Algebra of Programming
Lecture notes

Prof. Dr. Stefan Milius

Leon Vatthauer

March 29, 2024

Contents

1 Introduction **3**
1.1 Functions . . . 3
1.2 Data Types . . . 3
1.2.1 Natural Numbers . . . 4
1.2.2 Lists . . . 6

2 Category Theory **7**
2.1 Special Objects . . . 7
2.2 Initial and Terminal Objects . . . 7
2.3 Special Morphisms . . . 8
2.4 Duality . . . 9
2.5 Products and Coproducts . . . 10
2.6 Functors . . . 12
2.7 Natural Transformations . . . 14
2.8 Functor Algebras . . . 16
2.8.1 Initial F-algebras . . . 18
2.8.2 Term Algebras . . . 19
2.8.3 Parametric Data Types . . . 19
2.9 Functor Coalgebras . . . 19
2.9.1 Corecursion and Coinduction . . . 22
2.10 Limits . . . 23
2.11 Colimits . . . 25

3 Constructions **27**
3.1 CPO . . . 27
3.2 Initial Algebra Construction . . . 27
3.3 Terminal Coalgebra Construction . . . 27

Bibliography **29**

1 Introduction

This is a summary of the course "Algebra des Programmierens" taught by Prof. Dr. Stefan Milius in the winter term 2023/2024 at the FAU [1]. The course is based on [2] with [1] as a reference for category theory.

Goal of the course is to develop a mathematical theory for semantics of data types and their accompanying proof principles. The chosen environment is the field of category theory.

1.1 Functions

A function $f : X \to Y$ is a mapping from the set X (the domain of f) to the set Y (the codomain of f). More concretely f is a relation $f \subseteq X \times Y$ which is

- *left-total*, i.e. for all $x \in X$ exists some $y \in Y$ such that $(x, y) \in f$;
- *right-unique*, i.e. any $(x, y), (x, y') \in f$ imply $y = y'$.

Often, one is also interested in the symmetrical properties, a function is called

- *injective* or *left-unique* if for every $x, x' \in X$ the implication $f(x) = f(x') \to x = x'$ holds;
- *surjective* or *right-total* if for every $y \in Y$ there exists an $x \in X$ such that $f(x) = y$;
- *bijective* if it is injective and surjective.

Example 1.1.1.

1. The identity function $id_A : A \to A$, $id_A(x) = x$
2. The constant function $b! : A \to B$ for $b \in B$ defined by $b!(x) = b$
3. The inclusion function $i_A : A \to B$ for $A \subseteq B$ defined by $i_A(x) = x$
4. Constants $b : 1 \to B$, where $1 := *$. The function b is in bijection with the set B.
5. Composition of function $f : A \to B, g : B \to C$ called $g \circ f : A \to C$ defined by $(g \circ f)(x) = g(f(x))$.
6. The empty function ¡ $: \emptyset \to B$
7. The singleton function $! : A \to 1$

1.2 Data Types

Programs work with data that should ideally be organized in a useful manner. A useful representation for data in functional programming is by means of *algebraic data types*. Some basic data types (written in Haskell notation) are

```haskell
data Bool = True | False
data Nat  = Zero | Succ Nat
```

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg

These data types are declared by means of constructors, yielding concrete descriptions how inhabitants of these types are created. *Parametric data types* are additionally parametrized by another data type, e.g.

```
data Maybe a    = Nothing | Just a
data Either a b = Left a  | Right b
data List a     = Nil     | Cons a (List a)
```

Such data types (parametric or non-parametric) usually come with a principle for defining functions called recursion and in richer type systems (e.g. in a dependently typed setting) with a principle for proving facts about recursive functions called induction. Equivalently, every function defined by recursion can be defined via a *fold*-function which satisfies an identity and fusion law, which replace the induction principle. Let us now consider two examples of data types and illustrate this.

1.2.1 Natural Numbers

The type of natural numbers comes with a fold function $foldn : C \to (Nat \to C) \to Nat \to C$ for every C, defined by

$$\begin{aligned} foldn\ c\ h\ zero &= c \\ foldn\ c\ h\ (suc\ n) &= h\ (foldn\ c\ h\ n) \end{aligned}$$

Example 1.2.1. Let us now consider some functions defined in terms of $foldn$.

- $iszero : Nat \to Bool$ is defined by

$$iszero = foldn\ true\ false!$$

- $plus : Nat \to Nat \to Nat$ is defined by

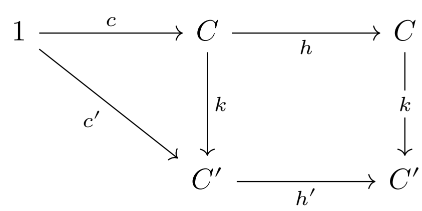
$$plus = foldn\ id\ (succ \circ eval)$$

  where $eval : (A \to B) \to A \to B$ is defined by

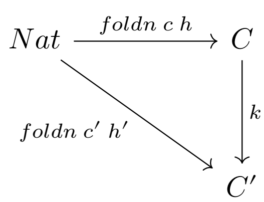
$$eval\ f\ a = f\ a$$

Proposition 1.2.2. *$foldn$ satisfies the following two rules*

1. ***Identity**: $foldn\ zero\ succ = id_{Nat}$*
2. ***Fusion**: for all $c : C$, $h, h' : Nat \to C$ and $k : C \to C'$ with $kc = c'$ and $kh = h'k$ follows $k \circ foldn\ c\ h = foldn\ c'\ h'$, or diagrammatically:*



implies



Proof. Both follow by induction over an argument $n : Nat$:

1. **Identity**:

   Case 1. $n = zero$

$$foldn\ zero\ succ\ zero = zero = id\ zero$$

   Case 2. $n = succ\ m$

$$\begin{aligned} foldn\ zero\ succ\ (succ\ m) &= succ(foldn\ zero\ succ\ m) \\ &= succ\ m && \text{(IH)} \\ &= id(succ\ m) \end{aligned}$$

2. **Fusion**:

   Case 1. $n = zero$

$$k(foldn\ c\ h\ zero) = k\ c = c' = foldn\ c'\ h'\ zero$$

   Case 2. $n = succ\ m$

$$\begin{aligned} k(foldn\ c\ h\ (succ\ m)) &= k(h(foldn\ c\ h\ m)) \\ &= h'(k(foldn\ c\ h\ m)) \\ &= h'(foldn\ c'\ h'\ m) && \text{(IH)} \\ &= foldn\ c'\ h'\ (succ\ m) \end{aligned}$$

□

Example 1.2.3. The identity and fusion laws can in turn be used to prove the following induction principle:

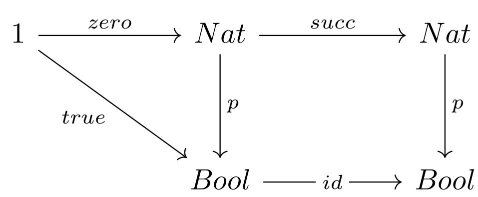
For any predicate $p : Nat \to Bool$,

1. $p\ zero = true$ and
2. $p \circ succ = p$

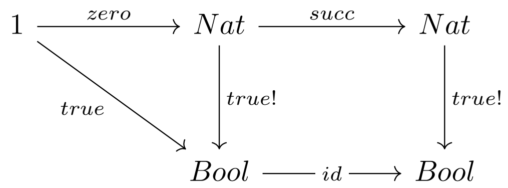
implies $p = true!$. This follows by

$$\begin{aligned} & p \\ = {} & p \circ (foldn\ zero\ succ) && (\textbf{Identity}) \\ = {} & foldn\ true\ id && (\textbf{Fusion}) \\ = {} & true! \circ (foldn\ zero\ succ) && (\textbf{Fusion}) \\ = {} & true!. && (\textbf{Identity}) \end{aligned}$$

Where the first application of **Fusion** is justified, since the diagram



commutes by the requisite properties of p, and the second application of **Fusion** is justified, since the diagram



trivially commutes.

1.2.2 Lists

We will now look at the *List* type and examine it for similar properties. Let us start with the fold function $foldr : C \to (A \to C \to C) \to List\ A \to C$, which is defined by

$$\begin{aligned} foldr\ c\ h\ nil &= c \\ foldr\ c\ h\ (cons\ x\ xs) &= h\ a\ (foldr\ c\ h\ xs) \end{aligned}$$

Example 1.2.4. Again, let us define some functions using $foldr$.

- $length : List\ A \to Nat$ is defined by

$$length = foldr\ zero\ (succ!)$$

- For $f : A \to B$ we can define *List*-mapping function $List\ f : List\ A \to List\ B$ by

$$List\ f = foldr\ nil\ (cons \circ f)$$

Proposition 1.2.5. *$foldr$ satisfies the following two rules*

1. ***Identity**: $foldr\ nil\ cons = id_{List\ A}$*
2. ***Fusion**: for all $c : C,\ h, h' : Nat$*

2 Category Theory

Categories consist of objects and morphisms between those objects, that can be composed in a coherent way. This yields a framework for abstraction of many mathematical concepts that enables us to reason on a very abstract level.

Definition 2.0.1 (Category)**.** A category $\mathscr{C}$ consists of

- a class of objects denoted $|\mathscr{C}|$,
- for every pair of objects $A, B \in |\mathscr{C}|$ a set of morphisms $\mathscr{C}(A, B)$ called the hom-set,
- a morphism $id_A : A \to A$ for every $A \in |\mathscr{C}|$
- a composition operator $(-) \circ (-) : \mathscr{C}(B, C) \to \mathscr{C}(A, B) \to \mathscr{C}(A, C)$ for every $A, B, C \in |\mathscr{C}|$

additionally the composition must be associative and $f \circ id_A = f = id_B \circ f$ for any $f : A \to B$.

Example 2.0.2. Some standard examples of categories and their objects and morphisms include:

Category	Objects	Morphisms
Set	Sets	Functions
Par	Sets	Partial functions
Rel	Sets	Binary relations
Gra	Directed Graphs	Graph homomorphisms
Pos	Partially ordered sets	Monotone mappings
Mon	Monoids	Monoid homomorphisms
Monoid $(M, \cdot, e)$	A single object $*$	$x : * \to *$ for every $x \in M$
Poset $(X, \leq)$	Elements of X	$x \leq y \iff \exists! f : x \to y$

2.1 Special Objects

Special objects play an important role in category theory. In this chapter we will characterize (finite) products and coproducts, as well as special morphisms such as isomorphisms, monomorphisms and epimorphisms.

2.2 Initial and Terminal Objects

Definition 2.2.1 (Initial and Terminal Objects)**.** The following is the categorical abstraction of "empty set" and "singleton set" respectively.

1. An object $0 \in |\mathsf{C}|$ is called initial if for every $B \in |C|$ there is a unique morphism $¡ : 0 \to B$.
2. An object $1 \in |\mathscr{C}|$ is called terminal if for every $A \in |C|$ there is a unique morphism $! : A \to 1$.

Example 2.2.2. Oftentimes the initial object is an empty structure and the terminal object a singleton structure, some examples are:

Category	Initial Object	Terminal Object
Set	$\emptyset$	$*$
Pos	$\emptyset$	$*$
Gra	Empty graph	Singleton graph
Poset $(X, \leq)$	$\bot \in X$ such that $\forall x \in X. \bot \leq x$	$\top \in X$ such that $\forall x \in X. x \leq \top$

2.3 Special Morphisms

Now let us characterize special morphisms.

Definition 2.3.1 (Special Morphisms)**.** Let $f : A \to B$ be a morphism. f is called

- an *isomorphism* (*iso*), if there exists an inverse $f^{-1} : B \to A$ such that $f \circ g = id_B$ and $g \circ f = id_A$;
- a *monomorphism* (*mono*), if for all $g, h : C \to A$ the implication $f \circ g = f \circ h \Rightarrow g = h$ holds;
- an *epimorphism* (*epi*), if for all $g, h : B \to C$ the implication $g \circ f = h \circ f \Rightarrow g = h$ holds.

Example 2.3.2. Let us consider what these notions instantiate to in concrete categories.

Category	Monomorphisms	Epimorphisms	Isomorphisms
Set	injective functions	surjective functions	bijective functions
Pos, Gra	injective morphisms	surjective morphisms	bijective morphisms
Poset $(X, \leq)$	all	all	all
Monoid $(M, \cdot, e)$	left cancellative $a \in M$	right cancellative $a \in M$	invertible $a \in M$

Proposition 2.3.3. *Every isomorphism is a monomorphism and an epimorphism.*

Proof. Let f be an isomorphism.

- $f \circ g = f \circ h$ implies $g = f^{-1} \circ f \circ g = f^{-1} \circ f \circ h = h$, thus f is a monomorphism.
- $g \circ f = h \circ f$ implies $g = g \circ f \circ f^{-1} = h \circ f \circ f^{-1} = h$, thus f is an epimorphism.

□

Proposition 2.3.4. *If $f \circ m$ is a monomorphism then m is also a monomorphism.*

Proof. Let $m \circ g = m \circ h$. To show that $g = h$ it suffices to show that $f \circ m \circ g = f \circ m \circ h$, which indeed follows by assumption. □

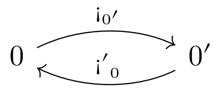
Proposition 2.3.5. *If $e \circ f$ is an epimorphism then e is also an epimorphism.*

Proof. Let $g \circ e = h \circ e$. To show that $g = h$ it suffices to show that $g \circ e \circ f = h \circ e \circ f$, which again follows by assumption. □

Categorical structures like initial objects are usually not uniquely identified, there might be multiple initial objects in a category. However, all initial objects in a category are isomorphic, we call this "unique up to isomorphism".

Proposition 2.3.6. *Initial objects are unique up to isomorphism.*

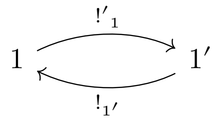
Proof. Let $0, 0' \in |\mathscr{C}|$ be two initial objects of $\mathscr{C}$ with the unique morphisms $¡_A : 0 \to A$ and $¡'_A : 0' \to A$. The isomorphism is:



Note that by uniqueness $¡_{0'} \circ ¡'_0 = ¡'_{0'} = id_{0'}$ and $¡'_0 \circ ¡_{0'} = ¡_0 = id_0$. □

Proposition 2.3.7. *Terminal objects are unique up to isomorphism.*

Proof. Let $1, 1' \in |\mathscr{C}|$ be two terminal objects of $\mathscr{C}$ with the unique morphisms $!_A : A \to 1$ and $!'_A : A \to 1'$. The isomorphism is:



Note that by uniqueness $!'_1 \circ !_{1'} = !'_{1'} = id_{1'}$ and $!_{1'} \circ !'_1 = !_1 = id_1$. □

2.4 Duality

Notice how similar the proofs of Proposition 2.3.4 and Proposition 2.3.5 as well as Proposition 2.3.6 and Proposition 2.3.7 are to each other. It seems that we should somehow be able to construct one proof from the other, such that the work required would be halved. This is actually the case, we can for example say that Proposition 2.3.5 follows from Proposition 2.3.4 by *duality*.

Definition 2.4.1 (Dual Category)**.** Every category $\mathscr{C}$ has a *dual category* $\mathscr{C}^{op}$ defined by

- $|\mathscr{C}^{op}| = |\mathscr{C}|$
- $\mathscr{C}^{op}(A, B) = \mathscr{C}(B, A)$

Example 2.4.2. Examples are:

1. In a poset the order relation gets reversed.
2. Rel^{op} is isomorphic to Rel, since subsets of $A \times B$ are in bijection with subsets of $B \times A$
3. $(\mathscr{C}^{op})^{op} = \mathscr{C}$

Every categorical notion can thus be dualized by viewing it in the dual category, some examples include:

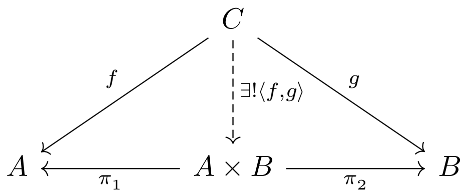
Notion	Dual Notion
Initial Object	Terminal Object
Monomorphism	Epimorphism
Isomorphism	Isomorphism

This yields a proof principle “by duality”, where every theorem yields another theorem by duality.

2.5 Products and Coproducts

The categorical abstraction of Cartesian products is:

Definition 2.5.1 (Product)**.** The *product* of two objects $A, B \in |\mathscr{C}|$ is an object that we call $A \times B$ together with morphisms $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ (the projections), where the following property holds:

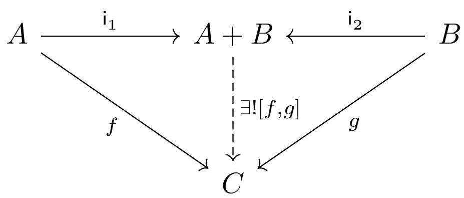


Example 2.5.2. Some examples include:

1. Set: The product of two sets A and B is the Cartesian product $A \times B = \{(a, b) \mid a \in A, b \in B\}$.
2. Gra: The product of two graphs has as vertices the Cartesian product of the vertices of both graphs and an edge $(v_1, u_1) \to (v_2, u_2)$ iff there exists edges $v_1 \to v_2$ and $u_1 \to u_2$.
3. Pos: Given two posets $(A, \leq), (B, \leq)$, the product is the Cartesian product of A and B where $(a, b) \leq (a', b') \iff a \leq a' \wedge b \leq b'$.
4. Let $(X, \leq)$ be a poset, the product of $a, b \in X$ is the greatest lower bound of a and b.

Dual to products are:

Definition 2.5.3 (Coproduct)**.** The *coproduct* of two objects $A, B \in |\mathscr{C}|$ is an object that we call $A + B$ together with morphisms $\mathsf{i}_1 : A \to A + B$ and $\mathsf{i}_2 : B \to A + B$ (the injections), where the following property holds:



Example 2.5.4. Examples include:

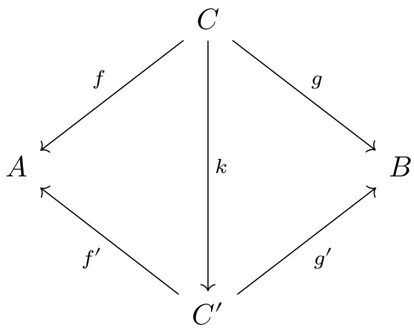
1. Set: The coproduct of two sets A and B is the disjoint union $A + B = \{(a, 0) \mid a \in A\} \cup \{(b, 1) | b \in B\}$.
2. Pos: The coproduct of ordered sets $(A, \leq)$ and $(B, \leq)$ is the disjoint union $A + B$ where $z \leq z'$ iff $z, z' \in A$ and $z \leq z'$ or $z, z' \in B$ and $z \leq z'$.
3. Gra: Analogous to Pos.
4. Let $(X, \leq)$ be a poset, the coproduct of $a, b \in X$ is the least upper bound of a and b.

5. Rel: Analogous to Set the coproduct is the disjoint union. Since $\mathsf{Rel} \cong \mathsf{Rel}^{op}$ we know that the product is also the disjoint union.

Proposition 2.5.5. *Products are unique up to isomorphism.*

Proof. The usual proof is somewhat analogous to the proof of Proposition 2.3.7. Instead, we will prove it like this:

Consider the category $span_{\mathscr{C}}(A, B)$ where objects are triples $A \xleftarrow{f} C \xrightarrow{g} B$ and morphisms $(C, f, g) \to (C', f', g')$ are morphisms $k : C \to C'$ in $\mathscr{C}$ such that



commutes. Products of A and B are the final objects in $span_{\mathscr{C}}(A, B)$ and are thus unique up to isomorphism. □

By duality, we get:

Proposition 2.5.6. *Coproducts are unique up to isomorphism.*

We can now characterize products (and later dually coproducts) as a commutative monoid:

Proposition 2.5.7. 1 *is a unit of* $\times$*, i.e.* $A \cong A \times 1$ *for any* $A \in |\mathscr{C}|$.

Proof. Take $\langle id_A, !_A \rangle : A \to A \times 1$ and $\pi_1 : A \times 1 \to A$, this indeed constitutes an isomorphism, since

$$\pi_1 \circ \langle id_A, !_A \rangle = id_A$$

by definition and

$$\langle id_A, !_A \rangle \circ \pi_1 = \langle \pi_1, !_A \rangle = \langle \pi_1, \pi_2 \rangle = id_{A \times 1},$$

because $\pi_2 = !_A : A \times 1 \to 1$ by uniqueness of $!_A$. □

Proposition 2.5.8. $\times$ *is associative, i.e.* $A \times (B \times C) \cong (A \times B) \times C$ *for any* $A, B, C \in |\mathscr{C}|$.

Proof. Take

$$\alpha = \langle \langle \pi_1, \pi_1 \circ \pi_2 \rangle, \pi_2 \circ \pi_2 \rangle : A \times (B \times C) \to (A \times B) \times C$$

and

$$\alpha^{-1} = \langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle : (A \times B) \times C \to A \times (B \times C).$$

The rest of the proof then amounts to simply rewriting. □

Proposition 2.5.9. $\times$ *is commutative, i.e.* $A \times B \cong B \times A$ *for any* $A, B \in |\mathscr{C}|$.

Proof. Take

$$\langle \pi_2, \pi_1 \rangle : A \times B \to B \times A$$

and

$$\langle \pi_2, \pi_1 \rangle : B \times A \to A \times B.$$

Indeed, $\langle \pi_2, \pi_1 \rangle \circ \langle \pi_2, \pi_1 \rangle = \langle \pi_2 \circ \langle \pi_2, \pi_1 \rangle, \pi_1 \circ \langle \pi_2, \pi_1 \rangle \rangle = \langle \pi_1, \pi_2 \rangle = id.$ □

Duality instantly yields the commutative monoid structure of coproducts:

Proposition 2.5.10. *0 is the unit of $+$, i.e. $A \cong A + 0$ for any $A \in |\mathscr{C}|$.*

Proposition 2.5.11. *$+$ is associative, i.e. $A + (B + C) \cong (A + B) + C$ for any $A, B, C \in |\mathscr{C}|$.*

Proposition 2.5.12. *$+$ is commutative, i.e. $A + B \cong B + A$ for any $A, B \in |\mathscr{C}|$.*

Remark 2.5.13. If a category has a terminal object and binary products one can form arbitrary n-ary products (finite products), such a category is called *Cartesian*. Dually a category with an initial object and binary coproducts is called *Cocartesian*.

2.6 Functors

Functors are morphisms between categories, concretely:

Definition 2.6.1 (Functor)**.** A functor $F : \mathscr{C} \to \mathscr{D}$ consists of

- a mapping $F : |\mathscr{C}| \to |\mathscr{D}|$ on objects and
- a mapping $F : \mathscr{C}(A, B) \to \mathscr{C}(FA, FB)$ on morphisms,

such that $F(id_A) = id_{FA}$ and $F(g \circ f) = Fg \circ Ff$.

Example 2.6.2. Usual examples of functors include

1. Constant functors mapping to a single object: $D! : \mathscr{C} \to \mathscr{D}, D \in |\mathscr{D}|$ with
$$D!(C) = D, \qquad D!(f) = id_D.$$
2. Identity functor: $\mathrm{Id}_{\mathscr{C}} : \mathscr{C} \to \mathscr{C}$ with
$$\mathrm{Id}_{\mathscr{C}}(C) = C, \qquad \mathrm{Id}_{\mathscr{C}}(f) = f.$$
3. Composition of functors: $(FG)(X) = F(GX), (FG)(f) = F(Gf)$
4. Square functor on Set: $Q : \mathsf{Set} \to \mathsf{Set}$ with
$$QX = X \times X, \qquad Qf = f \times f.$$
5. $list : Set \to Set$, see subsection 1.2.2.
6. For $A \in |\mathscr{C}|$ there is the hom-functor $\mathscr{C}(A, –) : \mathscr{C} \to Set$ given by
$$\mathscr{C}(A, B), \qquad \mathscr{C}(A, f : B \to B')(h : A \to B) = f \circ h : \mathscr{C}(A, B').$$
7. Functors between posets are monotonous maps, which in turn are the morphisms in Pos.

8. Functors between monoids are monoid homomorphisms, which in turn are the morphisms in Mon.
9. The power set functor $\mathcal{P} : \mathsf{Set} \to \mathsf{Set}$ defined by

$$\begin{aligned} \mathcal{P}X &= \{Y \mid Y \subseteq X\} \\ (\mathcal{P}f)Y &= f[Y] \subseteq X', \text{ for } Y \subseteq X. \end{aligned}$$

10. If $\mathscr{C}$ is a category that adds some structure to sets (like Mon or Pos) one usually can construct a *forgetful functor* $U : \mathscr{C} \to \mathsf{Set}$, e.g.

$$\begin{aligned} U_{\mathsf{Pos}} : \mathsf{Pos} &\to \mathsf{Set}; & (X, \leq) &\mapsto X \\ U_{\mathsf{Mon}} : \mathsf{Mon} &\to \mathsf{Set}; & (M, \cdot, e) &\mapsto M \end{aligned}$$

Using functors as morphisms, one can *almost* build a category CAT of *all* categories, however the collection of all categories is not a class (as required) but a *conglomerate*, thus CAT is called a *quasi-category*. The small categories (i.e. where the collection of objects is a set) form a 'real' category Cat.

We can however consider structures like products and isomorphisms in the quasi-category CAT:

Definition 2.6.3 (Products of Categories)**.** The product of two categories $\mathscr{C}, \mathscr{D}$ consists of

- $|\mathscr{C} \times \mathscr{D}| = |\mathscr{C}| \times |\mathscr{D}|$,
- $(\mathscr{C} \times \mathscr{D})((A_1, A_2), (B_1, B_2)) = \mathscr{C}(A_1, B_1) \times D(A_2, B_2)$,

with projection functors $\pi_1 : \mathscr{C} \times \mathscr{D} \to \mathscr{C}, \pi_2 : \mathscr{C} \times \mathscr{D} \to \mathscr{D}$.

Example 2.6.4. More examples of functors include:

11. The Cartesian product functor: $- \times - : \mathsf{Set} \times \mathsf{Set} \to \mathsf{Set}$.
12. The binary hom-functor $\mathscr{C}(-,-) : \mathscr{C}^{op} \times \mathscr{C} \to \mathsf{Set}$ with

$$\mathscr{C}(A, B), \qquad \mathscr{C}(g : X' \to X, f : Y \to Y')(h : X \to Y) = f \circ h \circ g : \mathscr{C}(X', Y').$$

Definition 2.6.5 (Covariant and Contravariant Functors)**.** A functor $F : \mathscr{C}^{op} \to \mathscr{D}$ is called a *contravariant* functor $\mathscr{C} \to \mathscr{D}$. For differentiation, we call 'normal' functors $\mathscr{C} \to \mathscr{D}$ *covariant*.

Example 2.6.6. Examples of contravariant functors include:

13. For every $Y \in |\mathscr{C}|$ there is a contravariant hom-functor $\mathscr{C}(-, Y) : \mathscr{C}^{op} \to \mathsf{Set}$ given by

$$\mathscr{C}(X, Y), \qquad \mathscr{C}(f : X' \to X, Y)(h : X \to Y) = h \circ f : \mathscr{C}(X', Y).$$

14. $2^{(-)} : \mathsf{Set}^{op} \to \mathsf{Set}$ where

$$2^X = \{f : X \to 2\} \cong \mathcal{P}X$$

and

$$2^{(f : X \to Y)} : 2^Y \to 2^X \cong \mathcal{P}Y \to \mathcal{P}X, \qquad Z \mapsto \{x \mid fx \in Z\} = f^{-1}[Z] \subseteq X.$$

15. For every functor $F : \mathscr{C} \to \mathscr{D}$ the identical functor $F^{op} : \mathscr{C}^{op} \to \mathscr{D}^{op}$, given by

$$F^{op}C = FC, \qquad F^{op}f = Ff.$$

Isomorphisms of categories are the isomorphisms in the quasi-category CAT, thus a functor is an isomorphism iff he is bijective on both objects and morphisms. However, oftentimes categories are not isomorphic but instead *equivalent* in the following sense:

Definition 2.6.7 (Equivalence Functors)**.** A functor $F : \mathscr{C} \to \mathscr{D}$ is called

- *full* if every $F : \mathscr{C}(A, B) \to \mathscr{D}(FA, FB)$ is surjective,
- *faithful* if every $F : \mathscr{C}(A, B) \to \mathscr{D}(FA, FB)$ is injective,
- *essentially surjective (dense)* if for every $D \in \mathscr{D}$ there exists a $C \in \mathscr{C}$ such that $D \cong FC$,
- an *equivalence* if F is full, faithful and dense.

Example 2.6.8. Let us consider two examples of equivalent categories:

1. The category Par is equivalent to Set_p, which is the category of pointed sets, where objects are tuples $(X, p), p \in X$ and morphisms are point-preserving.
2. The product category $\mathsf{Set} \times \mathsf{Set}$ is equivalent to the *slice category* $\mathsf{Set}/2$, where objects are maps $X \to 2$ and morphisms $h : (X \overset{f}{\to} 2) \to (Y \overset{g}{\to} 2)$ are maps $h : X \to Y$ such that $g \circ h = f$.

2.7 Natural Transformations

Natural transformation are morphisms between functors. The definition of "naturality" was one of the original goals of category theory.

Definition 2.7.1 (Natural Transformation)**.** Given two functors $F, G : \mathscr{C} \to \mathscr{D}$. A natural transformation $\alpha : F \to G$ between these functors is a family of morphisms

$$(\alpha_C : FC \to GC)_{C \in |\mathscr{C}|},$$

such that for any $f : A \to B$ the diagram

$$\begin{array}{ccc} FA & \xrightarrow{Ff} & FB \\ {\scriptstyle \alpha_A}\downarrow & & \downarrow{\scriptstyle \alpha_B} \\ GA & \xrightarrow{Gf} & GB \end{array}$$

commutes.

Example 2.7.2. Examples of natural transformations include:

1. The obvious function $flatten : Tree\ A \to List\ A$:

$$\begin{array}{ccc} Tree\ A & \xrightarrow{tree\ f} & Tree\ B \\ {\scriptstyle flatten_A}\downarrow & & \downarrow{\scriptstyle flatten_B} \\ List\ A & \xrightarrow{list\ f} & List\ B \end{array}$$

2. For $\mathrm{Id}, Q : Set \to Set$ we have $\delta : \mathrm{Id} \to Q$ given by $\delta_X(x) = (x, x)$.

3. On $\mathcal{P}$ we can define natural transformations $\eta : \mathrm{Id} \to \mathcal{P}$ and $\mu : \mathcal{P}\mathcal{P} \to \mathcal{P}$ by:

$$\begin{aligned} \eta_X : X &\to \mathcal{P}X \\ x &\mapsto \{x\} \end{aligned}$$

and

$$\begin{aligned} \mu_X : \mathcal{P}\mathcal{P}X &\to \mathcal{P}X \\ Z &\mapsto \bigcup Z. \end{aligned}$$

4. Between Q and $\mathcal{P}$ we can consider $\alpha, \beta : Q \to \mathcal{P}$ given by

$$\begin{aligned} \alpha_X(x, y) &= \{x, y\} \\ \beta_X(x, y) &= \{x\}. \end{aligned}$$

Functors $\mathscr{C} \to \mathscr{D}$ together with natural transformations as morphisms form a quasi-category $[\mathscr{C}, \mathscr{D}]$, that is called the functor category. If $\mathscr{C}$ is small, then $[\mathscr{C}, \mathscr{D}]$ is a category, where identity and composition are defined component wise.

Example 2.7.3. Let us examine concrete examples of functor categories:

1. $[2, \mathscr{C}] \cong \mathscr{C} \times \mathscr{C}$, where 2 is the *discrete* category with two objects, i.e. 2 has no morphisms besides the identities.
2. Let $\to$ be the category with 2 objects and a single non-trivial morphism m. $[\to, \mathscr{C}]$ is the *category of morphisms* of $\mathscr{C}$, where morphisms $Fm \to Gm$ are pairs of morphisms (f, g) where

$$\begin{array}{ccc} F0 & \xrightarrow{Fm} & F1 \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ G0 & \xrightarrow{Gm} & G1 \end{array}$$

commutes.

Definition 2.7.4 (Natural Isomorphism)**.** Isomorphisms in $[\mathscr{C}, \mathscr{D}]$ are called *natural isomorphisms.*

Proposition 2.7.5. *$\alpha : F \to G$ is a natural isomorphism* iff *every α_C is an isomorphism.*

Example 2.7.6. Let us consider some examples of natural isomorphisms:

1. In $[\mathsf{Set}, \mathsf{Set}]$ is $\mathrm{Id} \cong \mathsf{Set}(1, -)$, since of course $\mathrm{Id}\ X = X \cong X^1 = \mathsf{Set}(1, X)$.
2. Also in $[\mathsf{Set}, \mathsf{Set}]$ is $Q \cong \mathsf{Set}(2, -)$, similarly is $\lambda X.2 \times X \cong \lambda X.X + X$.
3. The forgetful functor $U : \mathsf{Pos} \to \mathsf{Set}$ is naturally isomorphic to $\mathsf{Pos}(1, -)$, because the constant mapping $x : 1 \to X$ is monotonous for every element x of a poset.

Proposition 2.7.7 (Yoneda Lemma)**.** *Let $A \in |\mathscr{C}|$ and $G : \mathscr{C} \to \mathsf{Set}$. Then the natural transformations*

$$\mathscr{C}(A, -) \to G$$

are in bijection with the elements of the set GA. In other words

$$[\mathscr{C}, \mathsf{Set}](\mathscr{C}(A, -), G) \cong GA$$

Proof. The mappings are

$$\begin{aligned} &Z : GA \to [\mathscr{C}, \mathsf{Set}](\mathscr{C}(A, -), G) \\ &Z\ x\ h = G\ h\ x \end{aligned}$$

and

$$\begin{aligned} &Y : [\mathscr{C}, \mathsf{Set}](\mathscr{C}(A, -), G) \to GA \\ &Y\ \alpha = \alpha_A\ id_A. \end{aligned}$$

We are left to check naturality of $Z\ x$ and that indeed Z and Y are inverse to each other, all of which follows by routine rewriting. □

Example 2.7.8. Let us consider an application of the Yoneda Lemma: how many natural transformations $\mathrm{Id} \to Q$ are there? Recall that $\mathrm{Id} \cong \mathsf{Set}(1, -)$, and by Yoneda there is exactly $|Q1| = 1$ natural transformation $\mathsf{Set}(1, -) \to Q$, thus the number of natural transformations $\mathrm{Id} \to Q$ is 1.

Furthermore, consider the number of natural transformations $Q \to Q$. Recall that $Q \cong \mathsf{Set}(2, -)$, and by Yoneda there are $|Q2| = 4$ natural transformations $\mathsf{Set}(2, -) \to Q$, thus the number of natural transformations $Q \to Q$ is 4.

2.8 Functor Algebras

Recall the fold functions that we introduced in chapter 1 in the category Set:

$$\begin{aligned} foldn &: (1 \to C) \to (C \to C) \qquad\quad \to Nat \quad\ \to C \\ foldr &: (1 \to C) \to (A \times C \to C) \to List\ A \to C \end{aligned}$$

These are examples of special *F-algebras* in Set. In this section we will introduce this notion and examine what makes the fold functions special.

Definition 2.8.1 (F-Algebras)**.** Let $F : \mathscr{C} \to \mathscr{C}$ be an endofunctor on $\mathscr{C}$. An *F-algebra* is a pair $(A \in |\mathscr{C}|, a : FA \to a)$. Homomorphisms between F-algebras (A, a) and (B, b) are morphisms $f : A \to B$ such that

$$\begin{array}{ccc} FA & \xrightarrow{\quad a \quad} & A \\ {\scriptstyle Ff}\downarrow & & \downarrow{\scriptstyle f} \\ FB & \xrightarrow{\quad b \quad} & B \end{array}$$

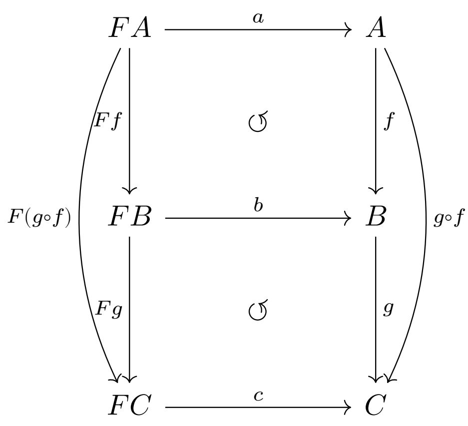
commutes.

Proposition 2.8.2. *F-algebras together with their homomorphisms form a category that we call* $\mathsf{Alg}(F)$.

Proof. Identities and composition are inherited by the underlying category $\mathscr{C}$. We are left to show that the identities are homomorphisms:

$$\begin{array}{ccc} FA & \xrightarrow{\quad a \quad} & A \\ {\scriptstyle Fid}\downarrow & \circlearrowleft & \downarrow{\scriptstyle id} \\ FA & \xrightarrow{\quad a \quad} & A \end{array}$$

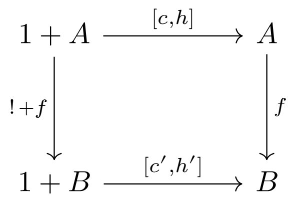
and that homomorphisms are closed under composition:



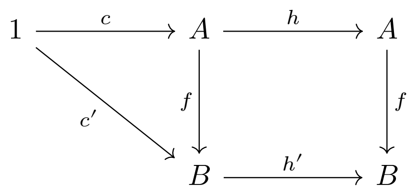
□

Example 2.8.3. Let us now consider the structure of the data types Nat and List as F-algebras:

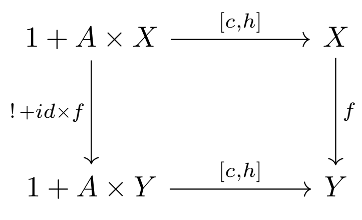
1. **Nat**: Take $\mathscr{C} = \mathsf{Set}$ and $FX = 1 + X$, the F-algebras and their morphisms have the following form:



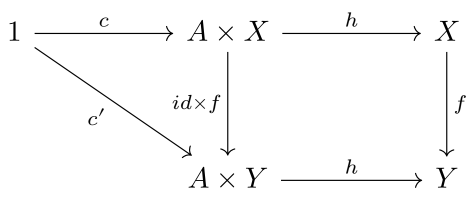
Which is equivalent to:



2. **List A**: Take $\mathscr{C} = \mathsf{Set}$ and $FX = 1 + A \times X$, where $A \in |\mathsf{Set}|$. The F-algebras and their morphisms take the following form:



Which again is equivalent to



2.8.1 Initial F-algebras

Initial F-algebras (i.e. the initial object in $\mathsf{Alg}(F)$) are of special interest to us. More concretely an F-algebra (I, i) is initial if for every (A, a) there exists a unique $(\!|a|\!) : I \to A$ such that

$$\begin{array}{ccc} FI & \xrightarrow{\;i\;} & I \\ {\scriptstyle F(\!|a|\!)}\downarrow & & \downarrow{\scriptstyle \exists!(\!|a|\!)} \\ FA & \xrightarrow[\;a\;]{} & A \end{array}$$

commutes. We sometimes denote that initial F-algebra as μF.

The dual notion of *terminal F-algebra* is usually not of interest, since it is just inherited from $\mathscr{C}$:

$$\begin{array}{ccc} FA & \xrightarrow{\;a\;} & A \\ {\scriptstyle F!}\downarrow & & \downarrow{\scriptstyle !} \\ F1 & \xrightarrow{\;!\;} & 1 \end{array}$$

Example 2.8.4. Important examples of initial F-algebras include:

1. In Example 2.8.3 (1) the data type Nat is the initial algebra together with the function $foldn$ that we defined in the introduction. Where the following diagram expresses the defining equations for $foldn$:

$$\begin{array}{ccc} 1 + Nat & \xrightarrow{[zero, succ]} & Nat \\ {\scriptstyle id + foldn(c,h)}\downarrow & & \downarrow{\scriptstyle foldn(c,h)} \\ 1 + C & \xrightarrow[{[c,h]}]{} & C \end{array}$$

2. Similarly, in Example 2.8.3 (2) the data type *List A* is the initial algebra:

$$\begin{array}{ccc} 1 + A \times List\ A & \xrightarrow{[nil, cons]} & List\ A \\ {\scriptstyle id + foldr(c,h)}\downarrow & & \downarrow{\scriptstyle foldr(c,h)} \\ 1 + A \times C & \xrightarrow{[c,h]} & C \end{array}$$

We can now abstract the fusion and identity laws that we defined for each data type in section 1.2:

Proposition 2.8.5. *Let (I, i) be an initial F-algebra. The following holds:*

1. ***Identity:*** $(\!|i|\!) = id_I : I \to I$,
2. ***Fusion:*** *Let $f : (A, a) \to (B, b)$ be a homomorphism between F-algebras, then*

$$\begin{array}{ccc} I & \xrightarrow{(\!|a|\!)} & A \\ & {\scriptstyle (\!|b|\!)}\searrow & \downarrow{\scriptstyle f} \\ & & B \end{array}$$

commutes.

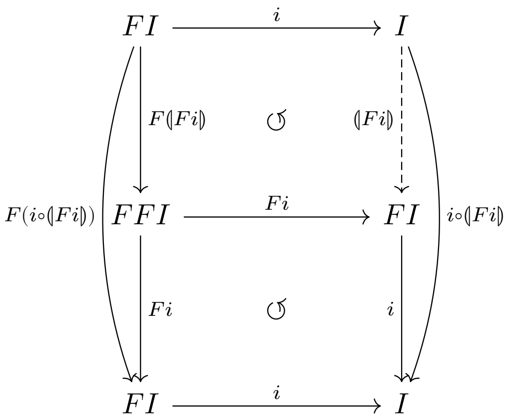
Proof. Both follow by uniqueness of homomorphisms out of the initial object:

1. By uniqueness of homomorphisms $(I, i) \to (I, i)$
2. By uniqueness of homomorphisms $(I, i) \to (B, b)$

□

Proposition 2.8.6 (Lambeks Lemma)**.** *Let (I, i) be an initial F-algebra. The F-algebra structure i is an isomorphism.*

Proof. Applying F on i yields another F-algebra (FI, Fi), which induces a homomorphism $\llparenthesis Fi \rrparenthesis : I \to FI$. $\llparenthesis FI \rrparenthesis$ is the inverse to i. Consider



from which we can follow that $i \circ \llparenthesis Fi \rrparenthesis = id_I : (I, i) \to (I, i)$ by uniqueness of the homomorphisms and thus also

$$\llparenthesis Fi \rrparenthesis \circ i = Fi \circ F\llparenthesis Fi \rrparenthesis = F(i \circ \llparenthesis Fi \rrparenthesis) = Fid_I = id_{FI}.$$

□

Example 2.8.7. Using Proposition 2.8.6, we can prove that not every functor F has an initial F-algebra. Consider the power set functor $\mathcal{P} : Set \to Set$. If there was an initial algebra $i : \mathcal{P}X \to X$, then $\mathcal{P}X \cong X$, which does not hold, because of Cantor's Theorem.

2.8.2 Term Algebras

2.8.3 Parametric Data Types

2.9 Functor Coalgebras

Coalgebras describe state based system by observations. Formally they are dual (we soon see in what sense) to F-algebras:

Definition 2.9.1 (F-Coalgebra)**.** Let $F : \mathscr{C} \to \mathscr{C}$ be an endofunctor. An F-coalgebra is a pair $(C, c : C \to FC)$ and a homomorphism between coalgebras $h : (B, b) \to (C, c)$ is a morphism

$B \to C$ such that

$$\begin{array}{ccc} B & \xrightarrow{\quad b \quad} & FB \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle Fh} \\ C & \xrightarrow{\quad c \quad} & FC \end{array}$$

commutes.

Coalgebras together with their homomorphisms form a category that we call $\mathsf{Coalg}(F)$. F-coalgebras and F-algebras are dual in the sense that $\mathsf{Coalg}(F) = \mathsf{Alg}(F^{op})^{op} \neq \mathsf{Alg}(F)^{op}$, where $F^{op} : \mathscr{C}^{op} \to \mathscr{C}^{op}$.

This time we are interested in *terminal F-coalgebras* (T, t), which are characterized by the fact that for any coalgebra (C, c) there exists a unique homomorphism such that

$$\begin{array}{ccc} C & \xrightarrow{\quad c \quad} & FC \\ {\scriptstyle [\![c]\!]}\downarrow & & \downarrow{\scriptstyle F[\![c]\!]} \\ T & \xrightarrow{\quad t \quad} & FT \end{array}$$

commutes. We sometimes denote the terminal F-algebra as νF.

Dual to F-algebras the *initial F-coalgebra* is trivial:

$$\begin{array}{ccc} 0 & \xrightarrow{\quad ¡ \quad} & F0 \\ {\scriptstyle ¡}\downarrow & & \downarrow{\scriptstyle F\,¡} \\ C & \xrightarrow{\quad c \quad} & FC \end{array}$$

Example 2.9.2. Let us now consider some examples of (terminal) F-coalgebras. They describe state systems, thus we will describe the corresponding automaton.

1. Let $FX = X + 1 : \mathsf{Set} \to \mathsf{Set}$. Let Q be the state space, then state transitions of this system have the form $Q \overset{d}{\to} Q + 1$, i.e. an element $q \in Q$ either has a next state $d\ q \in Q$ or terminates, $d\ q \in 1$.

   Homomorphisms are morphisms between state spaces that respect the state transitions d:

   - $h\ d\ q = d'\ h\ q$,
   - q terminates $\iff$ $h\ q$ terminates,

   which is expressed by the usual diagram:

   $$\begin{array}{ccc} Q & \xrightarrow{\quad d \quad} & Q + 1 \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h+!} \\ Q' & \xrightarrow{\quad d' \quad} & Q' + 1 \end{array}$$

   The terminal F-coalgebra is $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$, with $t : \mathbb{N}_\infty \to 1 + \mathbb{N}_\infty$ defined by

   $$t\ x := \begin{cases} \infty & \text{if } x = \infty \\ * \in 1 & \text{if } x = 0 \\ n & \text{if } x = n + 1 \end{cases}$$

and the unique homomorphism $h : (Q, d) \to (\mathbb{N}_\infty, t)$ returns the number of steps an element $q \in Q$ has to take until it terminates.

2. Let $FX = A \times X : \mathsf{Set} \to \mathsf{Set}$ for some set A. A coalgebra $Q \xrightarrow{\langle o,t\rangle} A \times Q$ returns for every $q \in Q$ an output $o\ q \in A$ and the next state $t\ q \in Q$.

   Homomorphisms $h : (Q, \langle o, t\rangle) \to (Q', \langle o', t'\rangle)$ are mappings $h : Q \to Q'$ such that

   - $o\ q = o'\ h\ q$,
   - $h\ t\ q = t\ h\ q$.

   The terminal F-coalgebra is A^ω, i.e. the set of *streams* over A with

   $$A^\omega \xrightarrow{\langle hd,tl\rangle} A \times A^\omega$$

   defined by

   $$(a_0, a_1, a_2, ...) \mapsto (a_0, (a_1, a_2, ...)).$$

   The unique homomorphism $h : (Q, \langle o, t\rangle) \to (A^\omega, \langle hd, tl\rangle)$ maps a state $q \in Q$ to the stream

   $$(o\ q, o(t\ q), o(t(t\ q)), ...).$$

3. Recall that deterministic finite automata (DFA) are tuples $A = (Q, \Sigma, \delta, q_0, E)$ where

   - Q is a state space,
   - Σ is a finite alphabet,
   - $\delta : Q \times \Sigma \to Q$ is a state transition function,
   - $q_0 \in Q$ is the initial state,
   - $E \subseteq Q$ is the set of accepting states.

   By changing the type of δ via currying to $\delta : Q \to Q^\Sigma$ and representing E as a map $f : Q \to 2$, we can represent a DFA (without the initial state) as a map

   $$Q \xrightarrow{\langle f,\delta\rangle} 2 \times Q^\Sigma.$$

   Thus, we can represent DFA as coalgebras for $FX = 2 \times X^\Sigma : \mathsf{Set} \to \mathsf{Set}$. A homomorphism between automata

   $$h : (Q \xrightarrow{\langle f,\delta\rangle} 2 \times Q^\Sigma) \to Q' \xrightarrow{\langle f',\delta'\rangle} 2 \times Q'^{\Sigma'}$$

   is then a mapping $h : Q \to Q'$ such that

   $$h(\delta\ (a, q)) = \delta'(a, h\ q).$$

   The terminal F-coalgebra is $2^{\Sigma^*} \cong \mathcal{P}(\Sigma^*)$, i.e. the set of formal languages over Σ with the structure $\langle \varepsilon?, \partial\rangle : \mathcal{P}(\Sigma^*) \to 2 \times \mathcal{P}(\Sigma^*)$ defined by

   $$\varepsilon?(L) = \begin{cases} 1 & \varepsilon \in L \\ 0 & \text{else} \end{cases}$$

   and

   $$\partial(L)(a) = a^{-1}L = \{w \mid aw \in L\}.$$

   Finally, the unique homomorphism $h : Q \to \mathcal{P}(\Sigma^*)$ returns for any $q \in Q$ the formal language that is accepted by q.

Proposition 2.9.3. *Let $T \xrightarrow{t} FT$ be a terminal F-coalgebra, then t is an isomorphism.*

Proof. Follows by duality from Proposition 2.8.6. □

2.9.1 Corecursion and Coinduction

Corecursion is a proof principle: each F-coalgebra (C, c) induces a unique homomorphism $h : (C, C) \to (\nu F, t)$.

Example 2.9.4. Let us consider some functions defined by corecursion:

1. Recall $A^\omega \xrightarrow{\langle hd,tl \rangle} A \times A^\omega$, we can define a function $zip : A^\omega \times A^\omega \to A^\omega$ by

$$hd(zip\ \sigma\ \tau) = hd\ \sigma; \qquad tl(zip\ \sigma\ \tau) = zip\ \tau\ (tl\ \sigma).$$

   This definition corresponds to the following coalgebra structure:

$$\begin{array}{ccc} A^\omega \times A^\omega & \xrightarrow{\langle hd \circ \pi_1, \langle \pi_2, tl \circ \pi_1 \rangle \rangle} & A \times (A^\omega \times A^\omega) \\ {\scriptstyle zip}\downarrow & & \downarrow{\scriptstyle id \times zip} \\ A^\omega & \xrightarrow[\langle hd,tl \rangle]{} & A \times A^\omega \end{array}$$

2. Similarly, for $f : A \to B$ we can define $map\ f : A^\omega \to B^\omega$ by

$$hd(map\ f\ as) = f(hd\ as); \qquad tl(map\ f\ as) = map\ f\ (tl\ as),$$

   which corresponds to the coalgebra structure:

$$\begin{array}{ccc} A^\omega & \xrightarrow{\langle hd,tl \rangle} & A \times A^\omega \\ {\scriptstyle map\ f}\downarrow & & \downarrow{\scriptstyle id \times map\ f} \\ B^\omega & \xrightarrow{\langle hd,tl \rangle} & B \times B^\omega \end{array}$$

Coinduction is a proof principle for showing *behavioral equivalence*.

Definition 2.9.5 (Behavioral equivalence)**.** Let $F : \mathsf{Set} \to \mathsf{Set}$. Two elements of coalgebras $x \in (C, c), y \in (D, d)$ are called behavioral equivalent "$x \sim y$" if there exist

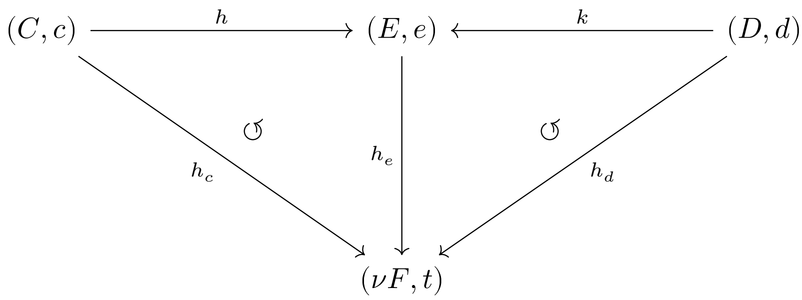
$$(C, c) \xrightarrow{h} (E, e) \xleftarrow{k} (D, d),$$

such that $h\ x = k\ y$.

Remark 2.9.6. If νF exists then

$$x \sim y \iff h_c\ x = h_d\ y,$$

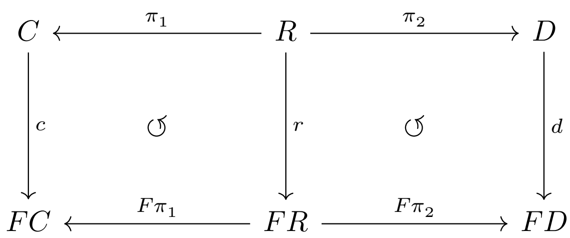
where $h_c : (C,c) \to (\nu F, t), h_d : (D,d) \to (\nu F, t)$ are the unique morphisms into the terminal coalgebra. The proof of "⇐" is clear, for "⇒" consider the diagram:



Example 2.9.7. As a consequence of the previous remark, we can follow that for $FX = 2 \times X^{\Sigma}$ the following holds:

$$x \sim y \iff x, yacceptthesameformallanguage.$$

Definition 2.9.8 (Bisimulation)**.** Let $F : \mathsf{Set} \to \mathsf{Set}$ and let $(C,c),(D,d)$ be F-coalgebras. A *bisimulation* is a relation $R \subseteq C \times C$ such that a coalgebra (R,r) exists where



commutes. Two elements $c \in C, d \in D$ are called *bisimilar* if there exists a bisimulation R such that xRy holds.

Proposition 2.9.9.

$$x, y \text{ bisimilar} \Rightarrow x \sim y$$

Example 2.9.10. Let us examine bisimilarity of streams, i.e. consider the functor $FX = A \times X$ and coalgebras

$$C \xrightarrow{\langle h,t \rangle} A \times C; \qquad D \xrightarrow{\langle h',t' \rangle} A \times D.$$

A relation $R \subseteq C \times D$ is a bisimulation if

- $h\ x = h'\ y$,
- $xRy \Rightarrow (t\ x)R(t'\ y)$.

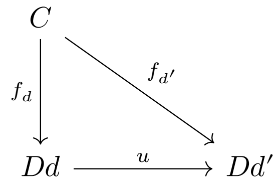
2.10 Limits

Limits are an abstraction of products and many other categorical concepts.

Definition 2.10.1 (Limit)**.** We will need to introduce some related notions first.

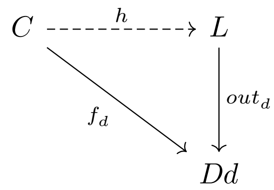
1. A *diagram* in $\mathscr{C}$ is a functor $D : \mathscr{D} \to \mathscr{C}$, where $\mathscr{D}$ is small.
2. A *cone* of a diagram $D : \mathscr{D} \to \mathscr{C}$ consists of

- an object $C \in |\mathscr{C}|$ called the *apex* and
- a family of morphisms $(f_d : C \to Dd)_{d \in |\mathscr{D}|}$ such that



commutes for every $u : d \to d'$.

3. A *limit* of a diagram D is a universal cone, i.e. a cone (L, out_d) such that for every cone (C, f_d) there exists a unique morphism $h : C \to L$ such that $out_d \circ h = f_d$ for all $d \in |\mathscr{D}|$:



The notion of limit can be instantiated to many interesting notions:

Definition 2.10.2 (Products (as limits))**.** Let $\mathscr{D}$ be the discrete category with 2 elements. Diagrams D are pairs (A, B) of objects of $\mathscr{C}$, cones are pairs of morphisms

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

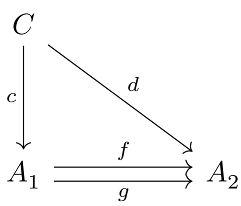
and limits of such diagrams are exactly products:

$$A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B.$$

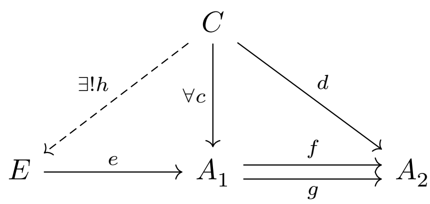
Definition 2.10.3 (Equalizer)**.** Let $\mathscr{D}$ be a category with two non-trivial and parallel morphisms $u, v : 1 \to 2$. Diagrams are parallel morphisms

$$A_1 \underset{g}{\overset{f}{\rightrightarrows}} A_2$$

and cones are pairs of morphisms $c : C \to A_1, dC \to A_2$, such that $f \circ c = d = g \circ c$:



A limit of such a diagram is called an *equalizer* of f and g:



Definition 2.10.4 (Regular Monomorphism)**.** A monomorphism is called *regular* if it is also an equalizer.

Proposition 2.10.5. *Every equalizer is a monomorphism and thus a regular monomorphism.*

Proof. □

Proposition 2.10.6. *e is a regular monomorphism and an epimorphism $\iff$ e is an isomorphism.*

Proof. □

Definition 2.10.7 (Pullback)**.**

Proposition 2.10.8. *Limits are unique up to isomorphism.*

Definition 2.10.9 (Complete Category)**.** A category $\mathscr{C}$ is called *complete* if every diagram in $\mathscr{C}$ has a limit.

Proposition 2.10.10. *$\mathscr{C}$ is complete iff $\mathscr{C}$ has all products and equalizers, i.e. using products and equalizer one can construct arbitrary limits.*

Definition 2.10.11 (Finitely Complete Category)**.** A category $\mathscr{C}$ is called *finitely complete* if every finite diagram in $\mathscr{C}$ has a limit.

Proposition 2.10.12. *The following are equivalent:*

1. *$\mathscr{C}$ is finitely complete*
2. *$\mathscr{C}$ has finite products and equalizers*
3. *$\mathscr{C}$ has finite products and pullbacks*
4. *$\mathscr{C}$ has a terminal object and pullbacks*

2.11 Colimits

3 Constructions

3.1 CPO

3.2 Initial Algebra Construction

3.3 Terminal Coalgebra Construction

Bibliography

[1] J. Adámek, H. Herrlich, and G. Strecker, *Abstract and concrete categories*. Wiley-Interscience, 1990.

[2] E. Poll and S. Thompson, 'Algebra of programming by richard bird and oege de moor, prentice hall, 1996 (dated 1997).,' *Journal of Functional Programming*, vol. 9, no. 3, pp. 347–354, 1999.