, h')$		$\forall\, h' \in \mathrm{Mor}\,\breve{\boldsymbol{\mathcal{C}}}_{\perp}$
$\{\![\boldsymbol{\mu} a.\, \tau]\!\}^{\Theta} h$	$:=$		$\mathsf{T}(\breve{\mathsf{H}}_{h})$		
wobei $\mathsf{H}_{h} h' := \{\![\tau]\!\}^{\Theta a}(h, h') \quad \forall\, h' \in \mathrm{Mor}\,\breve{\boldsymbol{\mathcal{C}}}_{\perp}$					

► **semantics of terms:** every term is mapped onto a (*not* necessarily natural) transformation.

$$\frac{\Theta, \Gamma \vdash P : \tau}{\{\![P]\!\}^{\Theta,\Gamma} : \{\![\tau]\!\}^{\Theta} \leftarrow \{\![\Gamma]\!\}^{\Theta}}$$

To simplify the description of the case analysis, we introduce $\forall\, A, B, C \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$ the function

$$\mathrm{flip} : \begin{cases} \boldsymbol{\mathcal{C}}(A^{B}, C) & \leftarrow \ \boldsymbol{\mathcal{C}}(A^{C}, B) \\ \mathrm{curry}(\mathrm{uncurry}\, f \ \cdot\ \mathit{swap}) & \leftarrow\!\shortmid \quad f \qquad \forall\, f \in \boldsymbol{\mathcal{C}}(A^{C}, B). \end{cases}$$

We assume the notations and references for environments and types as in Definition 6.1.5.

semantics of terms			
For every $A \in \mathrm{Ob}\,\{\![\Theta]\!\}$:			
$\{\![()]\!\}^{\Theta,\varepsilon}_{A}$	$:=$	$\boldsymbol{!}_{1}$	constant
$\{\![x]\!\}^{\Theta,(x:\tau)}_{A}$	$:=$	unit_{T}	variable
$\{\![P]\!\}^{\Theta,\Gamma(x:\tau)}_{A}$	$:=$	$\{\![P]\!\}^{\Theta,\Gamma}_{A} \cdot \grave{\pi}_{(\{\![\Gamma]\!\}^{\Theta} A,\, T)}$	extension of the environment
$\{\![\lambda x\colon \tau.\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\mathrm{curry}\, \{\![P]\!\}^{\Theta,\Gamma(x:\tau)}_{A}$	abstraction
$\{\![PQ]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\mathit{apply}_{(S,T)} \cdot \big\langle \{\![P]\!\}^{\Theta,\Gamma}_{A}, \{\![Q]\!\}^{\Theta,\Gamma}_{A} \big\rangle$	application
$\{\![(P,Q)]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\big\langle \{\![P]\!\}^{\Theta,\Gamma}_{A}, \{\![Q]\!\}^{\Theta,\Gamma}_{A} \big\rangle$	pairing
$\{\![\, \mathtt{fst}\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\grave{\pi}_{(S,T)} \cdot \{\![P]\!\}^{\Theta,\Gamma}_{A}$	projection to the left
$\{\![\, \mathtt{snd}\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\acute{\pi}_{(S,T)} \cdot \{\![P]\!\}^{\Theta,\Gamma}_{A}$	projection to the right
$\{\![\, \mathtt{inl}_{\sigma+\tau}\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\grave{\iota}_{(S,T)} \cdot \{\![P]\!\}^{\Theta,\Gamma}_{A}$	injection to the left
$\{\![\, \mathtt{inr}_{\sigma+\tau}\, Q]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\acute{\iota}_{(S,T)} \cdot \{\![Q]\!\}^{\Theta,\Gamma}_{A}$	injection to the right
$\{\![\, [P, Q]\,]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\mathrm{flip}\big[\mathrm{flip}\, \{\![P]\!\}^{\Theta,\Gamma}_{A}, \mathrm{flip}\, \{\![Q]\!\}^{\Theta,\Gamma}_{A}\big]$	case analysis
$\{\![\mu x\colon \tau.\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$Y_{T} \cdot \mathrm{curry}\, \{\![P]\!\}^{\Theta,\Gamma(x:\tau)}_{A}$	fixed point
$\big(\{\![\Lambda a.\, P]\!\}^{\Theta,\Gamma}_{A}\big)_{A'}$	$:=$	$\{\![P]\!\}^{\Theta a,\Gamma}_{(A,A')} \quad \forall\, A' \in \mathrm{Ob}\,\breve{\boldsymbol{\mathcal{C}}}_{\perp}$	type-abstraction
$\{\![P_{\sigma}]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\big(\{\![P]\!\}^{\Theta,\Gamma}_{A}\big)_{S}$	type-application
$\{\![\, \mathtt{intro}_{(\forall a.\, \tau)}\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\grave{\mathsf{P}}\, in_{\breve{\mathsf{H}}} \cdot \{\![P]\!\}^{\Theta,\Gamma}_{A}$	constructor
$\{\![\, \mathtt{elim}_{(\forall a.\, \tau)}\, P]\!\}^{\Theta,\Gamma}_{A}$	$:=$	$\grave{\mathsf{P}} in^{*}_{\breve{\mathsf{H}}} \cdot \{\![P]\!\}^{\Theta,\Gamma}_{A}$	destructor
where $S := \{\![\sigma]\!\}^{\Theta} A$ and $T := \{\![\tau]\!\}^{\Theta} A$ as well as $\mathsf{H} := \{\![\forall a.\, \tau]\!\}^{\Theta} A$			

6.3.2. Definition (denotational semantic equivalence). Analogically to the semantic equivalence ($\equiv^{op}$) w.r.t. the operational semantics, we define a semantic equivalence w.r.t. the operational semantics: Two closed terms P and Q of the same type are called **(denotationally) semantically equivalent**, if they have the same denotational semantics, we write $P \equiv^{cat} Q$, *i.e.* in more detail:

$$P \equiv^{cat} Q \;:\Longleftrightarrow\; \{\!\!\{P\}\!\!\} = \{\!\!\{Q\}\!\!\}.$$

6.3.3. Lemma (substitution lemma).

$$\frac{\Theta, \Gamma(x:\tau) \vdash P : \sigma \qquad \Theta, \Gamma \vdash Q : \tau \qquad A \in \mathrm{Ob}\,\{\!\!\{\Theta\}\!\!\}}{\{\!\!\{\,[Q/x]\,P\}\!\!\}_A^{\Theta,\Gamma} = \{\!\!\{P\}\!\!\}_A^{\Theta,\Gamma(x:\tau)} \cdot \langle id_{\{\!\!\{\Gamma\}\!\!\}^\Theta A}, \{\!\!\{Q\}\!\!\}_A^{\Theta,\Gamma}\rangle}$$

Proof. Immediately by structural induction on the recursive definition of $[\,.\,/\,.\,]$ in Definition 6.2.1 using the laws from Section 4. ∎

6.3.4. Theorem (soundness of the denotational semantics). The denotational semantics (Definition 6.3.9) is **sound** w.r.t. the operational semantics (Definition 6.2.3), *i.e.*, for every closed terms P, Q holds:

$$\frac{P \equiv^{op} Q}{P \equiv^{cat} Q}$$

Proof. Induction on the inference rules from the definition of $\Downarrow$ in Definition 6.2.3, where the above substitution Lemma 6.3.3 is used, yields:

$$\frac{\varepsilon, \varepsilon \vdash P : \tau \qquad \varepsilon, \varepsilon \vdash P' : \tau \qquad P \Downarrow P'}{\{\!\!\{P\}\!\!\}_1^{\varepsilon,\varepsilon} = \{\!\!\{P'\}\!\!\}_1^{\varepsilon,\varepsilon}},$$

which we will investigate later in more detail. Induction on the inference rules from the definition of $\equiv_\alpha$ in Definition 6.2.5, where the substitution Lemma 6.3.3 is used once again, yields:

$$\frac{\varepsilon, \varepsilon \vdash P : \tau \qquad \varepsilon, \varepsilon \vdash P' : \tau' \qquad P \equiv_\alpha P'}{\{\!\!\{P\}\!\!\}_1^{\varepsilon,\varepsilon} = \{\!\!\{P'\}\!\!\}_1^{\varepsilon,\varepsilon}}.$$

By means of the definition of $\equiv^{op}$ in Definition 6.3.2 follows the proposition.

As an example, we will demonstrate the above induction on the inference rules Definition 6.2.3 for the case analysis, which seems to be the most difficult rule: We observe the inference rule

$$\frac{R \Downarrow \mathtt{inl}_{\sigma+\tau}\, S \qquad [S/x]\,P \Downarrow C}{\mathtt{case}\; R\; \mathtt{of}\; \mathtt{inl}_{\sigma+\tau} x \to P \;\mathtt{or}\; \mathtt{inr}_{\sigma+\tau} y \to Q \Downarrow C}.$$

The induction premise is

$$\{\!\!\{R\}\!\!\}_1^{\varepsilon,\varepsilon} = \{\!\!\{\,\mathtt{inl}_{\sigma+\tau}\, S\}\!\!\}_1^{\varepsilon,\varepsilon} \quad \text{and} \quad \{\!\!\{\,[S/x]\,P\}\!\!\}_1^{\varepsilon,\varepsilon} = \{\!\!\{C\}\!\!\}_1^{\varepsilon,\varepsilon}$$

and we have to show

$$\{\!\!\{\mathtt{case}\; R\; \mathtt{of}\; \mathtt{inl}_{\sigma+\tau} x \to P \;\mathtt{or}\; \mathtt{inr}_{\sigma+\tau} y \to Q\}\!\!\}_1^{\varepsilon,\varepsilon} = \{\!\!\{C\}\!\!\}_1^{\varepsilon,\varepsilon}.$$

We calculate:

$$
\begin{array}{rl}
& \{\![\texttt{case}\ R\ \texttt{of}\ \texttt{inl}_{\sigma+\tau}x\ \rightarrow\ P\ \texttt{or}\ \texttt{inr}_{\sigma+\tau}y\ \rightarrow\ Q]\!\}_1^{\varepsilon,\varepsilon} \\
= & \quad \{\ \text{def. } \texttt{case}\ \} \\
& \{\![\,[\lambda x\colon \sigma.\ P, \lambda y\colon \tau.\ Q]R\,]\!\}_1^{\varepsilon,\varepsilon} \\
= & \quad \{\ \text{induction premise for } R \text{ and def. } \{\![\,.\,]\!\} \text{ according to Definition 6.2.3}\ \} \\
& \mathit{apply} \cdot \left\langle \text{flip}\left[\,\text{flip}(\text{curry}\ \{\![P]\!\}_1^{\varepsilon,(x:\sigma)}), \text{flip}(\text{curry}\ \{\![Q]\!\}_1^{\varepsilon,(y:\tau)})\,\right], \grave{\imath} \cdot \{\![S]\!\}_1^{\varepsilon,\varepsilon} \right\rangle \\
= & \quad \{\ \times\text{-fusion, def. uncurry, def. flip and uncurry} \cdot \text{curry} = \mathit{id}\ \} \\
& \text{uncurry}\left[\text{curry}(\{\![P]\!\}_1^{\varepsilon,(x:\sigma)} \cdot \mathit{swap}), \text{curry}(\{\![Q]\!\}_1^{\varepsilon,(y:\tau)} \cdot \mathit{swap})\right]\ \cdot\ \mathit{swap} \cdot \langle \mathit{id}, \grave{\imath} \cdot \{\![S]\!\}_1^{\varepsilon,\varepsilon}\rangle \\
= & \quad \{\ \text{def. uncurry, } \times\text{-fusion and } [\,.\,,\,.\,]\text{-cancelation}\ \} \\
& \mathit{apply} \cdot \left(\text{curry}(\{\![P]\!\}_1^{\varepsilon,(x:\sigma)} \cdot \mathit{swap}) \times \mathit{id}\right)\ \cdot\ \mathit{swap} \cdot \langle \mathit{id}, \{\![S]\!\}_1^{\varepsilon,\varepsilon}\rangle \\
= & \quad \{\ \text{def. uncurry, uncurry} \cdot \text{curry} = \mathit{id} \text{ and } \mathit{swap} \cdot \mathit{swap} = \mathit{id}\ \} \\
& \{\![P]\!\}_1^{\varepsilon,(x:\tau)} \cdot \langle \mathit{id}, \{\![S]\!\}_1^{\varepsilon,\varepsilon}\rangle \\
= & \quad \{\ \text{substitution Lemma 6.3.3}\ \} \\
& \{\![\,[S/x]\,P]\!\}_1^{\varepsilon,\varepsilon} \\
= & \quad \{\ \text{induction premise}\ \} \\
& \{\![C]\!\}_1^{\varepsilon,\varepsilon}.
\end{array}
$$

■

6.3.5. Definition (η-conversion). We want to view programs of a functional type (*i.e.* a type of the form σ^τ) as morphisms of a category. We may assume, that these programs have the syntax $\lambda V\colon T.\ X$ (*cf.* Definition 6.1.1), because

$$(\lambda x\colon \tau.\ Px)Q\ \equiv^{op}\ PQ,$$

which can be shown easily, using the application rule in Definition 6.2.3. We will specify this by introducing the **η-conversion** relation $\equiv_{\alpha\eta}$, which we will combine directly with the $\equiv_\alpha$-congruence:

$$\frac{P \equiv_\alpha Q}{P \equiv_{\alpha\eta} Q} \qquad \frac{Q \equiv_{\alpha\eta} P}{P \equiv_{\alpha\eta} Q} \qquad \frac{P \equiv_{\alpha\eta} Q}{\lambda x\colon \tau.\ Px \equiv_{\alpha\eta} Q}$$

The operational semantics is *not* invariant under $\alpha\eta$-conversion, but after all the following holds:

$$\frac{P \equiv_{\alpha\eta} P' \qquad Q \equiv_\alpha Q'}{PQ\ \equiv^{op}\ P'Q'}.$$

6.3.6. Lemma. The denotational semantics is invariant under $\alpha\eta$-conversion, *i.e.*:

$$\frac{P \equiv_{\alpha\eta} Q}{P\ \equiv^{cat}\ Q}$$

Proof. Immediately by means of the definition of $\{\![\,.\,]\!\}$ for abstraction, application and variable in Definition 6.3.1. ■

6.3.7. Definition and Lemma (category of types and programs). In virtue of

$$\begin{aligned}
\mathrm{Ob}\,\mathbf{\Lambda} &:= {}^{Type}/_{\equiv_\alpha} \\
\mathbf{\Lambda}([\sigma]_{\equiv_\alpha}, [\tau]_{\equiv_\alpha}) &:= {}^{\{\lambda x\colon \tau.\, P \;\mid\; P \in Prog^{\varepsilon,(x:\tau)}\,\sigma\}}/_{\equiv_\alpha} \quad \forall\, \sigma, \tau \in Type \\
id^{\mathbf{\Lambda}}_{[\tau]_{\equiv_\alpha}} &:= [\lambda x\colon \tau.\, x]_{\equiv_\alpha} \quad \forall\, \tau \in Type \\
[\lambda x\colon \tau.\, P]_{\equiv_\alpha} \cdot_{\mathbf{\Lambda}} [\lambda y\colon \varrho.\, Q]_{\equiv_\alpha} &:= [\lambda y\colon \varrho.\; [Q/x]\, P]_{\equiv_\alpha} \quad \begin{array}{l} \forall\, P \in Prog^{\varepsilon,(x:\tau)}\,\sigma \\ \forall\, Q \in Prog^{\varepsilon,(y:\sigma)}\,\tau \end{array}
\end{aligned}$$

we can define the small category $\mathbf{\Lambda}$ of all types with programs as morphisms.

Proof. That $\mathbf{\Lambda}$ is well-defined, follows by the inference rules from the definition of the α-congruence in Definition 6.2.5. We have to show the category-axioms from Definition 2.2.2. By means of the type inference rules Definition 6.1.5 we get that $id^{\mathbf{\Lambda}}$ and the composition $\cdot_{\mathbf{\Lambda}}$ are well-typed. With $S := [\sigma]_{\equiv_\alpha}$, $T := [\tau]_{\equiv_\alpha}$ and $\varphi := [\lambda x\colon \tau.\, P]$ we calculate

$$id_S \cdot \varphi = [\lambda z\colon \sigma.\, z]_{\equiv_\alpha} \cdot [\lambda x\colon \tau.\, P]_{\equiv_\alpha} = [\lambda x\colon \tau.\; [P/z]\, z]_{\equiv_\alpha} = [\lambda x\colon \tau.\, P]_{\equiv_\alpha} = \varphi$$

and

$$\varphi \cdot id_T = [\lambda x\colon \tau.\, P]_{\equiv_\alpha} \cdot [\lambda z\colon \tau.\, z]_{\equiv_\alpha} = [\lambda z\colon \tau.\; [z/x]\, P]_{\equiv_\alpha} = [\lambda x\colon \tau.\, P]_{\equiv_\alpha} = \varphi.$$

The associativity of $\cdot_{\mathbf{\Lambda}}$ can be shown, using

$$\frac{\varepsilon, (x : \tau_2) \vdash P : \tau_1 \qquad \varepsilon, (y : \tau_3) \vdash Q : \tau_2 \qquad \varepsilon, (z : \tau_4) \vdash R : \tau_3}{[R/y]([Q/x]\, P) = [\,[R/y]\, Q/x]\, P}$$

which follows by means of the inference rules of the definition of $[\,\bullet/\bullet\,]$ in Definition 6.2.1. ∎

6.3.8. Note ('everything modulo $\equiv_\alpha$'). Since our denotational semantics $\{\![\,\bullet\,]\!\}$ as well as the composition $\cdot_{\mathbf{\Lambda}}$ are independent of the representative of the $\equiv_\alpha$-equivalence-class, we can and we will identify a term and its $\equiv_\alpha$-equivalence-class, *i.e.* we will simply omit the brackets $[\,\bullet\,]_{\equiv_\alpha}$ and in particular we will simply write $=$ where we rather should write $\equiv_\alpha$.

6.3.9. Definition and Lemma (semantics functor). As a special case of our denotational semantics $\{\![\,\bullet\,]\!\}$ we define a semantics S of programs, which are $\mathbf{\Lambda}$-morphisms. For types $\sigma, \tau \in$ *Type* we define the functions

$$\begin{aligned}
\mathsf{S}P &:= \mathrm{uncurry}(\{\![P]\!\}^{\varepsilon,\varepsilon}_{1_1}) \cdot unit^{-1} \quad \forall\, P \in \mathrm{Mor}\,\mathbf{\Lambda} \\
\mathsf{S}\tau &:= \{\![\tau]\!\}^{\varepsilon}\, 1_{\mathbf{1}} \quad \forall\, \tau \in Type\,.
\end{aligned}$$

We get a functor

$$\mathsf{S} : \boldsymbol{\mathcal{C}} \leftarrow \mathbf{\Lambda}.$$

Proof. Because of Lemma 6.3.6, S is obviously well-defined. Notice that the domain of S consists of $\equiv_\alpha$-equivalence-classes (*cf.* Note 6.3.8). We have to show the functor-axioms from Definition 2.2.9. By means of the definition of $\{\![\,\bullet\,]\!\}$ in Definition 6.2.3 we get

$$\frac{\sigma, \tau \in Type \qquad P \in Prog\, \sigma^{\tau}}{\{\![\tau]\!\}^{\varepsilon}\, 1 \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}} \qquad \{\![P]\!\}^{\varepsilon,\varepsilon}_{1} : \{\![\tau]\!\}^{\varepsilon}\, 1 \xleftarrow[\boldsymbol{\mathcal{C}}]{} 1_{\boldsymbol{\mathcal{C}}}}$$

and thus S satisfies (typing). Let $\tau \in \mathit{Type}$.

$$
\begin{array}{ll}
& \mathsf{S}\, id_\tau \\
= & \quad \{ \text{ def. } id^{\mathbf{\Lambda}} \ \} \\
& \mathsf{S}(\lambda x\colon \tau.\ x) \\
= & \quad \{ \text{ def. } \mathsf{S} \ \} \\
& \text{uncurry}(\{\![\lambda x\colon \tau.\ x]\!\}_1^{\varepsilon,\varepsilon}) \cdot \mathit{unit}^{-1}_{\{\![\tau]\!\}^\varepsilon 1} \\
= & \quad \{ \text{ def. } \{\![\,\cdot\,]\!\}\text{-abstraction and uncurry} \cdot \text{curry} = id \ \} \\
& \{\![x]\!\}_1^{\varepsilon,(x:\tau)} \cdot \mathit{unit}^{-1}_{\{\![\tau]\!\}^\varepsilon 1} \\
= & \quad \{ \text{ def. } \{\![\,\cdot\,]\!\}\text{-variable and } \mathit{unit} \cdot \mathit{unit}^{-1} = id \ \} \\
& id_{\{\![\tau]\!\}^\varepsilon 1} \\
= & \quad \{ \text{ def. } \mathsf{S} \ \} \\
& id_{\mathsf{S}\,\tau}
\end{array}
$$

Thus S satisfies (identity).

Let $\varepsilon, (x : \tau) \vdash P : \sigma$ and $\varepsilon, (y : \varrho) \vdash Q : \tau$.

$$
\begin{array}{ll}
& \mathsf{S}\big((\lambda x\colon \tau.\ P) \ \cdot \ (\lambda y\colon \varrho.\ Q)\big) \\
= & \quad \{ \text{ def. } \cdot_{\mathbf{\Lambda}} \ \} \\
& \mathsf{S}(\lambda y\colon \varrho.\ [Q/x]\, P) \\
= & \quad \{ \text{ def. } \mathsf{S} \ \} \\
& \text{uncurry}(\{\![\lambda y\colon \varrho.\ [Q/x]\, P]\!\}_1^{\varepsilon,\varepsilon}) \cdot \mathit{unit}^{-1} \\
= & \quad \{ \ \{\![\,\cdot\,]\!\}\text{-abstraction \& curry} \cdot \text{uncurry} = id \ \} \\
& \{\![\,[Q/x]\, P]\!\}_1^{\varepsilon,(y:\varrho)} \cdot \mathit{unit}^{-1} \\
= & \quad \{ \text{ substitution Lemma 6.3.3 } \} \\
& \{\![P]\!\}_1^{\varepsilon,(y:\varrho)(x:\tau)} \cdot \big\langle id_1, \{\![Q]\!\}_1^{\varepsilon,(y:\varrho)} \big\rangle \cdot \mathit{unit}^{-1} \\
= & \quad \{ \text{ uncurry} \cdot \text{curry} = id \ \& \ \{\![\,\cdot\,]\!\}\text{-abstraction } \} \\
& \text{uncurry}(\{\![\lambda x\colon \tau.\ P]\!\}_1^{\varepsilon,(y:\varrho)}) \ \cdot \ \big\langle id_1, \{\![Q]\!\}_1^{\varepsilon,(y:\varrho)} \big\rangle \cdot \mathit{unit}^{-1} \\
= & \quad \{ \text{ def. uncurry \& } \times\text{-fusion \& } \langle\,\cdot\,,\,\cdot\,\rangle\text{-fusion } \} \\
& \mathit{apply} \cdot \big\langle \{\![\lambda x\colon \tau.\ P]\!\}_1^{\varepsilon,(y:\varrho)} \cdot \mathit{unit}^{-1}, \{\![Q]\!\}_1^{\varepsilon,(y:\varrho)} \cdot \mathit{unit}^{-1} \big\rangle \\
= & \quad \{ \ \{\![\,\cdot\,]\!\}\text{-extension of environment \& def. } \mathit{unit}^{-1} \ \& \ \mathbf{!}\text{-fusion } \} \\
& \mathit{apply} \cdot \big\langle \{\![\lambda x\colon \tau.\ P]\!\}_1^{\varepsilon,\varepsilon} \cdot \mathbf{!}_{\{\![y]\!\}^\varepsilon 1}, \{\![Q]\!\}_1^{\varepsilon,(y:\varrho)} \cdot \mathit{unit}^{-1} \big\rangle \\
= & \quad \{ \text{ def. uncurry \& } \times\text{-fusion } \} \\
& \text{uncurry}(\{\![\lambda x\colon \tau.\ P]\!\}_1^{\varepsilon,\varepsilon}) \cdot \langle \mathbf{!}_{\{\![y]\!\}^\varepsilon 1}, \{\![Q]\!\}_1^{\varepsilon,(y:\varrho)} \cdot \mathit{unit}^{-1} \rangle \\
= & \quad \{ \ id = \mathit{unit}^{-1} \cdot \mathit{unit} \ \& \ \mathit{unit}_{\{\![y]\!\}^\varepsilon 1} = \acute{\pi}_{(1,\{\![y]\!\}^\varepsilon 1)} \ \& \ \langle\,\cdot\,,\,\cdot\,\rangle\text{-cancelation } \} \\
& \text{uncurry}(\{\![\lambda x\colon \tau.\ P]\!\}_1^{\varepsilon,\varepsilon}) \cdot \mathit{unit}^{-1} \ \cdot \ \{\![Q]\!\}_1^{\varepsilon,(y:\varrho)} \cdot \mathit{unit}^{-1} \\
= & \quad \{ \text{ uncurry} \cdot \text{curry} = id \ \& \ \{\![\,\cdot\,]\!\}\text{-abstraction } \} \\
& \text{uncurry}(\{\![\lambda x\colon \tau.\ P]\!\}_1^{\varepsilon,\varepsilon}) \cdot \mathit{unit}^{-1} \ \cdot \ \text{uncurry}(\{\![\lambda y\colon \varrho.\ Q]\!\}_1^{\varepsilon,\varepsilon}) \cdot \mathit{unit}^{-1} \\
= & \quad \{ \text{ def. } \mathsf{S} \ \} \\
& \mathsf{S}(\lambda x\colon \tau.\ P) \ \cdot \ \mathsf{S}(\lambda y\colon \varrho.\ Q).
\end{array}
$$

And thus S satisfies (multiplicativity) as well. ∎

6.3.10. Definition (observational equivalence). Two programs $P, Q \in \mathit{Prog}\,\sigma^\tau$ are called **observationally equivalent**, we write $P \equiv^{obs} Q$, if they both do terminate or both do not terminate likewise in every 'program context', *i.e.* in more detail:

$$P \equiv^{obs} Q \;:\Longleftrightarrow\; \forall\,\varrho \in \mathit{Type}\ \forall\,L \in \mathit{Prog}\,\varrho^{\sigma}\ \forall\,R \in \mathit{Prog}\,\tau :: \\ \big(L(PR)\Downarrow \;\Longleftrightarrow\; L(QR)\Downarrow\big).$$

6.3.11. Note (adequacy of the denotational semantics). In this paper we will not investigate whether the denotational semantics is **adequate** w.r.t. the observational equivalence, *i.e.*:

$$\frac{P \equiv^{cat} Q}{P \equiv^{obs} Q}$$

for every closed terms P and Q.

6.4 Application of the hylomorphism-fusion

If we calculate modulo $\equiv^{cat}$, the semantics functor $\mathsf{S} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\Lambda}$ permits to use all the laws from the category $\boldsymbol{\mathcal{C}}$ in the category $\boldsymbol{\Lambda}$. For most of the laws this immediately is possible, since their only precondition is well-typedness. As an example we state the $(\!|\,.\,|\!)$-reflection and $[\,.\,,\,.\,]$-cancelation rules:

$$\begin{aligned} (\!|\lambda x\colon (\forall a.\,\tau)(\boldsymbol{\mu} a.\,\tau).\ \mathtt{intro}_{(\forall a.\,\tau)}\,x|\!)_{(\forall a.\,\tau)} &\equiv^{cat} \lambda y\colon (\boldsymbol{\mu} a.\,\tau).\ y,\\ (\lambda z\colon \sigma+\tau.\ [P,Q]z)\ \cdot\ (\lambda x\colon \sigma.\ \mathtt{inl}_{\sigma+\tau}\,x) &\equiv^{cat} P, \end{aligned}$$

where we have used the definition of $(\!|\,.\,|\!)$ on terms from Definition 6.4.4 in advance.

Having a closer look at the above laws (and most of the others), we find that it is possible, to derive them directly out of the operational semantics, and we would even get $\equiv^{obs}$ directly, instead of $\equiv^{cat}$.

A lucid exception are the fusion rules for cata- and anamorphisms, and the hylomorphism-fusion, which are valid only for F-(co)algebra-morphisms. We will investigate this in the following.

6.4.1. Definition (type-polynomial). In Definition 6.1.3 we had defined a notation for the type-substitution, resembling the application of terms. We use this notation to define the set of **type-polynomials** *PType* (*cf.* Definition 5.2.1) by

$$\frac{a \in \mathit{Tvar}}{(\forall a.\,a) \in \mathit{PType}} \qquad \frac{\tau \in \mathit{Type}}{(\forall a.\,\tau) \in \mathit{PType}}$$

$$\frac{(\forall a.\,\sigma), (\forall a.\,\varrho) \in \mathit{PType}}{(\forall a.\,\sigma\times\varrho), (\forall a.\,\sigma+\varrho) \in \mathit{PType}} \qquad \frac{(\forall a.\,\sigma) \in \mathit{PType} \quad \tau \in \mathit{Type}}{(\forall a.\,\sigma^{\tau}) \in \mathit{PType}}.$$

and the application of those on types by

$$(\forall a.\,\sigma)\tau = [\tau/a]\,\sigma \quad \forall\,a \in \mathit{Tvar} \quad \forall\,(\forall a.\,\sigma) \in \mathit{PType} \quad \forall\,\tau \in \mathit{Type}$$

according to Definition 6.1.3. The application of type-polynomials on programs $P \in \mathit{Prog}\,\alpha^{\beta}$ is defined by recursion on the above inference rules by

$$\begin{aligned} (\forall a.\,a)P &:= P\\ (\forall a.\,\tau)P &:= \lambda z\colon \tau.\ z\\ (\forall a.\,\sigma\times\varrho)P &:= \lambda z\colon \forall a.\,(\sigma\times\varrho)\beta.\ \big((\forall a.\,\sigma)P\,(\mathtt{fst}\,z), (\forall a.\,\tau)P\,(\mathtt{snd}\,z)\big)\\ (\forall a.\,\sigma+\varrho)P &:= \lambda z\colon \forall a.\,(\sigma+\varrho)\beta.\\ &\qquad\qquad \big[\lambda x\colon (\forall a.\,\sigma)\beta.\ \mathtt{inl}_{(\forall a.\,\sigma)\alpha+(\forall a.\,\varrho)\alpha}((\forall a.\,\sigma)\,Px),\\ &\qquad\qquad\ \lambda y\colon (\forall a.\,\varrho)\beta.\ \mathtt{inr}_{(\forall a.\,\sigma)\alpha+(\forall a.\,\varrho)\alpha}((\forall a.\,\varrho)\,Py)\big]z\\ (\forall a.\,\sigma^{\tau}) &:= \lambda f\colon \big((\forall a.\,\sigma)\beta\big)^{\tau}.\ \lambda z\colon \tau.\ (\forall a.\,\sigma)P\,(fz) \end{aligned}$$

6.4.2. Lemma. The application of type-polynomials

(i) is independent of α-congruence,

(ii) maps programs on programs, such that

$$\frac{(\forall a.\,\varrho) \in \mathit{PType} \qquad P \in \mathit{Prog}\,\sigma^{\tau}}{(\forall a.\,\varrho)P \in \mathit{Prog}\,\big((\forall a.\,\varrho)\sigma\big)^{(\forall a.\,\varrho)\tau}},$$

(iii) of the form $(\forall a.\,\varrho) \in \mathit{PType}$ is a functor $(\forall a.\,\varrho) : \mathbf{\Lambda} \leftarrow \mathbf{\Lambda}$.

Proof. All of the propositions easily follow by induction on the recursive definition in Definition 6.4.1, where (i) and (ii) may already be used for (iii), and thus we only have to show the functor-axioms. Intuitively, the propositions are evident, since the application of $(\forall a.\,\varrho)$ on P yields a term, which operates at each case like P on all components of products and sums of the type-scheme ϱ. ∎

6.4.3. Lemma. The denotational semantics maps type-polynomials on polynomial functors, *i.e.*:

$$\frac{(\forall a.\,\tau) \in \mathit{PType}}{\exists\,\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}}) :: \ \mathsf{F}\cdot\grave{\mathsf{P}} = \{\![\forall a.\,\varrho]\!\}^{\varepsilon} 1}$$

Proof. Since type-polynomials do only have constant exponents, they are mapped onto co-variant functors by the denotational semantics. A comparison of the inference rules from the definition of $\mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ (*cf.* Definition 5.2.1) with those from the definition of *PType* (*cf.* Definition 6.4.1) yields the proposition. ∎

6.4.4. Definition. We introduce some abbreviations for specific programs: Let $x, f \in \mathit{Var}$, $a \in \mathit{Tvar}$ and

$$P \in \mathbf{\Lambda}\big(\sigma, (\forall a.\,\varrho)\sigma\big) \text{ und } Q \in \mathbf{\Lambda}\big((\forall a.\,\varrho)\tau, \tau\big).$$

We set

$$\begin{aligned}
(\![P]\!)_{(\forall a.\,\varrho)} &:= \mu f\colon \sigma^{\boldsymbol{\mu} a.\,\varrho}.\ \lambda x\colon (\boldsymbol{\mu} a.\,\varrho).\ P((\forall a.\,\varrho)f(\mathtt{elim}_{(\forall a.\,\varrho)}\,x)),\\
[\![Q]\!]_{(\forall a.\,\varrho)} &:= \mu f\colon (\boldsymbol{\mu} a.\,\varrho)^{\tau}.\ \lambda x\colon \tau.\ \mathtt{intro}_{(\forall a.\,\varrho)}((\forall a.\,\varrho)f(Qx)),\\
[\![\,P,\,Q\,]\!]_{(\forall a.\,\varrho)} &:= \mu f\colon \sigma^{\tau}.\ \lambda x\colon \tau.\ P((\forall a.\,\varrho)f(Qx)).
\end{aligned}$$

6.4.5. Corollary. Under the preconditions of the above Definition 6.4.4 and $\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$ with $\mathsf{F}\cdot\grave{\mathsf{P}} = \{\![\forall a.\,\varrho]\!\}^{\varepsilon} 1$ holds:

$$\begin{aligned}
\mathsf{S}(\![P]\!)_{(\forall a.\,\varrho)} &= (\![\mathsf{S}P]\!)_{\mathsf{F}},\\
\mathsf{S}[\![Q]\!]_{(\forall a.\,\varrho)} &= [\![\mathsf{S}Q]\!]_{\mathsf{F}},\\
\mathsf{S}[\![\,P,\,Q\,]\!]_{(\forall a.\,\varrho)} &= [\![\,\mathsf{S}P,\,\mathsf{S}Q\,]\!]_{\mathsf{F}}.
\end{aligned}$$

This is easy to see, just by substituting the definitions on the left hand side and comparing the result with the according UP for the right hand side.

6.4.6. Example (recursive definition by catamorphism). We use the function *dup*, which we defined in Note 6.1.8. But this time we view the image under S in $\boldsymbol{\mathcal{C}}$. As in Note 6.1.8, we use the abbreviations

$$\begin{aligned}
\mathit{Zero} &:= \mathsf{S}(\lambda x\colon 1.\ \mathtt{intro}_{(\forall a.\,1+a)}(\mathtt{inl}_{1+\mathit{Nat}}\,x)) &&= \mathit{in}_{\mathsf{F}}\cdot\grave{\imath},\\
\mathit{Succ} &:= \mathsf{S}(\lambda y\colon \mathit{Nat}.\ \mathtt{intro}_{(\forall a.\,1+a)}(\mathtt{inr}_{1+\mathit{Nat}}\,y)) &&= \mathit{in}_{\mathsf{F}}\cdot\acute{\imath},
\end{aligned}$$

but this time for the images of the constructors components under S, where

$$\mathsf{F} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}}) \text{ with } \mathsf{F} \cdot \grave{\mathsf{P}} = \{[(\forall a.\, 1 + a)]\}^{\varepsilon} 1 \text{ also } \mathsf{F} = \mathsf{K}_1 + \mathsf{Id}.$$

The definition (according to the original program in Note 6.1.8) of $dup \in \mathcal{C}(\mathsf{TF}, \mathsf{TF})$ is given by the equations:

$$\begin{aligned} dup \cdot Zero &= Zero, \\ dup \cdot Succ &= Succ \cdot Succ \cdot dup \end{aligned}$$

We can combine theses two equations, using $[\,\centerdot\,,\centerdot\,]$, which is an equivalence transformation, since we can retrieve the original equations by multiplication by $\grave{\imath}$ or $\acute{\imath}$, respectively:

$$dup \cdot [Zero, Succ] = [Zero, Succ \cdot Succ \cdot dup]$$

On the left hand side we substitute the definitions of *Zero* and *Succ* and use $[\,\centerdot\,,\centerdot\,]$-fusion and reflection. On the left hand side we use $+$-fusion:

$$dup \cdot in_{\mathsf{F}} = [Zero, Succ \cdot Succ] \cdot (id + dup)$$

The constructor in_{F} is an isomorphism with inverse out_{F}. The definition of F yields $\mathsf{F}\,dup = id + dup$:

$$dup = [Zero, Succ \cdot Succ] \cdot \mathsf{F}\,dup \cdot out_{\mathsf{F}} = \big([Zero, Succ \cdot Succ] \overset{\mathsf{F}}{\leftarrow\!\circ}\, out_{\mathsf{F}}\big)\, dup$$

This is precisely the UP of the catamorphism and thus:

$$\begin{aligned} dup &= (\![\, [Zero, Succ \cdot Succ] \,]\!)_{\mathsf{F}} \\ &= \mathsf{S}\big([\lambda z\colon 1 + Nat. \\ &\qquad [\lambda x\colon 1.\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inl}_{1+Nat}\, x), \\ &\qquad\quad \lambda y\colon Nat.\ \mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}(\mathtt{intro}_{(\forall a.\, 1+a)}(\mathtt{inr}_{1+Nat}\, y))) \\ &\qquad]z]\big)_{(\forall a.\, 1+a)} \end{aligned}$$

6.4.7. Note. The above calculation in Example 6.4.6 may be generalized. An algorithm, which transforms many recursive definitions into cata-, ana- & hylomorphisms, can be found in [HIT96].

6.4.8. Definition (substitution of con- & destructors). For $(\forall a.\, \sigma), (\forall a.\, \tau) \in PType$, $\varrho \in Type$ and

$$\varrho \xleftarrow[\Lambda]{P} (\forall a.\, \sigma)\varrho \xleftarrow[\Lambda]{Q} \varrho$$

we define the substitution operators

$$[\varrho / \mu a.\, \tau] \qquad \text{and} \qquad \begin{bmatrix} \varrho \ / & \mu a.\, \tau \\ P \ / & \mathtt{intro}_{(\forall a.\, \tau)} \\ Q \ / & \mathtt{elim}_{(\forall a.\, \tau)} \end{bmatrix}$$

First we define the operator $[\varrho/\boldsymbol{\mu}a.\,\tau]$ recursively over the syntax of type-expressions:

$$
\begin{array}{llll}
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ 1 & := & 1 & \text{constant} \\[2ex]
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ a & := & a & \text{variable} \\[2ex]
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ (\boldsymbol{\mu}a.\,\pi) & := & \begin{cases} \varrho & : \pi = \tau, \\ \boldsymbol{\mu}a.\,[\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi & : \pi \neq \tau \end{cases} & \text{type fixed point} \\[2ex]
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ (\pi + \pi') & := & [\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi\ +\ [\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi' & \text{sum} \\[2ex]
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ (\pi \times \pi') & := & [\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi\ \times\ [\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi' & \text{product} \\[2ex]
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ \pi^{\pi'} & := & ([\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi)^{[\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi'} & \text{exponent} \\[2ex]
[\varrho/\boldsymbol{\mu}a.\,\tau]\ \ (\forall a.\,\pi) & := & \forall a.\ [\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi & \text{type-abstraction}
\end{array}
$$

Now we define the operator

$$\begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} := [///]$$

on terms according to Table 3 and by

$$[///]\big(f(P_1, \dots, P_n)\big)\ :=\ f([///]\,P_1, \dots, [///]\,P_n)$$

on all remaining expressions. Finally, we define two abbreviations for the special cases that the substitution operator is independent of P or Q, respectively:

$$\begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \end{bmatrix} R := \begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R \quad \forall \begin{array}{l}\text{terms } R \text{ which} \\ \text{do not contain an } \mathtt{elim}_{(\forall a.\,\tau)}\end{array}$$

and

$$\begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R := \begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R \quad \forall \begin{array}{l}\text{terms } R \text{ which} \\ \text{do not contain an } \mathtt{intro}_{(\forall a.\,\tau)}.\end{array}$$

6.4.9. Lemma (substitution lemma). With τ, ϱ, P, Q like in Definition 6.4.8 holds:

$$\frac{\Theta, \Gamma \vdash R : \pi}{\Theta, \Gamma \vdash \begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R : [\varrho/\boldsymbol{\mu}a.\,\tau]\,\pi}.$$

Proof. Induction on the recursive definition in Definition 6.4.8. ∎

6.4.10. Corollary. With τ, ϱ, P, Q like in Definition 6.4.8 holds:

$$\frac{R : \boldsymbol{\mu}a.\,\tau \underset{\boldsymbol{\Lambda}}{\longleftarrow} (\forall a.\,\tau)(\boldsymbol{\mu}a.\,\tau)}{\begin{bmatrix} \varrho & / & \boldsymbol{\mu}a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R : \varrho \underset{\boldsymbol{\Lambda}}{\longleftarrow} (\forall a.\,\tau)\varrho}$$

$[///]\ ()$	$:=\ ()$	constant
$[///]\ x$	$:=\ x$	variable
$[///]\ (\lambda x\colon \pi.\ R)$	$:=\ \lambda x\colon\ [\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi.\ [///]\,R$	abstraction
$[///]\ (\mu x\colon \pi.\ R)$	$:=\ \mu x\colon\ [\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi.\ [///]\,R$	fixed point
$[///]\ (\mathtt{inl}_\pi\, R)$	$:=\ \mathtt{inl}_{[\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi}([///]\,R)$	injection to the left
$[///]\ (\mathtt{inr}_\pi\, R)$	$:=\ \mathtt{inr}_{[\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi}([///]\,R)$	injection to the right
$[///]\ (\Lambda a.\, R)$	$:=\ \Lambda a.\ [///]\,R$	type-abstraction
$[///]\ R_\pi$	$:=\ ([///]\,R)_{[\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi}$	type-application
$[///]\ (\mathtt{intro}_{(\forall a.\,\pi)}\, R)$	$:= \begin{cases} P([///]\,R) & : \pi = \tau, \\ \mathtt{intro}_{(\forall a.\ [\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi)}([///]\,R) & : \pi \neq \tau \end{cases}$	constructor
$[///]\ (\mathtt{elim}_{(\forall a.\,\pi)}\, R)$	$:= \begin{cases} Q([///]\,R) & : \pi = \tau, \\ \mathtt{elim}_{(\forall a.\ [\varrho/\boldsymbol{\mu} a.\,\tau]\,\pi)}([///]\,R) & : \pi \neq \tau \end{cases}$	destructor

Table 3: Definition of the con-& destructor-substitution.

and

$$\frac{R : \big(\forall a.\,\tau\big)\big(\boldsymbol{\mu} a.\,\tau\big) \underset{\boldsymbol{\Lambda}}{\longleftarrow} \boldsymbol{\mu} a.\,\tau}{\begin{bmatrix} \varrho & / & \boldsymbol{\mu} a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R : (\forall a.\,\tau)\varrho \underset{\boldsymbol{\Lambda}}{\longleftarrow} \varrho}.$$

6.4.11. Lemma. Let $R \in \mathit{Prog}\Big(\big(\forall a.\,\tau\big)\big(\boldsymbol{\mu} a.\,\tau\big)\Big)^{\boldsymbol{\mu} a.\,\tau}$ where R does not contain an $\mathtt{elim}_{\forall a.\,\tau}$. By virtue of

$$\mathsf{H}\varphi := \begin{cases} \mathsf{S}\Big(\begin{bmatrix} \varrho & / & \boldsymbol{\mu} a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \end{bmatrix} R\Big) & : \exists\, P \in \mathit{Prog} ::\ \mathsf{S}P = \varphi \\ \bot_{|\varphi|_{\mathsf{G}}, \mathsf{F}|\varphi|_{\mathsf{G}}} & : \text{otherwise} \end{cases} \qquad \forall\, \varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{G})$$

and

$$\mathsf{H}\big(\varphi, f, \varphi'\big) := \big(\mathsf{H}\varphi, f, \mathsf{H}\varphi'\big) \quad \forall\, (\varphi, f, \varphi') \in \mathrm{Mor}(\boldsymbol{\mathcal{A}lg}\,\mathsf{G})$$

the substitution operator from Definition 6.4.8 yields a functor

$$\mathsf{H} : \boldsymbol{\mathcal{A}lg}\,\mathsf{F} \leftarrow \boldsymbol{\mathcal{A}lg}\,\mathsf{G}$$

with

$$|\centerdot|_{\mathsf{F}} \cdot \mathsf{H} = |\centerdot|_{\mathsf{G}}.$$

Proof. By means of Corollary 6.4.10 we get $\mathsf{H} : \mathrm{Ob}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{F}) \leftarrow \mathrm{Ob}(\mathcal{A}\boldsymbol{lg}_{\mathcal{C}}\,\mathsf{G})$. We show, again using Corollary 6.4.10,

$$\frac{f \cdot \varphi' = \varphi \cdot \mathsf{G}f}{f \cdot \mathsf{H}\varphi' = \mathsf{H}\varphi \cdot \mathsf{F}f}$$

by induction on the recursive definition of the substitution operator in Definition 6.4.8. The proposition follows by Lemma 4.4.1. ∎

6.4.12. Corollary (acid rain theorem (λ-calculus version)). Let $(\forall a.\,\sigma), (\forall a.\,\tau) \in \mathit{PType}$ and $P, Q, R, S \in \mathit{Prog}$. Let R contain no $\mathtt{elim}_{(\forall a.\,\tau)}$ and let S contain no $\mathtt{intro}_{(\forall a.\,\tau)}$. The following equations hold (provided that they are well-typed):

$$\begin{aligned}
(\!|P|\!)_{(\forall a.\,\tau)} \cdot [\![Q]\!]_{(\forall a.\,\tau)} &\equiv^{cat} [\![P,\, Q]\!]_{(\forall a.\,\tau)} \\
(\!|P|\!)_{(\forall a.\,\tau)} \cdot [\![R,\, Q]\!]_{(\forall a.\,\sigma)} &\equiv^{cat} [\![\begin{bmatrix} \varrho & / & \mu a.\,\tau \\ P & / & \mathtt{intro}_{(\forall a.\,\tau)} \end{bmatrix} R ,\, Q]\!]_{(\forall a.\,\sigma)} \\
[\![P,\, S]\!]_{(\forall a.\,\sigma)} \cdot [\![Q]\!]_{(\forall a.\,\tau)} &\equiv^{cat} [\![P,\, \begin{bmatrix} \varrho & / & \mu a.\,\tau \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} S]\!]_{(\forall a.\,\sigma)}
\end{aligned}$$

Proof. The propositions directly follow by Corollary 6.4.5 and the fact that S is a functor (*cf.* Definition 6.3.9) from the 'hylo-version' of the acid rain theorem 5.4.11, using the functor from Lemma 6.4.11. Additionally, we have to use the functor $\mathsf{H} : \mathcal{C}o\mathcal{A}\boldsymbol{lg}\,\mathsf{F} \leftarrow \mathcal{C}o\mathcal{A}\boldsymbol{lg}\,\mathsf{G}$ (dually to Lemma 6.4.11) given by

$$\mathsf{H}\psi := \begin{cases} \mathsf{S}(\begin{bmatrix} \varrho & / & \mu a.\,\tau \\ Q & / & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} S) & : \exists\, Q \in \mathit{Prog} :: \ \mathsf{S}Q = \psi \\ \bot_{\mathsf{F}(|\psi|_{\mathsf{G}}),|\psi|_{\mathsf{G}}} & : \text{otherwise} \end{cases} \qquad \forall\, \psi \in \mathrm{Ob}(\mathcal{C}o\mathcal{A}\boldsymbol{lg}\,\mathsf{G})$$

with $|\centerdot|_{\mathsf{F}} \cdot \mathsf{H} = |\centerdot|_{\mathsf{G}}$. ∎

6.4.13. Example. Unfortunately, it is rather complex, to calculate major examples in our language, as we have seen in Note 6.1.8 and Example 6.4.6 treating the simple function *dup*. We can shorten the calculations, if we use the images under S in $\mathcal{C}$ directly and apply Corollary 6.4.12 by simply substituting *in* and *out* instead of `intro` and `elim`, respectively. Unfortunately this loses the nature of Corollary 6.4.12 being a formal transformation of terms.

Let *dup*, *Zero* and *Succ* be the same $\mathcal{C}$-morphisms and F be the same polynomial functor as in Example 6.4.6. We found the representation:

$$\mathit{dup} = (\!|\, [\mathit{Zero}, \mathit{Succ} \cdot \mathit{Succ}] \,|\!)_{\mathsf{F}}$$

Now we try to find a single catamorphism for the composition $dup \cdot dup$:

$$
\begin{array}{ll}
& dup \cdot dup \\
= & \quad \{ \text{ see above representation } \} \\
& (\![\, [Zero, Succ \cdot Succ] \,]\!)_{\mathsf{F}} \;\cdot\; (\![\, [Zero, Succ \cdot Succ] \,]\!)_{\mathsf{F}} \\
= & \quad \{ \text{ def. } Zero = in_{\mathsf{F}} \cdot \grave{\imath} \text{ and } Succ = in_{\mathsf{F}} \cdot \acute{\imath} \ \} \\
& (\![\, [Zero, Succ \cdot Succ] \,]\!)_{\mathsf{F}} \;\cdot\; (\![\, [in_{\mathsf{F}} \cdot \grave{\imath}, in_{\mathsf{F}} \cdot \acute{\imath} \cdot in_{\mathsf{F}} \cdot \acute{\imath}] \,]\!)_{\mathsf{F}} \\
= & \quad \{ \text{ substitute } in_{\mathsf{F}} \text{ by } [Zero, Succ \cdot Succ] \text{ according to Corollary 6.4.12 or Theorem 5.4.11 } \} \\
& (\![\, [[Zero, Succ \cdot Succ] \cdot \grave{\imath}, [Zero, Succ \cdot Succ] \cdot \acute{\imath} \cdot [Zero, Succ \cdot Succ] \cdot \acute{\imath}] \,]\!)_{\mathsf{F}} \\
= & \quad \{ \ [\,\centerdot\,, \centerdot\,]\text{-cancelation } \} \\
& (\![\, [Zero, Succ \cdot Succ \cdot Succ \cdot Succ] \,]\!)_{\mathsf{F}}.
\end{array}
$$

The last expression retranslated into Haskell yields:

```
dupdup Zero   = Zero
dupdup Succ n = Succ $ Succ $ Succ $ Succ $ dupdup n
```

and the following equation holds:

$$\texttt{dupdup} \equiv^{cat} \texttt{dup . dup}$$

6.4.14. Example. We show an example, which was used in [Küh99] to demonstrate the differences between deforestation and some composition techniques from the theory of tree transducers [FV98]. We will need an instantiation of the first fusion law in Theorem 4.3.5:

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}\,\mathsf{F}, |\centerdot|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}lg}\,\mathsf{G}, |\centerdot|_{\mathsf{G}} \Leftarrow \mathsf{K}_B)}{((\![\varphi]\!)_{\mathsf{G}} \Leftarrow id_B) \cdot (\![\mathsf{H} in_{\mathsf{G}}]\!)_{\mathsf{F}} = (\![\mathsf{H}\varphi]\!)_{\mathsf{F}}},$$

where $B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$; $\mathsf{F}, \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol}\,\boldsymbol{\mathcal{C}})$ and $\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}\,\mathsf{G})$. We consider the following Haskell program:

```
data L = A L | B L | N

ex         :: L -> L
ex (A x1) =  B (ex x1)
ex (B x1) =  A (ex x1)
ex N      =  N

mir           :: L -> L -> L
mir (A x1) y1 =  A (mir x1 (A y1))
mir (B x1) y1 =  B (mir x1 (B y1))
mir N      y1 =  y1

exmir       :: L -> L -> L
exmir z0 z1 =  mir (ex z0) z1

mirex       :: L -> L -> L
mirex z0 z1 =  ex (mir z0 z1)
```

We want to deforest the two functions `exmir` and `mirex`. First we map the functions into the category $\boldsymbol{\mathcal{C}}$ using our semantics functor S

$$L := \mathsf{S}(\texttt{L}) \qquad ex := \mathsf{S}(\texttt{ex}) \qquad mir := \mathsf{S}(\texttt{mir})$$

$$exmir := \mathsf{S}(\texttt{exmir}) \qquad mirex := \mathsf{S}(\texttt{mirex})$$

and calculate the representations of `ex` and `mir` as catamorphisms (*cf.* Example 6.4.6):

$$\begin{aligned} ex &= (\![[b, a, n]]\!)_{\mathsf{F}} \\ mir &= (\![[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \end{aligned}$$

where we use the functor

$$\mathsf{F} := \mathsf{Id} + \mathsf{Id} + \mathsf{K}_1$$

and the abbreviations

$$\iota_1 := \grave{\iota} \cdot \grave{\iota} \qquad \iota_2 := \grave{\iota} \cdot \acute{\iota} \qquad \iota_3 = \acute{\iota}$$

and

$$a := in_{\mathsf{F}} \cdot \iota_1 \qquad b := in_{\mathsf{F}} \cdot \iota_2 \qquad n := in_{\mathsf{F}} \cdot \iota_3$$

such that

$$\mathsf{TF} = L \quad \text{and} \quad a = \mathsf{S}A \qquad b = \mathsf{S}B \qquad n = \mathsf{S}N.$$

We start with the easier function `exmir`:

$$\begin{aligned} exmir &= mir \cdot ex \\ &= (\![[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \cdot (\![[b, a, n]]\!)_{\mathsf{F}} \\ &= (\![[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \cdot (\![[in_{\mathsf{F}} \cdot \iota_2, in_{\mathsf{F}} \cdot \iota_1, in_{\mathsf{F}} \cdot \iota_3]]\!)_{\mathsf{F}} \\ &= (\![[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \cdot (\!|\mathsf{H} in_{\mathsf{F}}|\!)_{\mathsf{F}} \quad \text{where} \quad \mathsf{H}\varphi = [\varphi \cdot \iota_2, \varphi \cdot \iota_1, \varphi \cdot \iota_3] \quad \forall\, \varphi \\ &= (\!|\mathsf{H}[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}]|\!)_{\mathsf{F}} \\ &= (\![[[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}] \cdot \iota_2, [a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}] \cdot \iota_1, [a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}] \cdot \iota_3]]\!)_{\mathsf{F}} \\ &= (\![[b \Leftarrow b, a \Leftarrow a, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \end{aligned}$$

Now we try the function `mirex`, which, according to [Küh99], may not be treated satisfactorily by deforestation:

$$\begin{aligned} mirex &= (ex \Leftarrow id) \cdot mir \\ &= \big((\![[b, a, n]]\!)_{\mathsf{F}} \Leftarrow id\big) \cdot (\![[a \Leftarrow a, b \Leftarrow b, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \\ &= \big((\![[b, a, n]]\!)_{\mathsf{F}} \Leftarrow id\big) \cdot (\![[in_{\mathsf{F}} \cdot \iota_1 \Leftarrow in_{\mathsf{F}} \cdot \iota_1, in_{\mathsf{F}} \cdot \iota_2 \Leftarrow in_{\mathsf{F}} \cdot \iota_2, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \\ &= \big((\![[b, a, n]]\!)_{\mathsf{F}} \Leftarrow id\big) \cdot (\!|\mathsf{H} in_{\mathsf{F}}|\!)_{\mathsf{F}} \quad \text{where} \quad \mathsf{H}\varphi = [\varphi \cdot \iota_1 \Leftarrow \varphi \cdot \iota_1, \varphi \cdot \iota_2 \Leftarrow \varphi \cdot \iota_2, \text{curry}\,\acute{\pi}] \forall\, \varphi \\ &= (\!|\mathsf{H}[b, a, n]|\!)_{\mathsf{F}} \\ &= (\![[[b, a, n] \cdot \iota_1 \Leftarrow [b, a, n] \cdot \iota_1, [b, a, n] \cdot \iota_2 \Leftarrow [b, a, n] \cdot \iota_2, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \\ &= (\![[b \Leftarrow b, a \Leftarrow a, \text{curry}\,\acute{\pi}]]\!)_{\mathsf{F}} \end{aligned}$$

Thus $exmir = mirex$ and the above resulting catamorphism retranscripted into Haskell yields

```
mirex (A x1) y1 =  B (mir x1 (B y1))
mirex (B x1) y1 =  A (mir x1 (A y1))
mirex N      y1 =  y1
```

which is the same result that is obtained by classical deforestation for `exmir` but not for `mirex`.

7 Future work

7.1 Involutory categories

The idea of deriving the functor from Lemma 6.4.11, which is used in the acid rain theorem 5.4.11, unfortunately leads to the undesirable conditions: '... contains no $\mathtt{elim}_{(\forall a.\,\tau)}$.' or '... contains no $\mathtt{intro}_{(\forall a.\,\tau)}$.' But we can state a special case of the 'acid rain theorem', in which we can work around these conditions. Notice the analogies of the category $\boldsymbol{\mathcal{A}}$, which will be used in the following proof, with the involutory category $\breve{\boldsymbol{\mathcal{C}}}$ in Definition 3.3.3.

7.1.1. Theorem (variant acid rain theorem). Let $(\forall a.\,\sigma), (\forall a.\,\tau) \in \mathit{PType}$, $\varrho \in \mathit{Type}$ and

$$\varrho \xleftarrow[\Lambda]{P} (\forall a.\,\tau)\varrho \xleftarrow[\Lambda]{Q} \varrho$$

as well as

$$\boldsymbol{\mu} a.\,\tau \xleftarrow[\Lambda]{R} (\forall a.\,\sigma)(\boldsymbol{\mu} a.\,\tau) \xleftarrow[\Lambda]{S} \boldsymbol{\mu} a.\,\tau$$

but this time *without* any constrictions. The following holds:

$$(\!|P|\!)_{(\forall a.\,\tau)} \cdot [\![\, R,\, S \,]\!]_{(\forall a.\,\sigma)} \cdot [\![Q]\!]_{(\forall a.\,\tau)} \equiv^{cat} [\![\, \begin{bmatrix} \varrho \;/ & \boldsymbol{\mu} a.\,\tau \\ P \;/ & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q \;/ & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} R\,,\; \begin{bmatrix} \varrho \;/ & \boldsymbol{\mu} a.\,\tau \\ P \;/ & \mathtt{intro}_{(\forall a.\,\tau)} \\ Q \;/ & \mathtt{elim}_{(\forall a.\,\tau)} \end{bmatrix} S \,]\!]_{(\forall a.\,\sigma)}$$

Proof. For $\mathsf{F} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{C}}$ we define, analogically to the definition of $\boldsymbol{\mathcal{A}lg}\,\mathsf{F}$ (*cf.* Definition 2.2.33) and $\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F}$ (*cf.* Definition 2.2.33), the category $\boldsymbol{\mathcal{A}}\mathsf{F}$, whose objects are specific pairs of $\boldsymbol{\mathcal{C}}$-morphisms

$$A \xleftarrow[\boldsymbol{\mathcal{C}}_\perp]{\varphi} \mathsf{F}A \xleftarrow[\boldsymbol{\mathcal{C}}]{\psi} A$$

where $A \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$. Even though this is different from the category $\boldsymbol{\mathcal{A}lg}\,\mathsf{F} \times \boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F}$, we may introduce the projection functors in the obvious way:

$$\grave{\mathsf{P}} : \boldsymbol{\mathcal{A}lg}\,\mathsf{F} \leftarrow \boldsymbol{\mathcal{A}}\mathsf{F} \qquad \text{and} \qquad \acute{\mathsf{P}} : \boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{F} \leftarrow \boldsymbol{\mathcal{A}}\mathsf{F}.$$

Furthermore we may define the forgetful functor by

$$|\centerdot|_{\mathsf{F}} : \boldsymbol{\mathcal{C}} \leftarrow \boldsymbol{\mathcal{A}}\mathsf{F},$$

which obviously satisfies $|\centerdot|_{\mathsf{F}} = |\centerdot|_{\mathsf{F}} \cdot \grave{\mathsf{P}} = |\centerdot|_{\mathsf{F}} \cdot \acute{\mathsf{P}}$, where we used the same notation $|\centerdot|_{\mathsf{F}}$ for the three different forgetful functors. Since, according to Theorem 4.1.1, cata- and anamorphisms are natural transformations

$$(\!|\centerdot|\!)_{\mathsf{F}} = \big((\!|\varphi|\!)_{\mathsf{F}}\big)_{\varphi \in \mathrm{Ob}(\boldsymbol{\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F})} : |\centerdot|_{\mathsf{F}} \dot{\leftarrow} \mathsf{K}_{\mathsf{TF}} \cdot |\centerdot|_{\mathsf{F}}$$

$$[\![\centerdot]\!]_{\mathsf{F}} = \big([\![\psi]\!]_{\mathsf{F}}\big)_{\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}_{\boldsymbol{\mathcal{C}}}\,\mathsf{F})} : \mathsf{K}_{\mathsf{TF}} \cdot |\centerdot|_{\mathsf{F}} \dot{\leftarrow} |\centerdot|_{\mathsf{F}}$$

we have

$$\big((\!|\grave{\mathsf{P}}(\varphi,\psi)|\!)_{\mathsf{F}}\big)_{(\varphi,\psi) \in \mathrm{Ob}(\boldsymbol{\mathcal{A}}\mathsf{F})} : \underbrace{|\centerdot|_{\mathsf{F}} \cdot \grave{\mathsf{P}}}_{=|\centerdot|_{\mathsf{F}}} \dot{\leftarrow} \underbrace{\mathsf{K}_{\mathsf{TF}} \cdot |\centerdot|_{\mathsf{F}} \cdot \grave{\mathsf{P}}}_{=\mathsf{K}_{\mathsf{TF}}}$$

$$\big([\![\acute{\mathsf{P}}(\varphi,\psi)]\!]_{\mathsf{F}}\big)_{(\varphi,\psi) \in \mathrm{Ob}(\boldsymbol{\mathcal{A}}\mathsf{F})} : \underbrace{\mathsf{K}_{\mathsf{TF}} \cdot |\centerdot|_{\mathsf{F}} \cdot \acute{\mathsf{P}}}_{=\mathsf{K}_{\mathsf{TF}}} \dot{\leftarrow} \underbrace{|\centerdot|_{\mathsf{F}} \cdot \acute{\mathsf{P}}}_{=|\centerdot|_{\mathsf{F}}}$$

and thus the hylomorphism is a natural transformation as the composition of the above (*cf.* Corollary 5.4.8):

$$\big([\![\, \varphi,\, \psi \,]\!]_{\mathsf{F}}\big)_{(\varphi,\psi)} : |\,\boldsymbol{.}\,|_{\mathsf{F}} \dot{\leftarrow} |\,\boldsymbol{.}\,|_{\mathsf{F}}.$$

For a functor $\mathsf{H} : \boldsymbol{\mathcal{A}}\mathsf{F} \leftarrow \boldsymbol{\mathcal{A}}\mathsf{G}$ we introduce the abbreviation

$$[\![\, \mathsf{H}(\varphi,\, \psi) \,]\!]_{\mathsf{F}} := [\![\, (\grave{\mathsf{P}} \cdot \mathsf{H})(\varphi,\psi),\ (\acute{\mathsf{P}} \cdot \mathsf{H})(\varphi,\psi) \,]\!]_{\mathsf{F}} \quad \forall\, (\varphi,\psi) \in \mathrm{Ob}(\boldsymbol{\mathcal{A}}\mathsf{G}).$$

By means of Theorem 4.3.5 (once ordinary and once again dually) we find, analogically to Theorem 5.4.11, the rule:

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}}\mathsf{F}, |\,\boldsymbol{.}\,|_{\mathsf{F}}) \leftarrow (\boldsymbol{\mathcal{A}}\mathsf{G}, |\,\boldsymbol{.}\,|_{\mathsf{G}})}{(\!|\varphi|\!)_{\mathsf{G}} \cdot [\![\, \mathsf{H}(in_{\mathsf{G}},\, out_{\mathsf{G}}) \,]\!]_{\mathsf{F}} \cdot [\!(\psi)\!]_{\mathsf{G}} = [\![\, \mathsf{H}(\varphi,\, \psi) \,]\!]_{\mathsf{F}}}$$

Like in the proof of Lemma 6.4.11 we derive the functor $\mathsf{H} : \boldsymbol{\mathcal{A}}\mathsf{F} \leftarrow \boldsymbol{\mathcal{A}}\mathsf{G}$ in virtue of

$$\mathsf{H}(\varphi,\psi) := \begin{cases} \left(\mathsf{S}(\begin{bmatrix} \varrho & / & \boldsymbol{\mu} a.\, \tau \\ P & / & \mathtt{intro}_{(\forall a.\, \tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\, \tau)} \end{bmatrix} R), \mathsf{S}(\begin{bmatrix} \varrho & / & \boldsymbol{\mu} a.\, \tau \\ P & / & \mathtt{intro}_{(\forall a.\, \tau)} \\ Q & / & \mathtt{elim}_{(\forall a.\, \tau)} \end{bmatrix} S)\right) & : \exists\, P, Q \in \mathit{Prog} ::\ \mathsf{S}P = \varphi \ \wedge\ \mathsf{S}Q = \psi \\ (\perp_{|\varphi|_{\mathsf{G}}, \mathsf{F}|\varphi|_{\mathsf{G}}}, \perp_{\mathsf{F}|\psi|_{\mathsf{G}}, |\psi|_{\mathsf{G}}}) & : \text{otherwise} \qquad \forall\, (\varphi,\psi) \in \mathrm{Ob}(\boldsymbol{\mathcal{A}}\mathsf{G}) \end{cases}$$

and $\forall\, \big((\varphi, f, \varphi'), (\psi, g, \psi')\big) \in \mathrm{Mor}(\boldsymbol{\mathcal{A}}\mathsf{G})$

$$\mathsf{H}\big((\varphi, f, \varphi'), (\psi, g, \psi')\big) := \Big(\big((\grave{\mathsf{P}} \cdot \mathsf{H})(\varphi,\psi), f, (\grave{\mathsf{P}} \cdot \mathsf{H})(\varphi',\varphi')\big), \big((\acute{\mathsf{P}} \cdot \mathsf{H})(\varphi,\psi), g, (\acute{\mathsf{P}} \cdot \mathsf{H})(\varphi',\psi')\big)\Big).$$

The proposition follows similar to the proof of Corollary 6.4.12. ■

7.2 Functorial acid rain theorem

The functorial acid rain theorem 4.3.4 inspired us to the following: We want to develop an algorithm that, from a given recursive definition in our language, computes the concrete functors, which we called H in Corollary 4.3.2 and 4.3.4 and Theorem 4.3.5. This concrete functor H can be viewed as an 'abstraction from the constructor' [LS95], where $(\!|\mathsf{H} in_{\mathsf{G}}|\!)_{\mathsf{F}} = \mathtt{build}^{\mu\mathsf{G}} (\!|\mathsf{H}\,\boldsymbol{.}|\!)_{\mathsf{F}}$. The algorithm would be a generalization of the algorithm presented in [LS95] for our extended typed λ-calculus. As a first step, we started to characterize (Lemma 4.4.1) and to construct (Corollary 4.4.3) such concrete functors. In particular we want to investigate tree transducers [FV98], which can be viewed to some extent as terms of a λ-calculus with a specific syntax. The reason is, that we want to compare existing composition results for tree transducers with the fusion-rules presented in this paper.

7.3 Contravariant forgetful functors

Corollary 4.3.1 even permits to derive versions of Theorem 4.3.5 where a forgetful functor is contravariant, *e.g.*

$$\frac{\mathsf{H} : (\boldsymbol{\mathcal{A}lg}\,\mathsf{F}, |\,\boldsymbol{.}\,|_{\mathsf{F}}) \leftarrow \big((\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{G})^{\mathrm{op}}, \mathsf{K}_B \Leftarrow |\,\boldsymbol{.}\,|_G^{\mathrm{op}}\big)}{(id_B \Leftarrow [\!(\psi)\!]_{\mathsf{G}}) \ \cdot\ (\!|\mathsf{H}\, out_{\mathsf{G}}|\!)_{\mathsf{F}} \ =\ (\!|\mathsf{H}\,\psi|\!)_{\mathsf{F}}}$$

where $\mathsf{F}, \mathsf{G} \in \mathrm{Ob}(\boldsymbol{Pol\,\mathcal{C}})$; $\psi \in \mathrm{Ob}(\boldsymbol{\mathcal{C}o\mathcal{A}lg}\,\mathsf{G})$ and $B \in \mathrm{Ob}\,\boldsymbol{\mathcal{C}}$. The concrete functor H has the desired form to fuse two functions with accumulator parameters (*e.g.* `mir` in Example 6.4.14). We believe that it is possible to combine the above instance of the generalized acid rain theorem with the instance from Example 6.4.14. This would permit to fuse functions with accumulator parameters in a sufficient way, which would be a real improvement to the current possibilities of hylomorphism fusion.

Acknowledgments

I would like to thank Olaf Chitil[12] who co-supervised my masters thesis [Jür99], which is the basis for this work, for his help and patience during our discussions and Heiko Vogler[13] for giving me the time and possibility to write this report.

References

[AHS90] Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories.* Pure and Applied Mathematics. John Wiley & Sons, 1990.

[BdM97] Richard Bird and Oege de Moor. *Algebra of Programming.* International Series in Computer Science. Prentice Hall, 1997.

[Bra97] Torben Braüner. A simple adequate categorical model for PCF. In *Proceedings of the Third Conference in Typed Lambda Calculi and Applications (TLCA '97)*, volume 1210 of *Lecture Notes in Computer Science*, pages 82–98, Nancy, France, April 1997. Springer.

[dB89] Peter J. de Bruin. Naturalness of polymorphism. Technical Report CS 8916, Rijksuniversiteit Groningen, The Netherlands, 1989.

[Fel78] W. Felscher. *Naive Mengen und abstrakte Zahlen*, volume I. B. I. Wissenschaftsverlag, 1978.

[FP94] M. P. Fiore and G. D. Plotkin. An axiomatisation of computationally adequate domain theoretic models of FPC. In *Proceedings of the Ninth Annual (IEEE) Symposium on Logic in Computer Science*, pages 92–102, April 1994.

[FV98] Zoltán Fülöp and Heiko Vogler. *Syntax-directed semantics — Formal models based on tree transducers.* Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1998.

[GLP93] Andrew Gill, John Launchburry, and Simon L. Peyton Jones. A short cut to deforestation. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '93)*, pages 223–232, Copenhagen, Denmark, June 1993. ACM Press.

[Gun92] C. A. Gunter. *Semantics of Programming Languages.* The MIT Press, 1992.

[HIT96] Zhenjiang Hu, Hideya Iwasaki, and Masato Takeichi. Deriving structural hylomorphisms from recursive definitions. In *Proceedings of the 1st International Conference on Functional Programming*, pages 73–82, Philadelphia, PA, May 1996. ACM Press.

[Hud89] Paul Hudak. Conceptions, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, September 1989.

[Hug89] John Hughes. Why functional programming matters. *The Computer Journal*, 32(2):98–107, April 1989.

[Jür99] Claus Jürgensen. Kategorientheoretisch fundierte Programmtransformationen für Haskell. Diplomarbeit in Mathematik, Lehrstuhl für Informatik II, RWTH-Aachen, Germany, October 1999.

[Küh99] Armin Kühnemann. Comparison of deforestation techniques for functional programs and for tree transducers. In A. Middeldorp and T. Sato, editors, *Proceedings of the 4th Fuji International Symposium on Functional and Logic Programming (FLOPS'99)*, volume 1722 of *LNCS*, pages 114–130, Tsukuba, Japan, November 1999. Springer-Verlag.

[12] `http://www-users.cs.york.ac.uk/~olaf/`

[13] `http://www.orchid.inf.tu-dresden.de/gdp/vogler.html`

[LS95] John Launchburry and Tim Sheard. Warm fusion: Deriving build-catas from recursive definitions. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '95)*, pages 314–323, La Jolla, San Diego, CA, USA, June 1995. ACM Press.

[Mac72] S. Mac Lane. *Kategorien*. Springer-Verlag, Berlin · Heidelberg, Germany, 1972.

[PH99] Simon L. Peyton Jones and John Hughes, editors. *Haskell 98: A Non-strict, Purely Functional Language*. `http://haskell.org/onlinereport/`, 1999.

[SF93] Tim Sheard and Leo Fegaras. A fold for all seasons. In *Proceedings of Functional Programming Languages an Computer Architecture (FPCA '93)*, pages 233–242, Copenhagen, Denmark, June 1993. ACM Press.

[TM95] Akihiko Takano and Erik Meijer. Shortcut deforestation in calculational form. In *Proceedings of the Conference on Functional Programing Languages and Computer Architecture*, pages 306–313, La Jolla, CA, June 1995. ACM Press.

[Tüf98] Christian Tüffers. Erweiterung des Glasgow-Haskell-Compilers um die Erkennung von Hylomorphismen. Diplomarbeit in Informatik, Lehrstuhl für Informatik II, RWTH-Aachen, Germany, 1998.

[Wad89] Philip Wadler. Theorems for free! In *The 4th International Conference on Functional Programming Languages and Computer Architecture (FPCA '89)*, pages 347–359, London, September 1989. Imperial College, ACM Press.

[Wad90] Philip Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73(2):231–248, 1990.

Index