Chapter 4

CATEGORIES DERIVED FROM FUNCTORS AND NATURAL TRANSFORMATIONS

This chapter has two main motivations. One derives from the use of algebraic methods in computer science, the other from recent developments in (applied) Proof Theory. The two research directions are brought together nicely by the underlying categorical structures, which tidily generalize two constructions crucial form the perspective of this book namely, products and exponents. Here, they will be discussed in the context of monoidal and monoidal closed categories.

As already mentioned, geometry and algebra provided the background and motivations for the early developments of Category Theory. In these areas, Category Theory often suggested both a unified language and effective tools for an abstract description or specification of mathematical structures. This method, which is typical of the categorical approach, has been widely explored in computer science, in connection with ideas from universal algebra. The point is that, before performing a complicated task, a programmer needs a clear specification of it. This may be given, for example, by a set of equations, or by a logical system, or also by declarations of types (or sorts) and operations on them. Inference rules may specify a theory.

This abstract approach, in computer science as well as in mathematics, is meant to simplify the work for a concrete implementation, since only the essential or desired aspects of a task, a problem, or a mathematical structure are focused on and dealt with. Unstructured lists of goals or properties are hard to understand, are prone to errors, and may hide the core of the issue.

As we shall recall in section 4.1, algebras are usually described as sets with operations. Now (binary) operations need some kind of “product” on the carrier sets to be specified or typed, e.g., a group operation “$\cdot$” is $\cdot: G\times G\to G$. However, “$\times$” does not need to be the familar Cartesian product, as several relevant examples may be given by using (binary) functors that do not need to possess projections. One may then ask whether these functors may relate to suitable exponents, in a way that is similar for CCC’s. This further step will take us to the natural (and fruitful) generalization of CCC’s as monoidal closed categories and will relate the algebraic perspective to the “functional” one, which permeates this book.

4.1 Algebras Derived from Functors

An algebra is a set, or **carrier**, together with a family of functions, or **operations**, on the carrier. One may define categories of algebras by taking, as morphisms, well-behaved functions, the homomorphisms, between algebras of the same kind.

More precisely, let Ω be a set of operator symbols indexed by their arities. Ω_n , say, is the set operators of arity n .

4.1.1 Definition *The category* $\mathbf{Alg}_\Omega$ *, of* ***Ω-algebras****, has as objects pairs* (A,α)*, where* A *is a carrier set and, for each* n *and each operator symbol* $\rho_n \in \Omega_n$*,* α *yields a function* $\alpha_\rho : A^n \to A$*. An morphism* f *from* (A,α) *to* (A',α') *is a function* $f: A \to A$ *such that*

$$f(\alpha_\rho(a_1, \ldots, a_n)) = \alpha'_\rho(f(a_1, \ldots, f(a_n))$$

for all n, $\rho_n \in \Omega_n$ *and* $a_1, \ldots, a_n \in A$.

For example, any monoid $(A,\cdot)$ is an Ω-algebra, with a binary operation on a carrier set A. As an instance of this, take Kleene's $(PR,\circ)$, i.e., the monoid of the partial recursive functions, is an Ω-algebra over PR, as carrier, with the binary operation of composition, as operation.

Remark (Commutative) monoids provide the basic instances of linear and multilinear algebra. In particular, there is a tensor product $A \otimes B$ of commutative monoids that can be characterized by a universal bilinear map $\mu: A \times B \to A \otimes B$ such that for each bilinear $f: A \times B \to C$, there is a unique monoid homomorphism $g: A \otimes B \to C$ such that $f = g \circ \mu$. This idea, as well as the generality of monoids, provide the basic mathematical intuition for the categorical notions developed in this chapter.

One may specify more, though. For example, over $(PR,\circ)$ one may require the existence of distinguished functions, such as constants for the identity, the successor and everywhere constant functions. The behavior of these elements is specified by a set of well-known equations.

In general, this technique defines a class of Ω-algebras, called a **variety**. Varieties form full subcategories of the intended category $\mathbf{Alg}_\Omega$.

A well-known generalization of Ω-algebras is given by the many sorted, or heterogeneous, case. That is, the operators are specified over more than one carrier. The applications of these notions are nowadays a broadly construed area in computer science. However, until now, they have been more indebted to universal algebra and Abstract Model Theory than to Category Theory (see the References).

Example A **stack** is made out of finite words over a set **el** of elements $A \cup \{error\}$. Let $\{tt, ff\}$ be the boolean values and A^* the finite words on A . The following signature specifies a stack as a many sorted algebra with sorts el, stack, bool, and operators:

push : stack el → stack;
pop : stack → stack;
top : stack → el;

iserrel : el → bool;

iserrstack : stack → bool;

For variables x: stack and e: el, one has to set the following:

empty = λ

push(x,α) = "if $x \in A$ and $\alpha \in A^*$ then $x\alpha$ else error "

pop($x\alpha$) = "if $x \in A$ and $\alpha \in A^*$ then α else error "

top($x\alpha$) = "if $x \in A$ and $\alpha \in A^*$ then x else error "

iserrel(error) = tt

iserrstack(error) = tt

A few more equations specify the behavior of "iserr . . . " on elements and the stack, in the obvious way.

Observe now that a set Ω of operators determines a functor T: **Set**→**Set** defined by

$$T(A) = \{\rho(a_1, \ldots, a_n) \mid n \in \omega,\ \rho \in \Omega_n,\ a_1, \ldots, a_n \in A\}.$$

Then an ω-algebra (A,α) determines a set-theoretic function $f : F(A) \to A$ by

$$f(\rho(a_1, \ldots, a_n)) = \alpha_\rho(a_1, \ldots, a_n).$$

This informal remark suggests another generalization of the notion of Ω-algebra. Ω-algebras are based on carriers as sets; we may obtain a more category-theoretic description of the general concept of algebra by taking endofunctors over arbitrary categories, instead of T: **Set**→**Set** .

4.1.2 Definition *Let **C** be a category and T: **C**→**C** be an endofunctor. The category **T-alg** of T-algebras is defined as follows: the objects of **T-alg** are the pairs (c,α) with $c \in Ob_{\mathbf{C}}$ and $\alpha \in \mathbf{C}[T(c),c]$; the arrows between two objects (c,α) and (c',α') are all the arrows $h \in \mathbf{C}[c,c']$ such that $\alpha' \circ T(h) = h \circ \alpha$. Graphically,*

Objects (c,α) :
$$\begin{tikzcd} T(c) \arrow[d, "\alpha"] \\ c \end{tikzcd}$$

Morphisms h :
$$\begin{tikzcd} T(c) \arrow[r, "T(h)"] \arrow[d, "\alpha"] & T(c') \arrow[d, "\alpha'"] \\ c \arrow[r, "h"] & c' \end{tikzcd}$$

*Identities and composition are both inherited from **C**.*

Thus Ω-algebras are T-algebras for T: **Set**→**Set**, but T-algebras, in general, are defined over categories of structured data, not just sets. One may take, for example, a collection of data types which form a category **C** and include the type **p** of programs, over those data types, as object. Then

T: **C**→**C** and α:T(c)→c, given by T(c) = **p**×c and α(i,x) = "apply program i to input x " give a T-algebra (c,α) , for each object c (see example 1, after definition 4.2.3).

Consider now a preorder (P,≤) as a category. A functor T: P→P is a monotone function. In this setting, a T-algebra is a prefixed point for the functor T as the mere existence of α:T(e)→e corresponds to T(e) ≤ e, which means exactly that e is a prefixed point for T. Recall also that the *least* prefixed point of a monotonic function, if it exists, is always the least fixed point as well. This property has the following correspondent in Category Theory:

4.1.3 Proposition *Let* ***C*** *be a category and T:* ***C***→***C****. If (c,α) is an initial T-algebra, then α is an isomorphism from T(c) to c in* ***C.***

Proof Consider the following commutative diagram

$$\begin{tikzcd}
T(T(c)) \arrow[r, "T(\alpha)"] \arrow[d, "T(\alpha)"'] & T(c) \arrow[d, "\alpha"] \\
T(c) \arrow[r, "\alpha"] & c
\end{tikzcd}$$

(T(c),T(α)) is a T-algebra, and α∈**T-alg**[(T(c),T(α)),(c,α)]. Let η∈**T-alg**[(c,α),(T(c),T(α))] be the unique morphism given by initiality. Then α∘η and id_c are both morphisms in **T-alg**[(c,α),(c,α)] and by initiality they must be equal. Moreover,

η ∘ α	= T(α) ∘ T(η)	as η∈**T-alg**[(c,α), (T(c),T(α))]
	= T(α ∘ η)	since T is a functor
	= T(id_c)	
	= id_c	since T is a functor. ♦

We can now go a step further and extend T-algebras to a more general notion, based on functors that share only the target category, instead of endofunctors. This further step may be understood in a very abstract way. Look at the diagrams for definition 4.1.2 and generalize them by using, in the lower line, new objects d, d', say, instead of c, c', and a different functor instead of the (implicit) identity functor. This gives comma categories, which we denote by (F□↓G) instead of the frequently used (F□,G).

4.1.4 Definition *Let F:* ***C***→***A****, G:* ***D***→***A*** *be functors. The* ***comma category*** *(F□↓G) is defined as follows: the objects of (F□↓G) are the triples (c,f,d) with c∈Ob*$_{\boldsymbol{C}}$*, d∈Ob*$_{\boldsymbol{D}}$*, f∈****A****[F(c),G(d)]; the arrows between two objects (c,f,d) and (c',f',d') are all the pairs (h: c→c',k: d→d') such that f' ∘F(h) = G(k) ∘f . Graphically,*

$$\text{Objects } (c,f,d):\quad \begin{tikzcd} F(c) \arrow[d, "f"] \\ G(d) \end{tikzcd} \qquad \text{Morphisms } (h,k):\quad \begin{tikzcd} F(c) \arrow[r, "F(h)"] \arrow[d, "f"] & F(c') \arrow[d, "f'"] \\ G(d) \arrow[r, "G(k)"] & G(d') \end{tikzcd}$$

The identity of (c,f,d) *is* (id_c, id_d)*; composition is defined componentwise, that is,* $(h',k')\circ(h,k) = (h'\circ h, k'\circ k)$.

Observe that by this further abstraction, we hit upun a notion examined in another context. Indeed, if a is an object of $\mathbf{C}$, and $K_a: \mathbf{C}\to\mathbf{C}$ is the constant functor taking every object to a, and every morphism to id_a, then $(K_a\downarrow id_{\mathbf{C}})$ is just the category of objects over a, or slice category, mentioned in definition 1.3.4.

4.1.5 Example A (possibly finite) **graph** G is a triple (T,∂,V), where T is a set of arcs (or edges), V a set of nodes, and $\partial: T\to V\times V$ a function that gives the source and target of each arc. A morphism h from G to G' is a pair of functions $<f,g>$, $f: T\to T'$ and $g: V\to V'$ such that

$$g\circ p_0\circ\partial = p_0\circ\partial'\circ f \quad \text{and}$$
$$g\circ p_1\circ\partial = p_1\circ\partial'\circ f\,,$$

where p_0 and p_1 are the projections.

This, with the obvious componentwise composition of morphisms, defines the category **Graph**. **Graph** is a comma category $(Id\downarrow\Delta)$ defined by the identity and the diagonal functors Id, Δ: (**Fin**)**Set**$\to$(**Fin**)**Set**.

Exercises

1. Prove that **Graph** is cartesian.
2. Observe that a category is a graph with some extra structure: identities for each object (node) and composition for morphisms (arcs) of the due types. Give a (forgetful) functor from **Cat**, the category of small categories, to **Graph**. Is this functor full?

4.2 From monoids to monads

Consider again a monoid $(A,\cdot,e)$, i.e., a carrier A, a binary associative operation and a (specified) left and right identity for "$\cdot$".

The monoid may be described as the set A together with two functions $\mu: A\times A\to A$ and $\eta:1\to A$, where 1 is the singleton set (or terminal object in **Set**). The idea is that μ describes the internal operation "$\cdot$" and η picks up the identity in A. Associativity of the application and the properties of the identity are given by the following commutative diagrams:

$$
\begin{tikzcd}
A^3 \arrow[r, "id x\mu"] \arrow[d, "\mu x id"'] & A^2 \arrow[d, "\mu"] \\
A^2 \arrow[r] & A
\end{tikzcd}
\qquad
\begin{tikzcd}
1xA \arrow[r, "\eta x id"] \arrow[dr, "iso"'] & A^2 \arrow[d, "\mu"] & Ax1 \arrow[l, "id x\eta"'] \arrow[dl, "iso"] \\
& A &
\end{tikzcd}
$$

The abstract specification of monoids above can be generalized in several ways. We consider here the case where functors are used as carriers, instead of sets. To this purpose, recall that each endofunctor $T: \mathbf{C} \to \mathbf{C}$ has composition $T^{n+1}=T^n \circ T: \mathbf{C} \to \mathbf{C}$. Moreover, if $\mu: T^2 \to T$ is a natural transformation, whose components are $\mu_c: T^2c \to Tc$ for every $c \in Ob_{\mathbf{C}}$, then $T\mu: T^3 \to T^2$ and $\mu T: T^3 \to T^2$ are the natural transformations obtained by horizontal compositions, as in section 3.2. Their components are $(T\mu)_c = T(\mu_c): T^3c \to T^2c$ and $(\mu T)_c = \mu_{Tc}: T^3c \to T^2c$, respectively.

The idea is that an endofunctor of a category forms a monoid when the product of sets above is interpreted as a composition of functors.

4.2.1 Definition *A **monad** over a category $\mathbf{C}$ is a triple (T, μ, η), where $T: C \to C$ is a functor, $\mu: T^2 \to T$ and $\eta: Id_C \to T$ are natural transformations, and the following diagrams commute:*

$$
\begin{tikzcd}
T^3 \arrow[r, "\mu T"] \arrow[d, "T\mu"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}
\qquad
\begin{tikzcd}
T \arrow[r, "\eta T"] \arrow[dr, "id_T"'] & T^2 \arrow[d, "\mu"] & T \arrow[l, "T\eta"'] \arrow[dl, "id_T"] \\
& T &
\end{tikzcd}
$$

As in the case of monoids over a set, η and μ give the identity and the internal operation. The diagrams describe the behaviour of the (left and right) identity and associativity.

Remark Many other names have also been used in literature in place of "monad": the most common is "triple," but sometimes, you will find "triad" or "standard construction." See Barr and Wells (1985) for an interesting account of the history of this name.

Examples

1. Let M be a monoid and define T: **Set**→**Set** by $T(A)=M\times A$, $T(f)=id_M \times f$. Let $\eta_A: A \to M\times A$ and $\mu_A: M\times M\times A \to M\times A$ be, respectively, the functions that take a to $(1_M,a)$ and (m,n,a) to (mn,a). Then the associative and unarity identities follows from those of M .

2. Let T be the covariant power-set functor, i.e., T: **Set**$\rightarrow$**Set** as given by $T(A) = \mathcal{P}(A)$, the powerset of A, and $T(f)(B) = \{f(a) \mid a \in B \subseteq A\}$ as the image of B along f. Then η_A is the singleton map $\eta_A(a) = \{a\}$ and μ_A is the union map $\mu_A(B) = \cup B$. As for the next example, it is an easy exercise to show that (T, μ, η) is a monad.
3. As mentioned below definition 4.1.2, a preorder $(P, \leq)$ yields a category where endofunctors are monotone functions. If one has also a monad (T, μ, η), the natural transformations η and μ give that, for any $a \in P$, $a \leq Ta$ and $T(Ta) \leq Ta$, since $\eta_a: a \rightarrow Ta$, and $\mu_a: T(Ta) \rightarrow Ta$. By putting together these inequalities with the monotonicity of T, one has $Ta \leq T(Ta) \leq Ta$. Therefore, a monad in a partial order is a closure operator, since, in this case, $T(Ta) = Ta$.

4.2.2 Definition *A **comonad** over a category **C** is a triple (T, δ, ε), where T: **C**$\rightarrow$**C** is a functor, and $\delta: T \rightarrow T^2$ and $\varepsilon: T \rightarrow id_{\mathbf{C}}$ are natural transformations, such that the following diagrams commute:*

$$
\begin{tikzcd}
T^3 & T^2 \arrow[l, "\delta T"'] \\
T^2 \arrow[u, "T\delta"] & T \arrow[l, "\delta"] \arrow[u, "\delta"']
\end{tikzcd}
\qquad
\begin{tikzcd}
T & T^2 \arrow[l, "\varepsilon T"'] \arrow[r, "T\varepsilon"] & T \\
& T \arrow[ul, "id_T"] \arrow[u, "\delta"] \arrow[ur, "id_T"'] &
\end{tikzcd}
$$

In definition 4.1.2, we defined the notion of T-algebra over a generic endofunctor T: **C**$\rightarrow$**C**. Thus, one may have T-algebras, if any, over a monad (T, η, μ). In this case, though, it is sound to impose some extra conditions and ask that the monad's operation and identity, given by μ and η, commute with the operation of T-algebra. That is, a T-algebra (c, α) is **given by a monad** (T, η, μ) over a category **C** if it also satisfies the following commutative diagrams:

$$
\begin{tikzcd}
T^2c \arrow[r, "Th"] \arrow[d, "\mu_c"'] & Tc \arrow[d, "h"] \\
Tc \arrow[r, "h"] & c
\end{tikzcd}
\qquad
\begin{tikzcd}
c \arrow[r, "\eta_c"] \arrow[dr, "1"'] & Tc \arrow[d, "h"] \\
& c.
\end{tikzcd}
$$

Or, equivalently,

$$h \circ T(h) = h \circ \mu_c\,;$$
$$h \circ \eta_c = id_c\,.$$

When T is the functor of a monad (T, η, μ), we shall simply say T-algebras with the intended meaning to be derived by the monad. This collection of T-algebras can be organized in a category by adopting the same notion of morphism as in definition 4.1.2.

2.3 Definition *Let (T, η, μ) be a monad over a category* ***C***. *The* ***Eilenberg-Moore Category*** $\boldsymbol{C}^T$ *associated with the monad has T-algebras for objects, and for morphisms from (c, α) to (c',α') all the arrows $h \in \boldsymbol{C}[c,c']$ such that $\alpha' \circ T(h) = h \circ \alpha$.*

Examples

1. Consider the monad (T, η, μ) associated with a monoid M as in example 1 above. Then a T-algebra is a pair (A, α) where $\alpha: T(A) = M \times A \to A$. Let us write $m*a$ in place of $\alpha(m,a)$; then, for every $a \in A$ and $m,n \in M$, the equations of a T-algebra read:

$$1*a = a\ ;$$
$$(m\ n)*a = m*(n*a)\ .$$

In other words, an algebra for the monad associated with a monoid M is just what is usually called an **M-set**. Moreover the equation for a morphism h of T-algebras gives, for every $a \in A$ and $m \in M$, $h(ma) = m\ h(a)$, that is the usual notion of homomorphism of M-sets.

2. In connection with example 3 above, observe that if the category **C** is a preorder, then $c \le T(c)$, by η_c, and $T(c) \le c$, by α of the T-algebra. Thus, if **C** also happens to be a partial order, any T-algebra given by a monad is a fixed point in the p.o.set.

Let (T, η, μ) be a monad over a category **C**. Then, for every $c \in Ob_{\mathbf{C}}$, $(T(c), \mu_c)$ is a T-algebra. Indeed,

$$\mu_c \circ T(\mu_c) = \mu_c \circ \mu_{T(c)} \quad \text{by the associative law of } (T, \eta, \mu)$$
$$\mu_c \circ \eta_{T(c)} = id_c \quad \text{by the unity law of } (T, \eta, \mu)$$

The algebra $(T(c), \mu_c)$ is called the **free algebra** generated by c (with respect to T).

The computational significance of monads has been stressed recently in suggestions that they may help in understanding programs “as functions from values to computations.” The approach we sketch here, and which seems very promising, still belongs to a denotational view in program semantics; however, it suggests an alternative to the conceptual gap between the intensional (operational) and the extensional (denotational) approach to the semantics of programming languages. The idea, roughly, is to give a denotational semantics to computations. The intuition we have been mostly referring to, in the examples and in the presentation so far, is that objects are data types and morphisms are functions or programs in extenso. However, one may need a better display of the intensional aspects of computing and distinguish between values and computations. From this point of view, we can try to look at programs not as transformations from values to values, but as transformations from values (or programs) *to programs*. This should be done by stressing the effectiveness and the intensional nature of actual computations, without losing the insight and the elegance of the intended extensional approach. That is one should preserve the conceptual unity of the mathematical view (e.g., by categories and related structures), and without getting lost in the taxonomy and details of the operational descriptions.

The idea of a monad (T, η, μ) as a model for computations is that, for each set of values of type A, $T(A)$ is the object of computations of "type A." Then one may understand that $\eta_A: A \to TA$ maps values to computations, and $\mu_A: T^2A \to TA$ flattens a computation of a computation into a computation. If one also requires that η_A is a mono for every $A \in C$, then values form a "subset" of computations (see section 1.5, for subobjects).

Examples

1. The above example of the power-set functor as a monad may give a description of nondeterministic computations by looking at nondeterministic computations as sets of values. η_A picks up a specific computation, as $\eta_A(a)=\{a\}$, and μ_A says that a nondeterministic computation over non-deterministic computations, yields a nondeterministic computation, as $\mu_A(B) = \cup B \in P(A)$.
2. Even more expressively, computations with side effects may be described by the functor $T(A) = (A \times S)^S$, where S is a set of stores. Intuitively a computation takes a store and returns a value together with the modified store. Then a monad is obtained by setting, for each type (or object) A,

$$\eta_A(a) = \lambda s{:}S.(a, s) \text{ and } \mu_A(f) = \lambda s{:}S.\mathrm{eval}(fs)) .$$

The meaning of η_A should be clear; while for each $f \in (A \times S)^S$, $\mu_A(f)$ is the computation that, given a store s, first computes the pair (computation,store) given by $(f',s') = fs$; then evaluates f' applied to s' and returns a new pair (value,store).

In the application of monads the idea is to go from objects, or types, to the image by a functor of an object. More specifically, in the examples, programs take values in A, say, to computations in TB. Kleisli categories provide the right setting to describe this approach.

4.2.4 Definition *Given a monad (T, μ, η) over C, the* ***Kleisli category*** $\boldsymbol{C_T}$, *is the category whose objects are those of* $\boldsymbol{C}$ *; the set* $\boldsymbol{C_T}[A,B]$ *of morphisms from A to B in* $\boldsymbol{C_T}$ *is* $\boldsymbol{C}[A,TB]$*; the identity in* $\boldsymbol{C_T}[A,A]$ *is* $\eta : A \to TA$ *. Moreover, the composition of* $f \in \boldsymbol{C_T}[A,B]$ *and* $g \in \boldsymbol{C_T}[B,C]$ *in* $\boldsymbol{C_T}$ *is* $g \circ f = \mu_C \circ Tg \circ f : A \to TB \to T^2C \to TC$.

Exercises

1. Use the lifting functor $(\_)^\circ$ after proposition 3.2.3 and observe that the Kleisli category is the category of "total maps" over a given category of partial morphisms.
2. Give the notion of co-Kleisli category (see section 5.5).

Eilenberg-Moore and Kleisli categories will be used in the next chapter when discussing adjunctions and monads, and they will be applied in the semantics of linear logic.

4.3 Monoidal and monoidal closed categories

In section 4.2 we began by describing, with a diagram, the familiar notion of monoid. This motivated the definition of monads, when Cartesian products in **Set** are substituted with composition of functors, as in definition 4.2.1.

In both cases, we worked essentially "up to isomorphisms". As for monoids, we identified $(A\times A)\times A$ with $A\times(A\times A)$, in the upper left vertex of the first diagram in section 4.2, and called both A^3. Similarly for T^3, which is short for $T\circ(T\circ T)$ and $(T\circ T)\circ T$, we followed a similar procedure in definition 4.2.1. This is perfectly sound as both Cartesian product and composition are associative. The next step now is the explicit formalization of these (implicit) properties, including the behavior of the terminal object 1, as a left and right unit, since they are needed in all categories where it may be possible to define monoids and derived notions.

4.3.1 Definition *A **monoidal category** is a category $\boldsymbol{C}$ with a bifunctor $\otimes : \boldsymbol{C}x\boldsymbol{C} \to \boldsymbol{C}$, a (left and right) identity $e\in Ob_{\boldsymbol{C}}$ and natural isomorphisms, for each $a, b, c\in Ob_{\boldsymbol{C}}$,*

$\alpha_{a,b,c} : a\otimes(b\otimes c)\to(a\otimes b)\otimes c$;

$\lambda_a : e\otimes a\to a$;

$\rho_a : a\otimes e\to a$;

such that the following diagrams commute:

$$\begin{tikzcd}
a\otimes(b\otimes(c\otimes d)) \arrow[r, "\alpha"] \arrow[d, "1\otimes\alpha"] & (a\otimes b)\otimes(c\otimes d) \arrow[r, "\alpha"] & ((a\otimes b)\otimes c)\otimes d \\
a\otimes((b\otimes c)\otimes d) \arrow[rr, "\alpha"] & & (a\otimes(b\otimes c))\otimes d \arrow[u, "\alpha\otimes 1"']
\end{tikzcd}$$

$$\begin{tikzcd}
a\otimes(e\otimes c) \arrow[rr, "\alpha"] \arrow[dr, "id\otimes\lambda"'] & & (a\otimes e)\otimes c \arrow[dl, "\rho\otimes id"] \\
& a\otimes c &
\end{tikzcd}$$

Indexes for the natural transformations α, λ, and ρ will usually be skipped, when they are clear from the context.

Remarks 1 - The "tensor product" $\otimes$ in a monoidal category does not need to be unique, in contrast with the Cartesian product (see the example below in the category **Stab** and **Lin**). Observe also that the dual of a tensor product is, formally, just (another) tensor product, since isomorphisms define the monoidal structure and the reverse of an isomorphism is still an isomorphism.
2 - The motivation for the two diagrams in definition 4.3.1 originate from a relevant fact, whose treatment goes beyond our limited aims. In short, it may be shown that in momoidal categories, as defined above, any two (natural) isomorphisms built out of α, λ, ρ and id, by using $\otimes$ and composition, actually coincide ("coherence theorem"). For example, $a\otimes(b\otimes c)\otimes(a'\otimes b')$ and $(a\otimes b)\otimes(c\otimes a'\otimes b')$ are isomorphic in just one way.

4.3.2 Definition A **symmetric monoidal** *category* $(\mathbf{C}, \otimes, e, \alpha, \lambda, \rho)$ *is a monoidal category such that for all objects* a,b *there is a natural isomorphism* $\gamma_{a,b}: a\otimes b\to b\otimes a$ *and*

$$\gamma_{a,b} \circ \gamma_{b,a} = id_{b,a}$$
$$\rho_b = \lambda_b \circ \gamma_{b,e} : b\otimes e\to b$$

and moreover the following diagram commutes:

$$\begin{tikzcd}
a\otimes(b\otimes c) \arrow[r, "\alpha"] \arrow[d, "id_c\otimes\gamma"'] & (a\otimes b)\otimes c \arrow[r, "\gamma"] & c\otimes(a\otimes b) \arrow[d, "\alpha"] \\
a\otimes(c\otimes b) \arrow[r, "\alpha"] & (a\otimes c)\otimes b \arrow[r, "\gamma\otimes id_b"] & (c\otimes a)\otimes b
\end{tikzcd}$$

The diagram in 4.3.2 is motivated by an extension of the coherence theorem mentioned in the remark above.

Example Every Cartesian category is a symmetric monoidal category, with respect to the categorical product, and the obvious choice for the isomorphisms. Roughly, the tensor product, in the sense of monoidal categories, differs from the Cartesian product in that it has no projections and pairing functions.

Exercise Let **C** be a category with a coproduct for every pair of objects (i.e., **C** is co-Cartesian). Prove that **C** is a symmetric monoidal category.

Example In section 2.4.2, we introduced the category **Stab** of coherent domains and stable functions. Any coherent domain X is obtained from a set |X| of points and a binary relation $\uparrow$, coherence, on $|X|$. Stable maps are continuous functions which also preserve intersection of coherent arguments. **Stab** is a CCC.

In the same example, we noticed that one could also consider the linear maps between coherent domains. They are stable ones which also commute with respect to arbitrary unions (see exercise 4 below definition 2.4.2.6). **Lin** is the category of coherent domains and linear functions. Products in **Lin** are defined as for **Stab**, as the projections happen to be linear maps; thus, **Lin** is also Cartesian.

However, there is another relevant functor from **Lin**×**Lin** to **Lin**. Given coherent domains X and Y, let $(|X|\times|Y|,\uparrow^{\times})$ be given by $(x,y)\uparrow^{\times}(x',y')$ iff $x\uparrow x'$ and $y\uparrow y'$. Let $X\otimes Y$ be the coherent domain obtained from $(|X|\times|Y|,\uparrow^{\times})$. This is not a product in **Lin**, but the reader may easily check that it turns **Lin** into a symmetric monoidal category.

We will mention again the categories **Lin** and **Stab**, and their interesting interplay, in section 4.4, since they inspired Linear Logic.

Exercise Let $S^{\oplus}$ be the free commutative monoid generated by a set S. Prove then that, for $S^{\oplus}\otimes S'^{\oplus} = (S\times S')^{\oplus}$, the freely generated commutative monoids form a symmetric monoidal category.

Example Our next example is borrowed from recent investigations of parallelism and concurrency, based on a categorical description of well-known structures for those aspects of computations, namely, Petri nets.

The category of graphs was defined in the example 4.1.5. For readability we write $\partial_0 = p_0 \circ \partial$ and $\partial_1 = p_1 \circ \partial$ and set $(\partial_0,\partial_1: T\to V)$ for the graph $\mathbf{G} = (T, \partial, V)$. Now, every graph $\mathbf{G}$ may be turned into a category. Call $C(\mathbf{G})$ the category whose objects are the nodes V of $\mathbf{G}$, whose arrows are generated from the arcs T of $\mathbf{G}$ by adding the identity arc for each node and closing freely w.r.t. composition "$\circ$". Clearly, C is a functor from the category **Graph** in the example 4.1.5 to **Cat** (this method will be discussed when presenting adjunctions, see example 2 in section 5.3).

A (transition/place) **Petri net** is a graph $\mathbf{N} = (\partial_0,\partial_1: T\to S^{\oplus})$, where the arcs are called transitions and whose nodes are the elements of the free commutative monoid $S^{\oplus}$, generated by a (possibly finite) set S of labels, the places. Similarly for graphs, we may perform the free construction of a category $C(\mathbf{N})$, on top of a Petri net $\mathbf{N}$. However, $\mathbf{N}$ has an extra structure: namely the monoid structure of $S^{\oplus}$. Thus we may obtain a monoidal category as follows. Let $C^{\otimes}(\mathbf{N})$ be the category whose objects are the nodes of $\mathbf{N}$, with $u\otimes v = u\cup v$ in $S^{\oplus}$, and whose arrows are freely generated from the transitions T of $\mathbf{N}$ with respect to composition "$\circ$" and the monoidal operation $\otimes$ as well. In short, for each pair $f: u\to u'$ and $g: v\to v'$ one also has $f\otimes g: u\otimes v\to u'\otimes v'$ in $C^{\otimes}(\mathbf{N})$. The functoriality of $\otimes: C^{\otimes}(\mathbf{N})\to C^{\otimes}(\mathbf{N})$ is expressed by

$$(1) \qquad (f\otimes g)\circ(f'\otimes g') = (f\circ f')\otimes(g\circ g') .$$

The intended meaning of the monoidal operation $\_\otimes\_$ is the parallel composition of arrows, while "∘" is the sequential composition. Thus, net computations are understood as the closure of the transitions with respect to the parallel and the sequential composition. (*Exercise*: discuss the meaning of (1) above).

Monoidal categories provide the right setting for a generalization of the notion of "monoid".

4.3.3 Definition *Let $(\mathbf{C}, \otimes, e, \alpha, \lambda, \rho)$ a monoidal category. A* ***monoid*** *in* $\mathbf{C}$ *is an object c together with two arrows $\mu: c\otimes c\to c$ and $\eta: e\to c$ such that the following diagrams commute:*

$$\begin{tikzcd}
c\otimes(c\otimes c) \arrow[rr, "\alpha"] \arrow[d, "id_c\otimes\mu"'] & & (c\otimes c)\otimes c \arrow[d, "\mu\otimes id_c"] \\
c\otimes c \arrow[r, "\mu"] & c & c\otimes c \arrow[l, "\mu"']
\end{tikzcd}$$

$$\begin{tikzcd}
e\otimes c \arrow[r, "\eta\otimes id_c"] \arrow[dr, "\lambda"'] & c\otimes c \arrow[d, "\mu"] & c\otimes e \arrow[l, "id_c\otimes\eta"'] \arrow[dl, "\rho"] \\
& c &
\end{tikzcd}$$

A ***morphism*** *between two monoids (c, μ, η) and (c', μ', η') is an arrow $f: c\to c'$ in* $\mathbf{C}$ *such that*

$$f\circ\mu = \mu'\circ(f\otimes f) \ : c\otimes c\to c'$$
$$f\circ\eta = \eta' : e\to c'.$$

It is easy to prove that with the previous definition of morphisms, monoids over a monoidal category $\mathbf{C}$ form a category $\mathbf{Mon}_{\mathbf{C}}$.

Examples

1. **Set**, with a Cartesian product, is monoidal and the monoids in it are exactly the ordinary monoids.
2. Given an arbitrary category $\mathbf{C}$, consider the category of endofunctors $\mathbf{C}^{\mathbf{C}}$, with natural transformations as morphisms. Then composition "∘" of functors is a bifunctor from $\mathbf{C}^{\mathbf{C}}\times\mathbf{C}^{\mathbf{C}}$ to $\mathbf{C}^{\mathbf{C}}$ which turns $\mathbf{C}^{\mathbf{C}}$ into a monoidal category. The reader may easily check for exercise that the monoids in this category are exactly the monads in definition 4.2.1 and thus understand the elegant conceptual frame provided by the categorical developments of the notion of monoid.

Dually, we have the concept of "comonoid."

4.3.4 Definition *Let* $(\mathbf{C}, \otimes, e, \alpha, \lambda, \rho)$ *be a monoidal category. A* ***comonoid*** *in* $\mathbf{C}$ *is an object* c *together with two arrows* $\delta: c \to c\otimes c$, $\varepsilon: c \to e$ *such that the following diagrams commute:*

$$\begin{tikzcd}
c\otimes c \arrow[d, "id_c\otimes\delta"'] & c \arrow[l, "\delta"'] \arrow[r, "\delta"] & c\otimes c \arrow[d, "\delta\otimes id_c"] \\
c\otimes(c\otimes c) \arrow[rr, "\alpha"] & & (c\otimes c)\otimes c
\end{tikzcd}$$

$$\begin{tikzcd}
& c \arrow[dl, "\lambda^{-1}"'] \arrow[d, "\delta"] \arrow[dr, "\rho^{-1}"] & \\
e\otimes c & c\otimes c \arrow[l, "\varepsilon\otimes id_c"] \arrow[r, "id_c\otimes\varepsilon"'] & c\otimes e
\end{tikzcd}$$

A ***morphism*** *between two comonoids* (c, δ, ε) *and* $(c', \delta', \varepsilon')$ *is an arrow* $f: c\to c'$ *in* $\mathbf{C}$ *such that*

$$f\circ\delta' = (f\otimes f)\circ\delta \ : c\to c'\otimes c'$$
$$\varepsilon'\circ f = \varepsilon : c\to e' .$$

As we have pointed out, monoidal categories are given by an abstract notion of "product," described only in terms of (natural) isomorphisms or, equivalently, in terms of collections of isomorphisms between objects with no further properties (cartesian products also have projections).

Observe that, in the special case that **C** is Cartesian, each object is a comonoid: just set $\delta = \langle id,id\rangle$, i.e.,

$$\begin{tikzcd}
& c \arrow[dl, "id"'] \arrow[d, "\delta"] \arrow[dr, "id"] & \\
c & c\times c \arrow[l, "p_1"] \arrow[r, "p_2"'] & c
\end{tikzcd}$$

Similarly to the construction of cartesian closed categories from cartesian ones, one may extend the abstract or "equational" approach for monoidal categories and give a notion of monoidal *closed* categories.

4.3.5 Definition *Let **C** be a symmetric monoidal category, with respect to the bifunctor* $\otimes$*. Then **C** is **monoidal closed** if there is also a bifunctor* $\Rightarrow: \mathbf{C}\times\mathbf{C}\to\mathbf{C}$ *such that, for every object b, there exists an isomorphism* $\Lambda : \mathbf{C}[a\otimes b,c] \cong \mathbf{C}[a,b\Rightarrow c]$ *that is natural in a and c.*

Clearly, any CCC is monoidal closed.

3.6 Exercise

1. (Important for the purposes of sections 4.4 and 5.5) **Lin**, with $\otimes$ as in the example below definition 4.3.2, is symmetric monoidal closed (*hint*: see section 2.4.2 and use the coherent domain of traces of linear maps as "$\Rightarrow$"; note that this is not an exponent for the Cartesian product in **Lin**).
2. Consider the tensor product $\otimes$ of monoids given in the remark below definition 4.1.1. There is a natural isomorphism $A\otimes B \cong B\otimes A$, and the monoid homomorphisms between two commutative monoids A and B form a commutative monoid $[A\to B]$. Prove then that there is a natural isomorphism $[(A\otimes B)\to C] \cong [A\to[B\to C]]$, making the category **cMon**, of commutative monoids, a symmetric monoidal closed category.
3. Let e be the identity of a monoidal closed category. Prove then that $e\Rightarrow\alpha \cong \alpha$ for any object α. (*Hint:* use the diagram

$$\begin{tikzcd}
\alpha \arrow[r, "\rho^{-1}"] \arrow[d, "\Lambda(\rho)"'] & \alpha\otimes e \arrow[r, "\rho"] \arrow[d, "\Lambda(\rho)\otimes id"'] & \alpha \\
(e\Rightarrow\alpha) \arrow[r, "\rho^{-1}"'] & (e\Rightarrow\alpha)\otimes e \arrow[ur, "eval"'] &
\end{tikzcd}$$

in your proof.)

Example We already defined Petri nets and turned each individual net into a (monoidal) category. We could thus view the relation between parallelism and sequentiality as a functorial notion (see the example below definition 4.3.2). We consider now the collection of Petri nets as a category. The category **Petri**, of (transition/place) Petri nets, has Petri nets $(\partial_0,\partial_1: T\to S^{\oplus})$ as objects, where $S^{\oplus}$ is the free commutative monoid generated by S. Morphisms are pairs $<f,g>$, $f: T\to T'$ and $g: S^{\oplus}\to S'^{\oplus}$ as for graphs (see the example 4.1.5), with the extra condition that g is monoid homomorphism. (**Exercise**: describe **Petri** as a comma category). As the careful reader should have checked, given two graphs $G = (\partial_0,\partial_1: T\to V)$ and $G' = (\partial'_0, \partial'_1: T'\to V')$, their cartesian product is the graph $G\times G' = (\partial_0\times\partial'_0,\partial_1\times\partial'_1: T\times T'\to V\times V')$. When considering Petri nets $N = (\partial_0,\partial_1: T\to S^{\oplus})$ and $N' = (\partial'_0,\partial'_1: T'\to S'^{\oplus})$, their product as graphs, that is

$$N\times N' = (\partial_0\times\partial'_0,\partial_1\times\partial'_1 : T\times T'\to S^{\oplus}\times S'^{\oplus}),$$

is also a Petri net, since $S^{\oplus}\times S'^{\oplus} \cong (S+S')^{\oplus}$, i.e., the product of two free commutative monoids $S^{\oplus}$ and $S'^{\oplus}$ coincides with the monoid freely generated by $(S+S')$. The Petri net $N\times N'$ is called the *synchronous product* of the nets N and N'. Intuitively, the sychronous product of two Petri nets is the results of a composition operation with synchronization: the places of the result are the union of the places of the factors, while the transition in the synchronous product are pairs (i.e., synchronization) of the given transitions. Since $S^{\oplus}\oplus S'^{\oplus} \cong (S+S')^{\oplus}$ is valid as well, the category **Petri** has also coproducts, namely,

$$N\oplus N' = ([\partial_0,\partial'_0],[\partial_1,\partial'_1] : T+T' \to (S+S')^{\oplus}),$$

where $[\partial_i,\partial'_i]$ denotes the function induced on the coproduct $T+T'$ by the functions ∂_i and ∂'_i. Intuitively, the coproduct of two Petri nets is the result of a composition operation without synchronization: the two nets laid aside without interaction. The choice is nondeterministic because of the freedom of choosing an arbitrary initial state.

The initial net has no transition and no places, while the final net has one transition and no places. As the reader has proved for exercise, the tensor product of free commutative monoids satisfies $S^{\oplus}\otimes S'^{\oplus} = (S\times S')^{\oplus}$. Thus, one may define $\_\otimes\_ : \mathbf{Petri}^2\to\mathbf{Petri}$, the tensor product $N\otimes N'$ of two Petri nets N and N', by taking as transitions the Cartesian product of their transitions, as places the cartesian product of their places, and using the equation $S^{\oplus}\otimes S'^{\oplus} = (S\times S')^{\oplus}$ to define nodes from places. The unit object I is the Petri net $(\partial_0,\partial_1: [1]\to[1]^{\oplus})$, with $\partial_0 = \partial_1$ the inclusion of $[1]$ in $[1]^{\oplus}$. In conclusion **Petri** is cartesian, cocartesian and is a monoidal category with a tensor product derived from that operation on monoids.

The next step is to discuss the monoidal closed structure. This is easy when considering finite Petri nets since, whenever $S = \{a_1, ..., a_n\}$ is finite, we have

$$[S^{\oplus} \to S^{\oplus}] \cong \overbrace{S^{\oplus} \times \ldots \times S^{\oplus}}^{n} \cong \overbrace{S^{\oplus} \oplus \ldots \oplus S^{\oplus}}^{n} \cong (\overbrace{S'+ \ldots +S'}^{n})^{\oplus}$$

Define then the "⇒" functor, on finite Petri nets, as follows. Given Petri nets N and N', $N\Rightarrow N'$ has as arrows the set of triples

$\{(h: T\to T', g: S^{\oplus}\to S'^{\oplus}, g': S^{\oplus}\to S'^{\oplus})$ / g,g' are monoid homomorphisms, $\partial'_0 \circ h = g \circ \partial'_0$ and $\partial'_1 \circ h = g' \circ \partial_1 \}$.

For $S = \{a_1, ..., a_n\}$ the monoid of nodes is the free commutative monoid $(S'+ ... +S')^{\oplus}$, n times, and ∂_0 and ∂_1 are the second and third projection. It is then clear that (finite) **Petri** is a symmetric monoidal closed category.

Monoidal closed categories generalize Cartesian closed ones in that they also possess exponent objects $a\Rightarrow b$, or b^a, which "internalize" the hom-sets. One may then ask if there is a way to describe "internally" the behavior of funtors on morphisms. That is, given a monoidal closed category **C** and a functor F: $\mathbf{C}\to\mathbf{C}$, consider, say, $f\in\mathbf{C}[a,b]$. Then $F(f)\in\mathbf{C}[F(a),F(b)]$. Since b^a

and $F(b)^{F(a)}$ represent **C**[a,b] and **C**[F(a),F(b)] in **C**, one may study the conditions under which F is "represented" by a morphism in **C**[b^a, $F(b)^{F(a)}$], for each a and b.

4.3.7 Definition *Let* **C** *be a monoidal closed category. A functor* $F: \mathbf{C}\to\mathbf{C}$ *is* ***closed*** *if, for each a and b in* **C**, *there exists* $f_{ab}\in\mathbf{C}[b^a, F(b)^{F(a)}]$ *such that, for all* $g\in\mathbf{C}[a,b]$,

$$f_{ab}\circ \Lambda(g\circ\lambda_a) = \Lambda(F(g)\circ\lambda_{F(a)})$$

where $\Lambda : \mathbf{C}[e\otimes a,b] \cong \mathbf{C}[e,b^a]$ *and* $\lambda_a: e\otimes a\to a$ *are as in 4.3.1.*

The notion of a closed functor soundly formalizes our aim above, in view of proposition.4.3.8 below. This essentially says that, internally, f_{ab} takes the identity to the identity and behaves correctly w.r.t. to composition. We will call f_{ab} the **action** of F on b^a.

4.3.8 Proposition *Let* **C** *be monoidal closed category and* $F: \mathbf{C}\to\mathbf{C}$ *a closed functor. Then the collection* $\{f_{ab} \mid a,b\in Ob_{\mathbf{C}}\}$ *of its actions is natural in a and b. Moreover, let* $comp_{abc}: a^b\otimes b^c\to a^c$ *be the natural transformation which internalizes composition, and* $ID_a: e\to a^a$ *be the natural transformation which "picks up" the identity in* a^a, *i.e.,* $ID_a = \Lambda(\lambda_a)$. *Then the following equations hold:*

i. $f_{aa}\circ ID_a = ID_{F(a)}$

ii. $\forall g: b\to a,\ \forall h: c\to b,\ f_{ac}\circ comp\circ \Lambda(g\circ\lambda_b)\otimes\Lambda(h\circ\lambda_c) = comp\circ f_{ab}\otimes f_{bc}\circ \Lambda(g\circ\lambda_b)\otimes\Lambda(h\circ\lambda_c)$

Proof Exercise. ♦

4.3.9 Remark By similar techniques for monoidal categories, one may define other classes of categories. In particular, this can be done in order to study very weak frames dealing with a notion of exponent object. Consider for example a category **C**, not necessarily monoidal, but with an "exponent" functor which realizes the following natural isomorphism, **C**[a,c^b] ≅ **C**[b,c^a] satisfying suitable identities. **C** is called **symmetric closed**. Clearly any symmetric monoidal closed category is symmetric closed. This description of categories by (natural) isomorphisms may remind the reader of remark 3.3.3, where we described the set of isomorphisms which hold in all CCC's. Those equations do not characterize CCC's, as cartesian closure requires further structural properties, i.e. projections. They may be used instead to define the larger class of **symmetric monoidal closed categories with a terminal object**. (The reader may try to complete, as an exercise, the definition of the classes of categories in this remark.)

4.4. Monoidal Categories and Linear Logic

The perspective of this book is to introduce and use Category Theory mainly in order to understand "types as objects" of (Cartesian Closed) Categories. Indeed, this will be the focus of the chapters in

Part II. In the literature, so far, the categorical semantics of types has been mostly applied to the functional notion of type, in connection with proof theoretic investigations. The link to functional languages is described by the motto "propositions as types," which proved successful both in the mathematical investigation and in concrete applications as the design of new functional languages with powerful type systems. It seems promising to explore similar analogies for other approaches to computing, where nonfunctional constructs, e.g., parallel features, are considered. These motivations and the properties of relevant categories, namely **Stab** and **Lin**, the categories of stable and linear maps widely discussed in section 2.3 and in the previous section, have suggested a new formal system, **linear logic**.

The crucial difference between the old and new paradigms may be informally summarized as follows. When looking at "propositions as objects" one understands "proofs as functions", while one of the possible perspectives suggested by linear logic is the description of "proofs as actions". An action is roughly an operation which modifies its premises, by a sort of "physical reaction." In short, a step of logical inference modifies the state of the premises: for example, when deducing B from A, some resource in A is consumed. This is in contrast with classical and intuitionistic logic, which deal with static situations: if A implies B and A holds, then we deduce B, but A holds as before, also after the deductive process. Observe that, when ignoring the reaction, one obtains "proofs as functions" again, and thus the new approach may be considered a refinement of the "propositions as objects" analogy.

The proof theoretic description of this difference is based on a rewriting of the structural rules of deduction. We present next the core of linear logic and refer the reader to references for further readings or comparisons with other systems in Proof Theory.

Linear Logic is formally a Gentzen-like logic. The calculus of sequences of Gentzen seems a very suitable formalism for the study of proofs as dynamic actions; indeed the relevance of the *structural rules*, which instead play a quite obscure role in natural deduction, for example, allows a better formalization of the handling of formulas during the inference process, if one wants to focus on its dynamic aspects. The main difference between linear logic and Gentzen calculus of sequences is just in these structural rules. In particular, linear logic drops **weakening** and **contraction** rules, i.e.,

$$\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \text{and} \qquad \frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}$$

Intuitively, formulas can be thought of as resources, and the interpretation of a sequent of the form $A_1, \ldots, A_n \vdash B$ is that of a process (action) which *consumes* the resourses $A_1, \ldots, A_n$ and *produces* B. Thus, resources cannot be freely duplicated or erased; at most, they can be reordered. Moreover, this lack of structural rules suggests a duplication of the binary connectives of conjunction and disjunction. Namely, $\wedge$ and $\vee$ will be described as $\otimes$ and $\cup$ in the "multiplicative" case and as $\cap$ and $\oplus$ in the "additive" case below. The reason for this is clearly understood, say, in the introduction and elimination rules for $\wedge$: in the presence of contraction and weakening, Gentzen calculus does not distinguish between $(\otimes, r)$ and $(\cap, r)$, nor between $(\otimes, l)$ and $(\cap, l, 1)$ plus $(\cap, l, 2)$. Similarly for $\vee$, which is also described by two connectives, $\cup$ and $\oplus$.

In this section, we define and give categorical meaning to the core of "classical" linear logic by suggesting the basic structural properties of suitable categories where the reader may carry on the details of the interpretation. Other connectives or modalities, essentially the exponential connective "!" (of course), will be discussed in section 5.5.

4.4.1 Alphabet

1. atomic propositions, ranged over by A, B, C, . . .
2. logical symbol:
   - 2.1. multiplicative symbols:
     constants: 1 (mult. true), $\perp$ (mult. false)
     unary connective: $(\ )^{\perp}$ (linear negation)
     bynary connectives: $\otimes$ (mult. and), $\cup$ (mult. or), $—\!\circ$ (mult. or linear implication).
   - 2.2. additive symbols:
     constants: T (add. true), 0 (add. false)
     bynary connectives: $\cap$ (add. and), $\oplus$ (add. or).

The well-formed formulae are defined in the obvious way.
Greek capital letters $\Gamma, \Delta, \ldots$ denote finite (possibly empty) sequences of formulas separated by commas. The expression $\Gamma \vdash \Delta$ is called sequent.

4.4.2 Axioms and rules

Each set of (nonstructural) axioms and rules below begins with axioms and rules which deal with the identities relative to the specified connective: 1 for $\otimes$, $\perp$ for $\cup$, T for $\cap$, 0 for $\oplus$. The other rules are introduction rules.

1. structural rules

(id) $\qquad A \vdash A$

(exc, r) $\qquad \dfrac{\Gamma \vdash \Delta}{\Gamma' \vdash \Delta'}$ where Γ', Δ' are permutations of Γ, Δ.

(cut) $$\frac{\Gamma_1 \vdash A, \Delta \qquad A, \Gamma_2 \vdash \Delta'}{\Gamma_1, \Gamma_2 \vdash \Delta, \Delta'}$$

2. multiplicative rules

(1, r) $\vdash 1$

(1, l) $$\frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}$$

($\perp$, l) $$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \perp, \Delta}$$

($\perp$, l) $\perp \vdash$

($\otimes$, r) $$\frac{\Gamma_1 \vdash A, \Delta \qquad \Gamma_2 \vdash B, \Delta'}{\Gamma_1, \Gamma_2 \vdash A\otimes B, \Delta, \Delta'}$$

($\otimes$, l) $$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A\otimes B \vdash \Delta}$$

($\cup$, r) $$\frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A\cup B, \Delta}$$

($\cup$, l) $$\frac{\Gamma_1, A \vdash \Delta \qquad \Gamma_2, B \vdash \Delta'}{\Gamma_1, \Gamma_2, A\cup B \vdash \Delta, \Delta'}$$

($—$o, r) $$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A —\!\!o B, \Delta}$$

($—$o, l) $$\frac{\Gamma_1 \vdash A, \Delta \qquad \Gamma_2, B \vdash \Delta'}{\Gamma_1, \Gamma_2, A —\!\!o B \vdash \Delta, \Delta'}$$

3. additive rules

(T, r) $\Gamma \vdash T, \Delta$

(0, l) $\Gamma, 0 \vdash \Delta$

($\cap$, r) $$\frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A\cap B, \Delta}$$

($\cap$, l, 1) $$\frac{\Gamma, A \vdash \Delta}{\Gamma, A\cap B \vdash \Delta}$$

($\cap$, l, 2) $$\frac{\Gamma, B \vdash \Delta}{\Gamma, A\cap B \vdash \Delta}$$

(⊕,r,1) $\dfrac{\Gamma \mid\!\!- A, \Delta}{\Gamma \mid\!\!- A \oplus B, \Delta}$

(⊕,r,2) $\dfrac{\Gamma \mid\!\!- B, \Delta}{\Gamma \mid\!\!- A \oplus B, \Delta}$ (⊕, l) $\dfrac{\Gamma, A \mid\!\!- \Delta \quad \Gamma, B \mid\!\!- \Delta}{\Gamma, A \oplus B \mid\!\!- \Delta}$

3. linear negation

(⊥,r) $\dfrac{\Gamma, A \mid\!\!- \Delta}{\Gamma \mid\!\!- A^{\perp}, \Delta}$ (⊥,l) $\dfrac{\Gamma \mid\!\!- A, \Delta}{\Gamma, A^{\perp} \mid\!\!- \Delta}$

As suggested by the rule (⊗, l), the comma in the left hand side of a sequent has the same logical meaning as ⊗, while the commas in the right hand side correspond to the tensor sum ∪, the disjunction which is dual to ⊗.

Observe that the rules (⊥, r) and (⊥, l) are equivalent to

(contrap) $\dfrac{\Gamma, A \mid\!\!- B, \Delta}{\Gamma, B^{\perp} \mid\!\!- A^{\perp}, \Delta}$

(⊥, 1) $1 \mid\!\!- \perp^{\perp}$ and (⊥, 2) $1^{\perp} \mid\!\!- \perp$

(*Hint*: $1, \perp \mid\!\!- \perp$ gives both $\perp \mid\!\!- 1^{\perp}$ and $1 \mid\!\!- \perp^{\perp}$, while $1 \mid\!\!- 1, \perp$ gives both $\perp^{\perp} \mid\!\!- 1$ and $1^{\perp} \mid\!\!- \perp$; the rest is obvious).

From the rules one may easily derive the following sequents:

(μ_{AB}) $A, B \mid\!\!- A \otimes B$ by (⊗, r)

(eval_{AB}) $A, A \multimap B \mid\!\!- B$ by (—o, l)

In a few steps, one also obtains

(α_{ABC}) $(A \otimes B) \otimes C \mid\!\!- A \otimes (B \otimes C)$ ($\alpha^{-1}{}_{ABC}$) $A \otimes (B \otimes C) \mid\!\!- (A \otimes B) \otimes C$.

Exercise Show the logical equivalence of $(A\otimes B)$—oC and A—o$(B$—o$C)$. Derive also that A |- $(A^{\perp})^{\perp}$ and $(A^{\perp})^{\perp}$ |- A, i.e. $(\ )^{\perp}$ is "dualizing".

This may be enough to suggest that the categorical meaning of linear logic may be found in particular symmetric monoidal closed categories, where $\otimes$ and —o are interpreted by the tensorial product and the bifunctor $\Rightarrow$ given in definition 4.3.5, respectively. Observe that the cartesian product is needed too, so to provide an interpretation for $\cap$ and its identity T (see later). Negation and all dual constructions are taken care of by a "dualizing" endofunctor $(\text{-})^*$, which is closed in the sense of 4.3.7. Indeed, theorem 4.4.6 below shows that this functor, given $\otimes$ and $\cap$, immediately yields their duals.

4.4.4 Definition *A* ***∗-autonomous*** *category* ***K*** *is a symmetric monoidal closed category and a contravariant closed functor* $(\text{-})^*: \mathbf{K} \to \mathbf{K}$ *such that:*

- *there exists a natural isomorphism* $d : Id \cong (\text{-})^{**}$
- *the following diagram commutes (where* $(\text{-})^*$ *is the action of the functor on exponents)*

$$\begin{tikzcd} A \Rightarrow B \arrow[r, "(\text{-})^*"] \arrow[dr, "d\circ\_\circ d^{-1}"'] & B^*\Rightarrow A^* \arrow[d, "(\text{-})^*"] \\ & A^{**} \Rightarrow B^{**} \end{tikzcd}$$

A ∗-autonomous category is ***linear*** *if it is also cartesian.*

In other words, a linear category has both a monoidal and a Cartesian structure, $\otimes$ and $\cap$, plus a dualizing functor $(\text{-})^*$.

4.4.5 Proposition *Both the vertical and the horizontal arrows in the diagram in definition 4.4.4 are isomorphisms. Moreover,*

i. $A^* \cong B^*$ *iff* $A \cong B$;

ii. $A^* \cong A \Rightarrow 1^*$.

Proof By definition, the following diagram commutes and $d \circ \_ \circ d^{-1}$ is an isomorphism,

$$\begin{tikzcd} A^{**} \arrow[r, "f^{**}"] \arrow[d, "d^{-1}"'] & B^{**} \\ A \arrow[r, "f"'] & B \arrow[u, "d"'] \end{tikzcd}$$

Then the horizontal $(-)^*$, in the diagram in definition 4.4.4, is a split mono (see section 1.4) and the vertical one is a split epi. Since the latter is an instance of the former, they are both split monos and split epis as well and, thus, isomorphisms. In conclusion, $A \Rightarrow B \cong B^* \Rightarrow A^*$. Finally, $A^* \cong B^*$ implies $A \cong A^{**} \cong B^{**} \cong B$ and, by exercise 4.3.6.3, $A^* \cong 1 \Rightarrow A^*$. Thus, $A^* \cong B^*$ iff $A \cong B$ and, moreover, $A^* \cong A \Rightarrow 1^*$. ♦

4.4.6 Theorem *Let* $\boldsymbol{K}$ *be a linear category. Define*

$$A \cup B = (A^* \otimes B^*)^*$$
$$A \oplus B = (A^* \cap B^*)^*.$$

Then $\cup$ *is a dual to the tensor product and* $\oplus$ *is the coproduct in* $\boldsymbol{K}$. *Their identies are* $\perp = 1^*$ *and* $0 = T^*$, *respectively.*

Proof $\cup$ is a well-defined dual to $\otimes$: just compare with definition 4.3.1 (and the remark afterwards) and note that

- $A \cup \perp = (A^* \otimes 1)^* \cong A^{**} \cong A$ (similarly for the identity to the left),
- $(A^* \otimes B^*) \otimes C^* \cong A^* \otimes (B^* \otimes C^*)$ implies $(A \cup B)^* \otimes C^* \cong A^* \otimes (B \cup C)^*$ implies $((A \cup B)^* \otimes C^*)^* \cong (A^* \otimes (B \cup C)^*)^*$ implies $(A \cup B) \cup C \cong A \cup (B \cup C)$.

As for the Cartesian coproduct, note that, if p_{A^*} and p_{B^*} are the projections for $A^* \cap B^*$, then their images $(p_{A^*})^*$ and $(p_{B^*})^*$ via the $(-)^*$ functor are the injections for $A \oplus B$. Finally, $A \cup \perp = (A^* \otimes 1)^* \cong A$ and $A \oplus 0 = (A^* \cap T)^* \cong A$. ♦

The results in proposition 4.4.5 and theorem 4.4.6 prove the "dualizing" role of the $(-)^*$ functor and of $\perp = 1^*$. In particular, theorem 4.4.6 is a version of De Morgan rules in the semantic structures for linear logic, when described in categorical terms. Note also that

$$\begin{aligned} A^* \cup B &\cong (A \otimes B^*)^* \\ &\cong (A \otimes B^*) \Rightarrow \perp \\ &\cong A \Rightarrow (B^* \Rightarrow \perp) \\ &\cong A \Rightarrow (1 \Rightarrow B) \\ &\cong A \Rightarrow B \end{aligned}$$

which gives the "classic" flavor of this fragment of Linear Logic.

The meaning of linear logic as a deductive system is then given by interpreting each entailment $A \vdash B$ as a morphism between the interpretation of the formulas. Linear categories yield such an interpretation, which we sketch here, by induction on rules. Here are some of the basic cases. Some crucial ones are simply the properties corresponding to (μ_{AB}), $(eval_{AB})$, (α_{ABC}) and $(\alpha^{-1}{}_{ABC})$, given before definition 4.4.4, in monoidal closed categories. As for the others,

the rules	*are interpreted by*	*the fact that*

for 1 and $\perp$	1 and $\perp$ are (the unique) identies for $\otimes$ and $\cup$, which are expressed by commas, on the left or right, respectively, of $\vdash$.
$(\otimes,r)$ and $(\cup,l)$	$\otimes$ and $\cup$ are bifunctors
$(\cap,r)$ and $(\oplus,l)$	$\cap$ and $\oplus$ have pairing and sum of morphisms
$(\cap,l,1)$ and $(\cap,l,2)$	there exist projections for $\cap$
$(\oplus,r,1)$ and $(\oplus,r,2)$	there exist injections for $\oplus$
(contrap)	$(-)^*$ is so defined in linear categories
$(\perp,\perp)$	$\perp = 1^*$ and, thus, $\perp^* \cong 1$.

(By the argument before 4.4.4, (contrap) and $(\perp,\perp)$ are equivalent to $(\perp,r)$ and $(\perp,l)$.)

Example The category **Lin** of coherent domains and linear maps, in 2.4.2, not only provides an example of linear category, but it has even been the source of inspiration for linear logic, since linear functions depend on inputs "additively" as deductions from assumptions in this logic. (There is more than that analogy, though, as this example and section 5.5 should clarify.)

We noticed that **Lin** is Cartesian in 2.3.7(II). **Lin** is also co-Cartesian. Indeed, let $(|X|,\uparrow)$ and $(|Y|,\uparrow)$ be coherent structures. Define then the **coproduct** $X\oplus Y$ as the coherent domain associated with the coherent structure $(\{(0,z) \mid z\in|X|\}\cup\{(1,z) \mid z\in|Y|\},\uparrow^{\oplus})$, where $(a,x)\uparrow^{\oplus}(a',y)$ iff $a = a'$ and $x\uparrow y$. (Note the difference with the product: the support is the same, but the coherence relation changes. Give the embedding linear maps for exercise.)

The singleton coherent domain $T = 0 = \{\varnothing\}$, associated with the empty coherent structure, is both terminal and initial in the category. Indeed, it is the identity for both the product and the coproduct.

In the previous section we turned **Lin** into a symmetric monoidal closed category (see exercise 4.3.6). Recall, in particular, that the **tensor product** $A\otimes B$ is the coherent domain associated with the coherent structure $(|A|\times|B|, \uparrow^{\otimes})$, where $(x,y)\uparrow^{\otimes}(x',y')$ iff $x\uparrow x'$ and $y\uparrow y'$.

The dual of the tensor product, is $\cup$ given by the **tensor sum** $A\cup B$, that is, the coherent domain associated with the coherent structure $(|A|\times|B|, \uparrow^{\cup})$, where $(x,y)\uparrow^{\cup}(x',y')$ iff $((x,y) = (x',y')$ or $x\uparrow\uparrow x'$ or $y\uparrow\uparrow y')$. Also in this case, the tensor product and sum have the same support, but the coherence relation changes.

Notice that the identity for both the tensor product and its dual is the coherent domain $\underline{1} = \perp = \{\varnothing,\{1\}\}$, associated with $(\{1\}, =)$.

As for the contravariant functor (-)* , given a coherent structure (|A|, ↑), define A* as the coherent domain associated with (|A|, ↑*), where x↑*y iff ~(x↑y) in A . On a linear function f: A→B, f*: B*→A* is defined in the following way: (y,x)∈Tr(f*) iff (x,y)∈Tr(f) .

As one could expect (and easily compute), in coherent domains one has T* = 0 = T and $\underline{1}$* = ⊥ = $\underline{1}$. Moreover, the equations in theorem 4.4.6 are realized in **Lin**.

Memo For the reader's convenience, we summarize the identities in linear logic (and linear categories):

$$1 \text{ for } \otimes \,;\ \perp \text{ for } \cup \,;\ T \text{ for } \cap \,;\ 0 \text{ for } \oplus$$

which are interpreted in **Lin** as $\underline{1}$ = ⊥ = {∅,{1}} and T = 0 = {∅}, with the obvious coherent structure.

References Textbooks, which stress the applications of Category Theory to computer science with an algebraic perspective, are Arbib and Manes (1975) and Rydeheard and Burstall (1988). Goguen and Burstall (1984) contains a survey and references to part of this area, broadly construed, which ranges from the work in Elgot (1971), to the contributions of the ADJ group (see, at the end of this volume, the many references which include the names of Goguen, Thatcher, Wagner, or Wright).

Besides the references to general Category Theory, the reader may consult Barr (1979) and Barr and Wells (1985) for monoidal categories, monads and their mathematical applications. Further references, which also discuss Linear Logic, are Barr (1990) and Marti-Oliet and Meseguer (1990). The recent applications of “monoidal notions” we described are borrowed from Moggi (1989), as for the understanding of programs as “functions from values to computations”, and from Meseguer and Montanari (1988) or Degano and al. (1989), as for the examples on Petri nets (see also Marti-Oliet and Meseguer (1989)).

The main reference for linear logic is Girard (1987). Further work may be found, among others, in Girard and Lafont (1987), Lafont (1988), DePavia (1987), and Seely (1987), where ∗-autonomous categories are proposed for its semantics. Lambek (1968) contains early work on the categorical significance of logical systems and discusses weakenings of the structural rules.