Reference Sheet for Elementary Category Theory

Categories

A **category** $\mathcal{C}$ consists of a collection of "objects" $\mathsf{Obj}\,\mathcal{C}$, a collection of "(homo)morphisms" $\mathsf{Hom}_{\mathcal{C}}(a,b)$ for any $a, b : \mathsf{Obj}\,\mathcal{C}$ –also denoted "$a \to_{\mathcal{C}} b$"–, an operation Id associating a morphism $\mathsf{Id}_a : \mathsf{Hom}(a,a)$ to each object a, and a dependently-typed "composition" operation $\_ \circ \_ : \forall\{A\,B\,C : \mathsf{Obj}\} \to \mathsf{Hom}(B,C) \to \mathsf{Hom}(A,B) \to \mathsf{Hom}(A,C)$ that is required to be associative with Id as identity.

It is convenient to define a pair of operations $\mathsf{src}, \mathsf{tgt}$ from morphisms to objects as follows:

$$f : X \to_{\mathcal{C}} Y \quad \equiv \quad \mathsf{src}\ f = X \ \wedge\ \mathsf{tgt}\ f = Y \qquad \mathsf{src},\mathsf{tgt}\text{-Definition}$$

Instead of $\mathsf{Hom}_{\mathcal{C}}$ we can instead assume primitive a ternary relation $\_ : \_ \to_{\mathcal{C}} \_$ and regain $\mathsf{Hom}_{\mathcal{C}}$ precisely when the relation is functional in its last two arguments:

$$f : A \to_{\mathcal{C}} B \ \wedge\ f : A' \to_{\mathcal{C}} B' \implies A = A' \ \wedge\ B = B' \qquad \text{TYPE-UNIQUE}$$

When this condition is dropped, we obtain a *pre-category*; e.g., the familiar *Sets* is a pre-category that is usually treated as a category by making morphisms contain the information about their source and target: $(A, f, B) : A \to B$ rather than just f.
This is sometimes easier to give than Hom! C.f. Alg(F).

Here's an equivalence-preserving property that is useful in algebraic calculations,

$$f : A \to B \quad \wedge \quad g : B \to A \quad \equiv \quad g \circ f : A \to A \quad \wedge \quad f \circ g : B \to B \qquad \text{COMPOSITION}$$

Example Categories.

- Matrices with real number values determine a category whose objects are the natural numbers, morphisms $n \to m$ are $n \times m$ matrices, Id is the identity matrix, and composition is matrix multiplication.
- Each preorder determines a category: The objects are the elements and there is a morphism $a \to b$ named, say, "(a,b)", precisely when $a \le b$; composition boils down to transitivity of $\le$.
- Each digraph determines a category: The objects are the nodes and the paths are the morphisms typed with their starting and ending node. Composition is catenation of paths and identity is the empty path.
- Suppose we have an 'interface', in the programming sense, of constant, function, and relation symbols –this is also called a *signature*.
  Let $\mathcal{T}$ be any collection of sentences in the first-order language of signature Σ. Then we can define a category $\mathsf{Mod}\,\mathcal{T}$ whose objects are implementations of interface Σ satisfying constraints $\mathcal{T}$, and whose morphisms are functions that preserve the Σ structure. Ignoring $\mathcal{T}$, gives us 'functor algebras'.
  Particular examples include monoids and structure-preserving maps between them; likewise digraphs, posets, rings, etc and their homomorphisms.

Even when morphisms are functions, the objects need not be sets: Sometimes the objects are *operations* –with an appropriate definition of typing for the functions. The categories of F-algebras are an example of this.

"Gluing" Morphisms Together

Traditional function application is replaced by the more generic concept of functional *composition* suggested by morphism-arrow chaining: Whenever we have two morphisms such that the target type of one of them, say $g : B \leftarrow A$ is the same as the source type of the other, say $f : C \leftarrow B$ then "f after g", their *composite morphism*, $f \circ g : C \leftarrow A$ can be defined. It "glues" f and g together, "sequentially":

$$C \xleftarrow{f} B \xleftarrow{g} A \qquad \Rightarrow \qquad C \xleftarrow{f \circ g} A \qquad \text{COMPOSITION-TYPE}$$

Composition is the basis for gluing morphisms together to build more complex morphisms. However, not every two morphisms can be glued together by composition.

Types provide the interface for putting morphisms together to obtain more complex functions.

A *split* arises wherever two morphisms do not compose but share the same source.
- ◇ Since they share the same source, their outputs can be paired: $c \mapsto (f\,c, g\,c)$.
- ◇ This duplicates the input so that the functions can be executed in "parallel" on it.

A *product* appears when there is no explicit relationship between the types of the morphisms.
- ◇ We regard their sources as projections of a product, whence they can be seen as *splits*.
- ◇ This $(c,d) \mapsto (f\,c, g\,d)$ corresponds to the "parallel" application of f and g, each with its *own* input.

An *either* arises wherever two morphisms do not compose but share the same target.
- ◇ Apply f if the input is from the "A side" or apply g if it is from the "B side".
- ◇ This is a "case analysis" of the input with branches being either f or g.

A *sum* appears when there is no explicit relationship between the types of the morphisms.
- ◇ We regard their targets as injections into a sum, whence they can be seen as *eithers*.

A *transpose* arises when we need to combine a binary morphism with a unary morphism.
- ◇ I.e., it arises when a composition chain is interrupted by an extra product argument.
- ◇ Express f as a C-indexed family, $f_c : A \to B$, of morphisms such that applying a function at any index behaves like f; i.e., $f_c\,a = f(c,a)$. Each f_c can now be composed with g. Let $\overline{(\ )}$ denote the operation $f \mapsto f_c$.

$$A \xleftarrow{f} C \xrightarrow{g} B \qquad \equiv \qquad A \times B \xleftarrow{\langle f,g \rangle} C \qquad \text{SPLIT-TYPE}$$

$$A \xleftarrow{f} C \ \wedge\ B \xleftarrow{g} D \qquad \equiv \qquad A \times B \xleftarrow{f \times g} C \times D \qquad \times\text{-TYPE}$$

$$A \xrightarrow{f} C \xleftarrow{g} B \qquad \equiv \qquad A + B \xrightarrow{[f,g]} C \qquad \text{EITHER-TYPE}$$

$$A \xrightarrow{f} C \ \wedge\ B \xrightarrow{g} D \qquad \equiv \qquad A + B \xrightarrow{f+g} C + D \qquad +\text{-TYPE}$$

$$B \xleftarrow{f} C \times A \qquad \equiv \qquad B^A \xleftarrow{\overline{f}} C \qquad \text{TRANSPOSE-TYPE}$$

$$B \xleftarrow{f} C \times A \ \wedge\ C \xleftarrow{g} D \qquad \Rightarrow \qquad B^A \xleftarrow{\overline{f} \circ g} D \qquad \text{TRANSPOSE-COMBINATION}$$

Functors

A **functor** $F : \mathcal{A} \to \mathcal{B}$ is a pair of mappings, denoted by one name, from the objects, and morphisms, of $\mathcal{A}$ to those of $\mathcal{B}$ such that it respects the categorical structure:

$$F\,f : F\,A \to_{\mathcal{B}} F\,B \quad \Leftarrow \quad f : A \to_{\mathcal{A}} B \qquad \text{FUNCTOR-TYPE}$$

$$F\,\mathsf{Id}_A \;=\; \mathsf{Id}_{F\,A} \qquad \text{FUNCTOR}$$

$$F\,(f \circ g) \;=\; F\,f \circ F\,g \qquad \text{FUNCTOR}$$

The two axioms are equivalent to the single statement that *functors distribute over finite compositions, with* Id *being the empty composition:*

$$F(f_0 \circ \cdots \circ f_{n-1}) \;=\; F\,f_0 \circ \cdots \circ F\,f_{n-1}$$

Use of Functors.

- In the definition of a category, "objects" are "just things" for which no internal structure is observable by categorical means –composition, identities, morphisms, typing.
  Functors form the tool to deal with "structured" objects
  Indeed in $\mathcal{S}et$ the aspect of a structure is that it has "constituents", and that it is possible to apply a function to all the individual constituents; this is done by $F\,f : F\,A \to F\,B$.
- For example, let $I\!I\,A = A \times A$ and $I\!I\,f = (x, y) \mapsto (f\,x, f\,y)$. So $I\!I$ is or represents the structure of pairs; $I\!I\,A$ is the set of pairs of A, and $I\!I\,f$ is the function that applies f to each constituent of a pair.
  - A *binary operation on* A is then just a function $I\!I\,A \to A$; in the same sense we obtain F*-ary operations.*
- Also, *Seq* is or represents the structure of sequences; *Seq* A is the structure of sequences over A, and *Seq* f is the function that applies f to each constituent of a sequence.
- Even though $F\,A$ is still just an object, a thing with no observable internal structure, the functor properties enable to exploit the "structure" of $F\,A$ by allowing us to "apply" a f to each "constituent" by using $F\,f$.

Category $\mathcal{A}lg(F)$

- For a functor $F : \mathcal{A} \to \mathcal{D}$, this category has F*-algebras*, F-ary operations in $\mathcal{D}$ as, objects – i.e., objects are $\mathcal{D}$-arrows $F\,A \to A$ – and F-homomorphisms as morphisms, and it inherits composition and identities from $\mathcal{D}$.

$$f : \oplus \to_F \otimes \quad \equiv \quad \oplus ; f = F\,f ; \otimes \qquad \text{DEFN-HOMOMORPHISM}$$

$$\mathsf{Id} : \oplus \to_F \oplus \qquad \text{ID-HOMOMORPHISM}$$

$$g \circ f : \odot \leftarrow_F \oplus \qquad \Leftarrow \qquad g : \odot \leftarrow_F \otimes \;\wedge\; f : \otimes \leftarrow_F \oplus \qquad \text{COMP-HOMOMORHISM}$$

  Note that category axiom (**??**) is not fulfilled since a function can be a homomorphism between several distinct operations. However, we pretend it is a category in the way discussed earlier, and so the carrier of an algebra is fully determined by the operation itself, so that the operation itself can be considered the algebra.

  COMP-HOMOMORHISM *renders a semantic property as a syntactic condition!*

- A **contravariant functor** $\mathcal{C} \to \mathcal{D}$ is just a functor $\mathcal{C}^{op} \to \mathcal{D}^{op}$.
- A **bifunctor** from $\mathcal{C}$ to $\mathcal{D}$ is just a functor $\mathcal{C}^{2} \to \mathcal{D}$.

Naturality

A natural transformation is nothing but a structure preserving map between functors. "Structure preservation" makes sense, here, since we've seen already that a functor is, or represents, a structure that objects might have.

As discussed before for the case $F : \mathcal{C} \to \mathcal{S}et$, each $F\,A$ denotes a structured set and F denotes the structure itself.

Example Structures: $I\!I$ is the structure of pairs, *Seq* is the structure of sequences, *Seq Seq* the structure of sequences of sequences, $I\!I$ *Seq* the structure of pairs of sequences –which is naturally isomorphic to *Seq* $I\!I$ the structure of sequences of pairs!–, and so on.

A "transformation" from structure F to structure G is a family of functions $\eta : \forall\{A\} \to F\,A \to G\,A$; and it is "natural" if each η_A doesn't affect the *constituents* of the structured elements in $F\,A$ but only reshapes the structure of the elements, from an F-structure to a G-structure.

Reshaping the structure by η *commutes with subjecting the constituents to an arbitrary morphism.*

$$\eta : F \dot{\to} G \quad \equiv \quad \forall f \bullet \eta_{\mathsf{tgt}\,f} \circ F\,f \;=\; G\,f \circ \eta_{\mathsf{src}\,f} \qquad \text{NTRF-DEF}$$

This is 'naturally' remembered: Morphism $\eta_{\mathsf{tgt}\,f}$ has type $F(\mathsf{tgt}\,f) \to G(\mathsf{tgt}\,f)$ and therefore appears at the target side of an occurrence of f; similarly $\eta_{\mathsf{src}\,f}$ occurs at the source side of an f. *Moreover* since η is a transformation *from* F to G, functor F occurs at the source side of an η and functor G at the target side.

- One also says η_a *is natural in* its parameter a.
- If we take $G = \mathsf{Id}$, then natural transformations $F \to \mathsf{Id}$ are precisely F-homomorphisms.
- Indeed, a natural transformation is a sort-of homomorphism in that the image of a morphism after reshaping is the same as the reshaping of the image.

Example natural transformations

- *rev* $: Seq \dot{\to} Seq : [a_1, \ldots, a_n] \mapsto [a_n, \ldots, a_1]$ reverses its argument thereby reshaping a sequence structure into a sequence structure without affecting the constituents.
- *inits* $: Seq \dot{\to} Seq\ Seq : [a_1, \ldots, a_n] \mapsto [[], [a_1], \cdots, [a_1, \ldots, a_n]]$ yields all initial parts of its argument thereby reshaping a sequence structure into a sequence of sequences structure, not affecting the constituents of its argument.

$$J\eta : JF \dot{\to} JG \quad \Leftarrow \quad \eta : F \dot{\to} G \quad \text{where } (J\eta)_A \coloneqq J(\eta_A) \qquad \text{NTR-FTR}$$

$$\eta K : FK \dot{\to} GK \quad \Leftarrow \quad \eta : F \dot{\to} G \quad \text{where } (\eta K)_A \coloneqq \eta_{(K\,A)} \qquad \text{NTR-POLY}$$

$$\mathsf{Id}_F : F \dot{\to} F \quad \text{where } (\mathsf{Id}_F)_A \coloneqq \mathsf{Id}_{(F\,A)} \qquad \text{NTRF-ID}$$

$$\eta \circ \epsilon : H \dot{\leftarrow} F \quad \Leftarrow \quad \eta : H \dot{\leftarrow} G \;\wedge\; \epsilon : G \dot{\leftarrow} F \quad \text{where } (\eta \circ \epsilon)_A = \eta_A \circ \epsilon_A \qquad \text{NTRF-COMPOSE}$$

Category $\mathcal{F}unc(\mathcal{C}, \mathcal{D})$ consists of functors $\mathcal{C} \to \mathcal{D}$ as objects and natrual transformations between them as objects. The identity transformation is indeed an identity for transformation composition, which is associative.

Heuristic To prove $\phi = \phi_1 \circ \cdots \circ \phi_n : F \dot{\to} G$ is a natural transformation, it suffices to show that each ϕ is a natural transformation. E.g., without even knowing the definitions, naturality of *tails* $=$ *Seq rev* $\circ$ *inits* $\circ$ *rev* can be proven –just type checking!

Adjunctions

An adjunction is a particular one-one correspondence between different kinds of morphisms in different categories.

An **adjunction** consists of two functors $L : \mathcal{A} \to \mathcal{B}$ and $R : \mathcal{B} \to \mathcal{A}$, as well as two (not necessarily natural!) transformations $\eta : \mathsf{Id} \to RL$ and $\epsilon : LR \to \mathsf{Id}$ such that

Provided $f : A \to_{\mathcal{A}} RB$ and $g : LA \to_{\mathcal{B}} B$

$$f = Rg \circ \eta_A \quad \equiv \quad \epsilon_B \circ Lf = g \qquad \text{Adjunction}$$

Reading right-to-left: In the equation $\epsilon_B \circ Lf = g$ there is a unique solution to the unknown f. Dually for the other direction.

That is, *each L-algebra g is uniquely determined –as an L-map followed by an ϵ-reduce– by its restriction to the adjunction's unit η.*

A famous example is "Free $\dashv$ Forgetful", e.g. to *define* the list datatype, for which the above becomes: Homomorphisms on lists are uniquely determined, as a map followed by a reduce, by their restriction to the singleton sequences.

We may call f the restriction, or lowering, of g to the "unital case" and write $f = \lfloor g \rfloor = Rg \circ \eta_A$. Also, we may call g the extension, or raising, of f to an L-homomorphism and write $g = \lceil f \rceil = \epsilon_B \circ Lf$. The above equivalence now reads:

$$f = \lfloor g \rfloor \quad \equiv \quad \lceil f \rceil = g \qquad \text{Adjunction-Inverse}$$

$$\lfloor g \rfloor_{A,B} = Rg \circ \eta_A \ : \ A \to_{\mathcal{A}} RB \text{ where } g : LA \to_{\mathcal{B}} B \qquad \text{lad-Type}$$

$$\lceil f \rceil_{A,B} = \epsilon_B \circ Lf \ : \ LA \to_{\mathcal{B}} B \text{ where } f : A \to_{\mathcal{A}} RB \qquad \text{rad-Type}$$

Note that $\lceil$ is like 'r' and the argument to $\lceil\rceil$ must involve the R-ight adjoint in its type; **L**ad takes morphisms involving the **L**eft adjoint ;)

This equivalence expresses that 'lad' $\lfloor\rfloor$, from *l*eft *ad*jungate, and 'rad' $\lceil\rceil$, from *r*ight *ad*jungate, are each other's inverses and constitute a correspondence between certain morphisms. *Being a bijective pair, lad and rad are injective, surjective, and undo one another.*

We may think of $\mathcal{B}$ as having all complicated problems so we abstract away some difficulties by *r*aising up to a cleaner, simpler, domain via rad $\lceil\rceil$; we then solve our problem there, then go back *down* to the more complicated concrete issue via $\lfloor\rfloor$, lad.
(E.g., $\mathcal{B}$ is the category of monoids, and $\mathcal{A}$ is the category of sets; L is the list functor.)

The η and ϵ determine each other and they are *natural* transformations. ntrf-Adj

"zig-zag laws" The unit has a post-inverse while the counit has a pre-inverse:

$$\mathsf{Id} = R\epsilon \circ \eta \qquad \text{unit-Inverse}$$

$$\mathsf{Id} = \epsilon \circ L\eta \qquad \text{Inverse-counit}$$

The unit and counit can be regained from the adjunction inverses,

$$\eta = \lfloor \mathsf{Id} \rfloor \qquad \text{unit-Def}$$

$$\epsilon = \lceil \mathsf{Id} \rceil \qquad \text{counit-Def}$$

Lad and rad themselves are solutions to the problems of interest, (Adjunction).

$$\epsilon \circ L\lfloor g \rfloor = g \qquad \text{lad-Self}$$

$$R\lceil f \rceil \circ \eta = f \qquad \text{rad-Self}$$

The following laws assert a kind of monoic-ness for ϵ and a kind of epic-ness for η. Pragmatically they allow us to prove an equality by shifting to a possibly easier equality obligation.

$$Rg \circ \eta = Rg' \circ \eta \quad \equiv \quad g = g' \qquad \text{lad-Unique}$$

$$\epsilon \circ Lf = \epsilon \circ Lf' \quad \equiv \quad f = f' \qquad \text{rad-Unique}$$

Lad and rad are natural transformations in the category $\mathcal{F}unc(\mathcal{A}^{op} \times \mathcal{B}, \mathcal{S}et)$ realising $(LX \to Y) \cong (X \to GY)$ where X, Y are the first and second projection functors and $(- \to -) : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{S}et$ is the hom-functor such that $(f \to g)h = g \circ h \circ f$.
By extensionality in $\mathcal{S}et$, their naturality amounts to the laws:

$$\lfloor y \circ g \circ Lx \rfloor \ = \ Ry \circ \lfloor g \rfloor \circ x \qquad \text{lad-Fusion}$$

$$\lceil Ry \circ f \circ x \rceil \ = \ y \circ \lceil f \rceil \circ Lx \qquad \text{rad-Fusion}$$

Also,

- ◇ Left adjoints preserve colimits such as initial objects and sums.
- ◇ Right adjoints preserve limits such as terminal objects and products.

Constant Combinators

Opposite to the identity functions which do not lose any information, we find functions which lose all, or almost all, information. Regardless of their input, the output of these functions is always the same value.

The Constant Combinator $\mathcal{K} : \mathcal{C} \to \mathcal{F}unc(\mathcal{D}, \mathcal{C})$

- ◇ For objects x, the "constant functor":
  $\mathcal{K}x\, y = x$ and $\mathcal{K}x\, f = \mathsf{Id}_x$ for objects y and morphisms f.
- ◇ For morphisms f, the "constant natural transformation":
  $\mathcal{K}f : \mathcal{K}(\mathsf{src} f) \nrightarrow \mathcal{K}(\mathsf{tgt} f)$ sending objects y to morphism $\mathcal{K}f\, y = f$.

Sometimes it is convenient to notate $\underline{c} = \mathcal{K}\, c$ and refer to this as the *everywhere c* operation.

$$\underline{c}\, a = c \qquad \text{constant-Defn}$$

The following property defines constant functions at the 'pointfree level':

$$\underline{c} \circ f = \underline{c} \qquad \text{constant-Fusion}$$

Constant functions force any difference in behaviour for any two functions to disappear:

$$\underline{c} \circ f = \underline{c} \circ g \qquad \text{constant-Equality}$$

Interestingly in $\mathcal{S}et$, composition and application are bridged explicitly by the constant functions:

$$f \circ \underline{c} = \underline{f\, c} \qquad \text{Set-Bridge}$$

Monics and Epics

Identity functions and constant functions are limit points of the functional spectrum with respect to information preservation. All the other functions are in-between: They "lose" some information, which is regarded as uninteresting for some reason.

How do functions lose information? Basically in two ways: They may be "blind" enough to confuse different inputs, by mapping them to the same output, or they may ignore values of their target. For instance, $\underline{c}$ confuses all inputs by mapping them all onto c. Moreover, it ignores all values of its target apart from c.

Functions which do not confuse their inputs are called *monics*: They are "post-cancellable":

$$f \text{ monic} \quad \equiv \quad (\forall h, k \bullet f \circ h = f \circ k \quad \equiv \quad h = k) \qquad \text{MONIC-DEFN}$$

Functions which do not ignore values of their target are called *epics*: They are "pre-cancellable":

$$f \text{ epic} \quad \equiv \quad (\forall h, k \bullet h \circ f = k \circ f \quad \equiv \quad h = k) \qquad \text{EPIC-DEFN}$$

Intuitively, $h = k$ on all points of their source precisely when they are equal on all image points of f, since f being epic means it outputs all values of their source.

It is easy to check that "the" identity function is monic and epic, while any constant function $\underline{c}$ is not monic and is only epic when its target consists only of c.

Isos

An arrow is an *iso* iff it is invertible; i.e., there is an "inverse" morphism f^{-1} with

$$f \circ f^{-1} = \mathsf{Id} \quad \wedge \quad f^{-1} \circ f = \mathsf{Id} \qquad \text{INVERSE-CHAR}$$

To *construct* f^{-1}, we begin by identifying its type which may give insight into its necessary 'shape' –e.g., as a sum or a product– then we pick one of these equations and try to reduce it as much as possible until we arrive at a definition of $f^{\smile}$, or its 'components'.

- ◇ E.g., $coassocr = [\mathsf{Id}+, \circ] : (A + B) + C \cong A + (B + C)$, its inverse *coassocl* must be of the shape $[x, [y, z]]$ for unknowns x, y, z which can be calculated by solving the equation $[x, [y, z]] \circ coassocr = \mathsf{Id}$ –Do it!

The following rules can be of help if f^{-1} is found handier than isomorphism f in reasoning,

$$f \circ x = y \quad \equiv \quad x = f^{-1} \circ y \qquad \text{INVERSE-SHUNTING1}$$

$$x \circ f = y \quad \equiv \quad x = y \circ f^{-1} \qquad \text{INVERSE-SHUNTING2}$$

Isos are necessarily monic and epic, but in general the other way around is not true.

Isomorphisms are very important because they convert data from one "format", say A, to another format, say B, without losing information. So f and $f^{\smile}$ are faithful protocols between the two formats A and B. Of course, these formats contain the same "amount" of information although the same data adopts a "different" shape in each of them. —c.f. Example Structures.

Isomorphic data domains are regarded as "abstractly" the same; then one write $A \cong B$.
Finally, note that all classes of functions referred to so far —identities, constants, epics, monics, and isos— are closed under composition.
Monics to the initial object are necessarily isos!

Skolemisation

If a property P holds for precisely one class of isomorphic objects, and for any two objects in the same class there is precisely one isomorphism from one to the other, then we say that *the P-object is unique up to unique isomorphism.* For example, in $\mathcal{S}et$ the one-point set is unique up to a unique isomorphism, but the two-point set is not.

For example, an object A is "initial" iff $\forall B \bullet \exists_1 f \bullet f : A \to B$, and such objects are unique up to unique isomorphism –prove it! The formulation of the definition is clear but it's not very well suited for *algebraic manipulation.*

A convenient formulation is obtained by 'skolemisation': An assertion of the form

$$\forall x \bullet \exists_1 y \bullet R\, x\, y$$

is equivalent to: There's a function $\mathcal{F}$ such that

$$\forall x, y \bullet R\, x\, y \;\equiv\; y = \mathcal{F}\, x \qquad (\mathcal{F}\text{-CHAR})$$

In the former formulation it is the existential quantification "$\exists y$" inside the scope of a universal one that hinders effective calculation. In the latter formulation the existence claim is brought to a more global level: A reasoning need no longer be interrupted by the declaration and naming of the existence of a unique y that depends on x; it can be denoted just $\mathcal{F}\, x$. As usual, the final universal quantification can be omitted, thus simplifying the formulation once more.

In view of the important role of the various y's, these y's deserve a particular notation that triggers the reader of their particular properties. We employ bracket notation such as $(\!|x|\!)$ for such $\mathcal{F}\, x$: An advantage of the bracket notation is that no extra parentheses are needed for composite arguments x, which we expect to occur often.

The formula *characterising* $\mathcal{F}$ may be called '$\mathcal{F}$-Char' and it immediately give us some results by truthifying each side, namely 'Self' and 'Id'. A bit more on the naming:

Type	Possibly non-syntactic constraint on notation being well-formed
Self	It, itself, is a solution
Id	How Id can be expressed using it
Uniq	Its problem has a unique solution
Fusion	How it behaves with respect to composition
Composition	How two instances, in full subcategories, compose

Note that the last 3 indicate how the concept interacts with the categorical structure: $=, \circ, \mathsf{Id}$. Also note that Self says there's at least one solution and Uniq says there is at most one solution, so together they are equivalent to $\mathcal{F}$-Char –however those two proofs are usually not easier nor more elegant than a proof of $\mathcal{F}$-Char directly.

Proving $\mathcal{F}$-Char is straightforwardly accomplished by providing a definition for $\mathcal{F}$ and establishing $\mathcal{F}$-Char –these two steps can be done in parallel! Almost every such proof has the following format, or a circular implication thereof: For arbitrary x and y,

$$\begin{array}{ll} & R\, x\, y \\ \equiv & \vdots \\ & y = \text{"an expression not involving } y\text{"} \\ \equiv & \quad \{\ define\ \mathcal{F}\, x \text{ to be the right side of the previous equation } \} \\ & y = \mathcal{F}\, x \end{array}$$

Initiality

An object *0 is initial* if there's a mapping $(\!|-|\!)$, from objects to morphisms, such that INITIAL-CHAR holds; from which we obtain a host of useful corollaries. Alternative notations for $(\!|B|\!)$ are $¡_B$, or $(\!|0 \to B|\!)$ to make the dependency on 0 explicit.

$$f : 0 \to B \quad \equiv \quad f = (\!|B|\!) \qquad \text{INITIAL-CHAR}$$

$$(\!|B|\!) : 0 \to B \qquad \text{INITIAL-SELF}$$

$$\mathsf{Id}_0 = (\!|0|\!) \qquad \text{INITIAL-ID}$$

$$f, g : 0 \to B \quad \Rightarrow \quad f = g \qquad \text{INITIAL-UNIQ}$$

Provided objects B, C are both in $\mathcal{A}$ and $\mathcal{B}$, which are full subcategories of some category $\mathcal{C}$:

$$(\!|C \leftarrow B|\!)_{\mathcal{B}} \circ (\!|B \leftarrow A|\!)_{\mathcal{A}} = (\!|C \leftarrow A|\!)_{\mathcal{A}} \qquad \text{INITIAL-COMPOSE}$$

Provided $\mathcal{D}$ is built on top of $\mathcal{C}$; i.e., $\mathcal{D}$-objects are composite entities in $\mathcal{C}$:

$$B \text{ is an object in } \mathcal{D} \quad \Rightarrow \quad (\!|B|\!) \text{ is a morphism in } \mathcal{C} \qquad \text{INITIAL-TYPE}$$

These laws become much more interesting when the category is built upon another one and the typing is expressed as one or more equations in the underlying category. In particular the importance of fusion laws cannot be over-emphasised; it is proven by a strengthening step of the form $f \circ (\!|B|\!) : 0 \to C \quad \Leftarrow \quad (\!|B|\!) : 0 \to B \quad \wedge \quad f : B \to C$.

For example, it can be seen that the datatype of sequences is 'the' initial object in a suitable category, and the mediator $(\!|-|\!)$ captures "definitions by induction on the structure"! Hence induction arguments can be replaced by initiality arguments! Woah!

Colimits

Each colimit is a certain initial object, and each initial object is a certain colimit.

- ◇ A *diagram in* $\mathcal{C}$ is a functor $D : \mathcal{D} \to \mathcal{C}$.
- ◇ Define the constant functor $\underline{C}\, x = C$ for objects x and $\underline{C}\, f = \mathsf{Id}_C$ for morphisms f. For functions $g : A \to B$, we define the natural transformation $\underline{g} = x \mapsto g : \underline{A} \dot{\to} \underline{B}$.
- ◇ The category $\bigvee D$, built upon $\mathcal{C}$, has objects $\gamma : D \dot{\to} \underline{C}$ called "co-cones", for some object $C =: \mathsf{tgt}\,\gamma$, and a morphism from γ to δ is a $\mathcal{C}$-morphism x such that $\underline{x} \circ \gamma = \delta$.
- ◇ A *colimit for* D is an initial object in $\bigvee D$; which may or may not exist.

Writing $-/\gamma$ for $(\!|-|\!)$ and working out the definition of co-cone in terms of equations in $\mathcal{C}$, we obtain: $\gamma : Obj(\bigvee D)$ *is a colimit for* D if there is a mapping $-/\gamma$ such that */-Type* and */-Char* hold.

$$\delta \text{ cocone for } D \quad \Rightarrow \quad \gamma\backslash\delta : \mathsf{tgt}\,\gamma \to \mathsf{tgt}\,\delta \qquad /\text{-TYPE}$$

Well-formedness convention: In each law the variables are quantified in such a way that the premise of /-Type is met. The notation $\delta/\cdots$ is only senseful if δ is a co-cone for D, like in arithmetic where the notation mn is only sensful if n differs from 0.

$$\underline{x} \circ \gamma = \delta \quad \equiv \quad x = \delta/\gamma \qquad /\text{-CHAR}$$

Notice that for given $x : C \to C'$ the equation $\delta/\gamma = x$ defines δ, since by /-Char that one equation equivales the family of equations $\delta_A = x \circ \gamma_A$. This allows us to define a natural transformation –or 'eithers' in the case of sums– using a single function *having* the type of the mediating arrow.

$$\underline{\delta/\gamma} \circ \gamma = \delta \qquad /\text{-SELF-CANCELLATION}$$

$$\gamma/\gamma = \mathsf{Id} \qquad /\text{-ID}$$

$$x \circ \delta/\gamma = (\underline{x} \circ \delta)/\gamma \qquad /\text{-FUSION}$$

$$\underline{x} \circ \gamma = \underline{y} \circ \gamma \quad \Rightarrow \quad x = y \qquad /\text{-UNIQUE}$$

This expresses that colimits γ have an epic-like property:
The component morphisms γ_A are *jointly epic*.

The following law confirms the choice of notation once more.

$$\epsilon/\delta \circ \delta/\gamma = \epsilon/\gamma \qquad /\text{-COMPOSE}$$

The next law tells us that functors distribute over the /-notation provided the implicit well-formedness condition that $F\gamma$ is a colimit holds –clearly this condition is valid when F preserves colimits.

$$F(\delta/\gamma) = F\delta/F\gamma \qquad /\text{-FUNCTOR-DIST}$$

$$\delta F/\gamma F = \delta/\gamma \qquad /\text{-PRE-FUNCTOR-ELIM}$$

Limits

Dually, the category $\bigwedge D$ has objects being "cones" $\gamma : \underline{C} \to D$ where $C =: \mathsf{src}\,\gamma$ is a $\mathcal{C}$-object, and a morphism to γ *from* δ is a $\mathcal{C}$-morphism x such that $\gamma \circ \underline{x} = \delta$. In terms of $\mathcal{C}$, $\gamma : Obj(\bigwedge D)$ *is a limit for* D if there is a mapping $\gamma\backslash-$ such that the following \-Type and \-Char hold, from which we obtain a host of corollaries. As usual, there is the implicit well-formedness condition. Theorem \-Unique expresses that limits γ have an monic-like property: The component morphisms γ_A are *jointly monic*.

$$\delta \text{ cone for } D \quad \Rightarrow \quad \gamma\backslash\delta : \mathsf{src}\,\delta \to \mathsf{src}\,\gamma \qquad \backslash\text{-TYPE}$$

$$\gamma \circ \underline{x} = \delta \quad \equiv \quad x = \gamma\backslash\delta \qquad \backslash\text{-CHAR}$$

$$\gamma \circ \underline{\gamma\backslash\delta} = \delta \qquad \backslash\text{-SELF}$$

$$\gamma\backslash\gamma = \mathsf{Id} \qquad \backslash\text{-ID}$$

$$\gamma\backslash\delta \circ x = \gamma\backslash(\delta \circ \underline{x}) \qquad \backslash\text{-FUSION}$$

$$\gamma \circ \underline{x} = \gamma \circ \underline{y} \quad \Rightarrow \quad x = y \qquad \backslash\text{-UNIQUE}$$

$$F(\gamma\backslash\delta) = F\gamma\backslash F\delta \qquad \backslash\text{-FUNCTOR-DIST}$$

$$\gamma F\backslash\delta F = \gamma\backslash\delta \qquad \backslash\text{-PRE-FUNCTOR-ELIM}$$

Sums

Take D and $\mathcal{D}$ as suggested by $D\mathcal{D} = \begin{pmatrix} A & B \\ \bullet & \bullet \end{pmatrix}$. Then a cocone δ for D is a two-member family $\delta = (f, g)$ with $f : A \to C$ and $g : B \to C$, where $C = \mathsf{tgt}\,\delta$.

Let $\gamma = (\mathsf{inl}, \mathsf{inr})$ be a colimit for D, let $A + B = \mathsf{tgt}\,\gamma$, and write $[f, g]$ in-place of $\gamma\backslash(f, g)$, then the \-laws yield: $(\mathsf{inl}, \mathsf{inr}, A + B)$ *form a sum of A and B* if there is a mapping $[-, -]$ such that []-Type and []-Char hold.

$$f : A \to C \quad \wedge \quad g : B \to C \qquad \Rightarrow \qquad [f, g] : A + B \to C \qquad \text{[]-Type}$$

$$x \circ \mathsf{inl} = f \quad \wedge \quad x \circ \mathsf{inr} = g \qquad \equiv \qquad x = [f, g] \qquad \text{[]-Char}$$

$$[f, g] \circ \mathsf{inl} = f \qquad \wedge \qquad [f, g] \circ \mathsf{inr} = g \qquad \text{[]-Cancellation; []-Self}$$

$$[\mathsf{inl}, \mathsf{inr}] = \mathsf{Id} \qquad \text{[]-Id}$$

$$x \circ \mathsf{inl} = y \circ \mathsf{inl} \quad \wedge \quad x \circ \mathsf{inr} = y \circ \mathsf{inr} \qquad \Rightarrow \qquad x = y \qquad \text{[]-Unique}$$

$$x \circ [f, g] = [x \circ f, x \circ g] \qquad \text{[]-Fusion}$$

The implicit well-formedness condition in the next law is that $(F\,\mathsf{inl}, F\,\mathsf{inr}, F(A + B))$ form a sum of $F\,A$ and $F\,B$.

$$F\,[f, g]_{\mathcal{C}} = [F\,f, F\,g]_{\mathcal{D}} \qquad \text{where } F : \mathcal{C} \to \mathcal{D} \qquad \text{[]-Functor-Dist}$$

In the pointwise setting, notice that the cancellation law serves to define the casing construct [-,-]. Then that casing is a form of conditional can be read from the characterisation. With this view, fusion, post-distributivity of composition over casing is just the usual law that function application distributes over conditionals and the casing extensionality law is the body-idempotency of conditionals.

For categories in which sums exist, we define for $f : A \to B$ and $g : C \to D$,

$$f + g = [\mathsf{inl} \circ f, \mathsf{inr} \circ g] : A + C \to B + D \qquad +\text{-Definition}$$

$$(f + g) \circ \mathsf{inl} = f \circ \mathsf{inl} \qquad \wedge \qquad (f + g) \circ \mathsf{inr} = \mathsf{inr} \circ g \qquad \text{Injections-Naturality}$$

$$[h \circ \mathsf{inl}, h \circ \mathsf{inr}] = h \qquad \text{Extensionality}$$

$$[h, j] \circ (f + g) = [h \circ f, j \circ g] \qquad \text{Absorption}$$

$$\mathsf{Id} + \mathsf{Id} = \mathsf{Id} \qquad \wedge \qquad (f + g) \circ (h + j) = (f \circ h) + (g \circ j) \qquad +\text{-BiFunctoriality}$$

$$[f, g] = [h, j] \qquad \equiv \qquad f = h \quad \wedge \quad g = j \qquad \text{Structural Equality}$$

Products

Take D and $\mathcal{D}$ as suggested by $D\mathcal{D} = \begin{pmatrix} A & B \\ \bullet & \bullet \end{pmatrix}$. Then a cone δ for D is a two-member family $\delta = (f, g)$ with $f : C \to A$ and $g : C \to B$, where $C = \mathsf{tgt}\,\delta$.

Let $\gamma = (\mathsf{fst}, \mathsf{snd})$ be a limit for D, let $A + B = \mathsf{tgt}\,\gamma$, and write $\langle f, g\rangle$ in-place of $(f, g)/\gamma$, then the /-laws yield: $(\mathsf{fst}, \mathsf{snd}, A \times B)$ *form a product of A and B* if there is an operation $\langle -, -\rangle$ satisfying the Char and Type laws below; from which we obtain a host of corollaries.

$$f : C \to A \quad \wedge \quad g : C \to B \qquad \Rightarrow \qquad \langle f, g\rangle : C \to A \times B \qquad \langle\rangle\text{-Type}$$

$$\mathsf{fst} \circ x = f \quad \wedge \quad \mathsf{snd} \circ x = g \qquad \equiv \qquad x = \langle f, g\rangle \qquad \langle\rangle\text{-Char}$$

$$\mathsf{fst} \circ \langle f, g\rangle = f \qquad \wedge \qquad \mathsf{snd} \circ \langle f, g\rangle = g \qquad \langle\rangle\text{-Cancellation; } \langle\rangle\text{-Self}$$

$$\langle \mathsf{fst}, \mathsf{snd}\rangle = \mathsf{Id} \qquad \langle\rangle\text{-Id}$$

$$\mathsf{fst} \circ x = \mathsf{fst} \circ y \quad \wedge \quad \mathsf{snd} \circ x = \mathsf{snd} \circ y \qquad \Rightarrow \qquad x = y \qquad \langle\rangle\text{-Unique}$$

$$\langle f, g\rangle \circ x = \langle f \circ x, g \circ x\rangle \qquad \langle\rangle\text{-Fusion}$$

$$F\,\langle f, g\rangle_{\mathcal{C}} = \langle F\,f, F\,g\rangle_{\mathcal{D}} \qquad \text{where } F : \mathcal{C} \to \mathcal{D} \qquad \langle\rangle\text{-Functor-Dist}$$

The characterisation says that the essential properties of ordered pairs is that their components are retrievable and they are completely determined by their components.

Notice that the cancellation rule is essentially the *definitions of projections* in the pointwise setting; likewise absorption is akin to the pointwise definition of the product bi-map.

The fusion laws give us a pointfree rendition of their usual pointwise definitions: All applications have been lifted to compositions!

For categories in which products exist, we define for $f : A \to B$ and $g : C \to D$,

$$f \times g = \langle f \circ \mathsf{fst}, g \circ \mathsf{snd}\rangle : A \times C \to B \times D \qquad \times\text{-Definition}$$

$$\mathsf{fst} \circ (f \times g) = f \circ \mathsf{fst} \qquad \wedge \qquad \mathsf{snd} \circ (f \times g) = g \circ \mathsf{snd} \qquad \text{Projections-Naturality}$$

$$\langle \mathsf{fst} \circ h, \mathsf{snd} \circ h\rangle = h \qquad \text{Extensionality}$$

$$(f \times g) \circ \langle h, j\rangle = \langle f \circ h, g \circ j\rangle \qquad \text{Absorption}$$

$$\mathsf{Id} \times \mathsf{Id} = \mathsf{Id} \qquad \wedge \qquad (f \times g) \circ (h \times j) = (f \circ h) \times (g \circ j) \qquad \times\text{-BiFunctoriality}$$

$$\langle f, g\rangle = \langle h, j\rangle \qquad \equiv \qquad f = h \quad \wedge \quad g = j \qquad \text{Structural Equality}$$

Finitary Sums and Products

All properties studied for binary *splits* and binary *eithers* extend to the finitary case. For the particular situation $n = 1$, we will have $\langle f \rangle = [f] = f$ and $\mathsf{inl} = \mathsf{fst} = \mathsf{Id}$, of course.

For the particular situation $n = 0$, finitary products "degenerate" to terminal object 1 and finitary sums "degenerate" to initial object 0. The standard notation for the empty split $\langle\rangle$ is $!_C$, where C is the source. Dually, the standard notation for the empty either $[]$ is $?_C$.

$$\langle\rangle_0 = []_1 \qquad \text{Empty Exchange Rule}$$

Mixing products and coproducts

Any $f : A + B \to C \times D$ can be expressed alternatively as an *either* or as a *split*. It turns out that both formats are identical:

$$\langle [f, g], [h, j] \rangle = [\langle f, h \rangle, \langle g, j \rangle] \qquad \text{Exchange Rule}$$

E.g., $\mathsf{undistr} = \langle [\mathsf{fst}, \mathsf{fst}], \mathsf{snd} + \mathsf{snd} \rangle = [\mathsf{Id} \times \mathsf{inl}, \mathsf{Id} \times \mathsf{inr}] : (A \times B) + (A \times C) \to A \times (B + C)$.

$$[f \times g, h \times k] \;=\; \langle [f, h] \circ (\mathsf{fst} + \mathsf{fst}), [g, k] \circ (\mathsf{snd} + \mathsf{snd}) \rangle \qquad \text{Cool-Property}$$

$$\langle f + g, h + k \rangle \;=\; [\langle f, h \rangle (\mathsf{inl} \times \mathsf{inl}), \langle g, k \rangle (\mathsf{inr} \times \mathsf{inr})] \qquad \text{Co-cool-Property}$$

Also, since constants ignore their inputs,

$$[\langle f, \underline{k} \rangle, \langle g, \underline{k} \rangle] = \langle [f, g], \underline{k} \rangle \qquad \text{Exchange-with-constant}$$

Coequaliser

Take D and $\mathcal{D}$ as suggested by $D\mathcal{D} = \left(\overset{A}{\bullet} \rightrightarrows^{f}_{g} \overset{B}{\bullet}\right)$; where $f, g : A \to B$ are given. Then a cocone δ for D is a two-member family $\delta = (q', q)$ with $q' : A \to C, q : B \to C, C = \mathsf{tgt}\,\delta$ and $\delta_A \circ \underline{C}h = Dh \circ \delta_B$; in-particular $q' = f \circ q = g \circ q$ whence q' is fully-determined by q alone.
Let $\gamma = (p', p) : Obj(\bigvee D)$ be a colimit for D and write $-p/$ in-place of $-/\gamma$, then the $/$-laws yield: *p is a coequaliser of (f, g)* if there is a mapping $-/p$ such that *CoEq-Type* and *CoEq-Char* hold.

$$f \circ q = g \circ q \qquad \Rightarrow \qquad q/p : \mathsf{tgt}\,p \to \mathsf{tgt}\,q \qquad \text{CoEq-Type}$$

Well-formedness convention: In each law the variables are quantified in such a way that the premise of CoEq-Type is met. The notation $q/\cdots$ is only senseful if $f \circ q = g \circ q$, like in arithmetic where the notation mn is only sensful if n differs from 0.

$$x \circ p = q \qquad \equiv \qquad x = q/p \qquad \text{CoEq-Char}$$

$$q/p \circ p = q \qquad \text{CoEq-Self}$$

$$p/p = \mathsf{Id} \qquad \text{CoEq-Id}$$

$$x \circ q/p = (x \circ q)/p \qquad \text{CoEq-Fusion}$$

$$x \circ p = y \circ p \qquad \Rightarrow \qquad x = y \qquad \text{CoEq-Unique}$$

$$r/q \circ q/p = r/p \qquad \text{CoEq-Compose}$$

References

A Gentle Introduction to Category Theory — the calculational approach
by Maarten Fokkinga

An excellent introduction to category theory with examples motivated from programming, in-particular working with sequences. All steps are shown in a calculational style –which Fokkinga has made available for use with LaTeX– thereby making it suitable for self-study.

Clear, concise, and an illuminating read.

I've deviated from his exposition by using backwards composition '$\circ$' rather than diagrammatic composition ';', as such my limit notation is his colimit notation! Be careful.

I've also consulted the delightful read Program Design by Calculation of José Oliveira.

Very accessible for anyone who wants an introduction to functional programming! The category theory is mostly implicit, but presented elegantly!