Chapter 9

REFLEXIVE OBJECTS AND THE TYPE-FREE LAMBDA CALCULUS

The main aim of this book is to present category theoretic tools for the understanding of some common constructions in computer science. This is largely done in the perspective of denotational semantics, in particular in this second part of the book. It is commonly agreed that a fruitful area of application of denotational semantics has been the understanding and, in some cases, the design of functional languages. This is exactly because the theory of the intended structures is a "theory of functions," indeed Category Theory.

Functional languages are based on the notion of application and functional abstraction. That is programs are "applied," like functions, to data and, given the formal, algebraic definition of a function, it may be turned into an applicative program by "functional completeness" or "lambda abstraction." Observe that the expressive power is mostly based on recursive definitions, even though a different approach is suggested by the higher order calculi discussed in chapter 11.

The aim of this chapter is to clarify the categorical significance of the quoted expressions in the previous paragraph, e.g., "applied", "functional completeness", "lambda abstraction", "uniform", "recursive definition", in the context of a "type-free" programming style. In the previous chapter we dealt with the **typed** λ-calculus, and we discussed typed functional "application" and "abstraction" which have an immediate interpretation in CCC's. As already mentioned, it is easy to conceive a variant of the previous calculus by just erasing all type restrictions in the term formation rules. This defines the (**type-free** or **un(i)typed**) **λ-calculus**, where there is no distinction between functions and data. (In remark 9.5.12 we will suggest some good reasons by which one may better like to consider the type-free λ-calculus as a *typed* calculus with just one type: a *unityped* calculus). The set Λ of terms of the λ-calculus is thus defined by means of the following rules, starting by a given set of (type-free) variables V:

Variables if $x \in V$, then $x \in \Lambda$;
Application if $M \in \Lambda$, and $N \in \Lambda$ then $MN \in \Lambda$;
Abstraction if $M \in \Lambda$, then $\lambda x.M \in \Lambda$.

Free and bound occurences of a variable in a term, and the substitution $[N/x]M$ of a term N for a variable x in M, are defined as for the typed calculus. As usual, we identify terms that differ from each other only for the names of bound variables (α-conversion).

The **λ-theory** deals with the convertibility $M = N$ of two terms M and N. It is axiomatized by the rules

β. $(\lambda x.M)N = [N/x]M$, for x free for N in M
η. $\lambda y.My = M$, for y not free in M (write $y \in FV(M)$)

toghether with the axioms and rules needed for turning "=" into a congruence relation.

The λ-calculus is the prototype of every untyped functional programming language. Many functional languages were directly derived from the λ-calculus, from Landin's ISWIM (a notational variant of λ-calculus with an explicit recursive operator) to Edinburgh ML. Even McCarthy's language LISP, the first running functional programming language, and still one of the most used in several applications of computer science, is greatly endebted to the λ-calculus. Besides the λ-notation, LISP inherits from λ-calculus both the formal elegance and the concise syntax, essentially adding only a few primitives for list manipulation. The main difference is in the binding strategy for variables, which is static for λ-calculus and dynamic for LISP. For example, without taking into account the inessential syntactic differences between the two formalisms, let us see how the following expression is evaluated in the two languages:

$$(\lambda z.\ (\lambda y.\ (\lambda z.yM)N\ )\ (\lambda x.xz)\ )P$$

In λ-calculus, we have the following reduction sequence of reductions:

$$(\lambda z.\ (\lambda y.\ (\lambda z.yM)N\ )\ (\lambda x.xz)\ )P \rightarrow \lambda y.\ (\lambda z.yM)N\ )\ (\lambda x.xP)$$
$$\rightarrow \lambda z.\ (\lambda x.xP)M)N$$
$$\rightarrow (\lambda x.xP)M$$
$$\rightarrow MP$$

In contrast to this, LISP will first bind z to P, then bind y to λx.xz; next z will be rebound to N, and finally yM will be evaluated. This means that x will be bound to M, and then Mz is evaluated. Since LISP uses dynamic binding, the latest active bindings of the variable z is used, i.e., the evaluation of $(\lambda z.\ (\lambda y.\ (\lambda z.yM)N\ )\ (\lambda x.xz)\ )P$ is reduced to the evaluation of MN.

This has been often considered as an anomaly of LISP: in many LISP dialects, there are syntactic constructs for defining functions that guarantee a static binding for their formal parameters and, moreover, some recent LISP-like languages have completely converted to static binding (e.g., Scheme). A first consequence of dynamic binding is that the rule of α-conversion does not hold any more: in the example above, if we replace z with another variable in λz.yM, we obtain a different behavior. LISP violates *referential transparency*, while λ-calculus does satisfy it. This is not only a merely theoretical property: in programming terms, referential transparency means that, in order to understand a structured program, we need only to understand the *denotation* of the subprograms, and not their *connotations* (for example, we do not need to be concerned with the naming of variables used within the programs). These ideas are expressed in the philosophy of *modular programming*, that is of the programming style that requires the construction of program segments as self-contained boxes, or modules, with well-defined interfaces. We shall discuss in the last chapters of this book how this philosophy applies so well to strongly typed polymorphic languages.

The current treatment of both programming concepts of referential transparency and modularity provides a relevant example of an area that is greatly indebted to twenty-odd years work of in denotational semantics. We present in this chapter the categorical understanding of the semantics of

type-free Combinatory Logic and λ-calculus, whose challenging mathematical meaning actually started that work. In section 9.4, we hint at how the categorical approach suggested a new set of combinators and a simple abstract machine for implementig head reduction (CAM).

9.1 Combinatory Logic

Combinatory Logic (CL) is based on an even simpler language than λ-calculus: it just contains variables and two constant symbols K and S. Their operational behaviour is axiomatized by the rules for equality and

k. $Kxy = x$

s. $Sxyz = xz(yz)$

where, as for the λ-calculus, $M_1M_2...M_n$ stands for $(...(M_1M_2)...M_n)$.

The expressive power of λ-calculus and CL is due to their **combinatorial completeness**. That is, for any variable x and term M in their languages, there exists $\langle x\rangle M$ such that

abs. $(\langle x\rangle M)N = [N/x]M$, and $x \notin FV(\langle x\rangle M)$.

For the λ-calculus, this comes with the definition: just set $\langle x\rangle M = \lambda x.M$. As for CL, define inductively

$\langle x\rangle x = I \equiv SKK$;

$\langle x\rangle M = KM$, if M does not contain x;

$\langle x\rangle MN = S(\langle x\rangle M)(\langle x\rangle N)$.

(In general, for $\underline{x} = x_1, ...,x_n$, set $\langle \underline{x}\rangle M = \langle x_1\rangle...(\langle x_n\rangle M)$).

As a matter of fact, CL is the simplest type-free language which is functionally complete; moreover, and surprisingly enough, in 1936 Kleene proved that CL is powerful enough to compute all partial recursive functions.

Note that in type-free universes, there is no distinction between data and functions. In set-theoretic terms, this means that it is possible to apply one to the other in an undistinguished **applicative structure** (X,·), i.e., a set X with a binary operation · .

9.1.1 Definition *A model (X,·, K, S) of CL, called **Combinatory Algebra**, is an applicative structure (X,·) with two distinguished elements K, S∈X such that*

$\forall x,y \quad (K\cdot x)\cdot y = x$

$\forall x,y,z \quad ((S\cdot x)\cdot y)\cdot z = (x\cdot z)\cdot(y\cdot z)$.

As usual, we suppose that the operation · of the applicative structure associate to the left; moreover we shall usually omit it when writing terms. For instance, $(K\cdot x)\cdot y$ will be simply written as Kxy.

9.1.2 Definition *Given an environment ξ, that is a map from the set of variables of CL to X, the* ***interpretation*** *$[M]\xi$ of a combinatory term M in ξ, is inductively defined as follows:*

$$[K]\xi = \mathsf{K}$$
$$[S]\xi = \mathsf{S}$$
$$[x]\xi = \xi(x)$$
$$[MN]\xi = [M]\xi[N]\xi .$$

An interesting semantic consequence of (abs) is the following lemma which will be used later on.

9.1.3 Lemma *Let $(X,\cdot, \mathsf{K}, \mathsf{S})$ be a Combinatory Algebra. For any combinatory term M, any environment ξ and any $a\in X$,*

$$[<x>M]\xi \cdot a = [M]\xi(x=a)$$

where $\xi(x=a)$ is the environment defined by : $\xi(x=a)(z) = $ if $x=z$ then a else $\xi(z)$.

Proof

$[<x>M]\xi \cdot a = [<x>M]\xi \cdot [x]\xi(x=a)$

$= [<x>M]\xi(x=a) \cdot [x]\xi(x=a)$ since x do not occur in <x>M

$= [(<x>M)x]\xi(x=a)$

$= [M]\xi(x=a)$ by (abs). ♦

Clearly, the λ-calculus is at least as expressive as CL, since $\mathsf{K}_\lambda \equiv \lambda xy.x$ and $\mathsf{S}_\lambda \equiv \lambda xyz.xz(yz)$ represent K and S in λ-calculus (and do the same job). By this definition of K and S we obtain a sound translation form CL to λ-calculus, i.e., a translation which preserves term equalities. In the other direction, the abstraction mechanism <x>M described above naturally suggests the following translation.

9.1.4 Definition *Given a λ-term M, the associated term M_{CL} in Combinatory Logic is inductively defined by*

$$x_{CL} = x$$
$$(MN)_{CL} = M_{CL}N_{CL}$$
$$(\lambda x.M)_{CL} = <x>M_{CL}.$$

Unfortunately, this translation is not sound, that is, not all the equations provable in the λ-theory still hold after the translation. Consider for example the two equal terms $M \equiv \lambda y.x$ and $N \equiv \lambda y.(\lambda z.z)x$. Their translation by means of combinators is, respectively:

$M_{CL} = (\lambda y.x)_{CL}$

$= <y>x_{CL}$

$= <y>x$

$= \mathsf{K}x$

$$\begin{aligned} N_{CL} &= (\lambda y.(\lambda z.z)x)_{CL} \\ &= \langle y\rangle((\lambda z.z)x)_{CL} \\ &= \langle y\rangle((\lambda z.z)_{CL}x_{CL}) \\ &= \langle y\rangle((SKK)x) \\ &= S((\langle y\rangle(SKK))\ \langle y\rangle x\ ) \\ &= S((K(SKK))Kx) \end{aligned}$$

and $Kx \neq S((K(SKK))Kx)$.

The problem derives from the fact that in Combinatory Logic $M = N$ does not imply $\langle x\rangle M = \langle x\rangle N$. This fact is independent from the particular abstraction mechanism adopted and it is actually related to the absence of a "canonical" choice for $\langle x\rangle M$ (see references).

From the point of view of computer science, the interest in Combinatory Logic derives more from implementation then from semantics. Indeed, β-conversion, as it is formulated in the λ-calculus, give rise to the well-known, conceptually simple, but syntactically fastidious problem of name clashes. For instance, $M \equiv (\lambda xy.x)y$ does not reduce to $\lambda y.y$, but to $\lambda z.y$. This kind of problems does not sussist in Cominatory Logic, which thus provides a convenient intermediate code where the λ-calculus can be compiled before execution. For example, the previous term M is compiled as:

$$\begin{aligned} M_{CL} &= ((\lambda xy.x)y)_{CL} \\ &= (\lambda xy.x)_{CL}\ (y)_{CL} \\ &= (\langle x\rangle Kx)y \\ &= S(KK)(SKK)y \end{aligned}$$

and its reduction, using, say, an innermost-leftmost strategy, yields:

$$\begin{aligned} S(KK)(SKK)y &= (KKy)(SKKy) \\ &= K(SKKy) \\ &= K((Ky(Ky)) \\ &= Ky\ . \end{aligned}$$

9.2 From Categories to Functionally Complete Applicative Structures

In this section, we suggest how to understand, in categorical terms, the difference between "functional completeness" and "lambda abstraction" and, later, characterize both notions, in absence of type constraints. As mentioned in the introduction, CL is the simplest type-free language that is functionally complete, since, for every term M, there exists $\langle x\rangle M$ that satisfies (abs). In case the choice of $\langle x\rangle M$ is "uniform in M", one has lambda abstraction and λ-calculus: i.e., $\langle x\rangle M$ is canonically given by $\lambda x.M$.

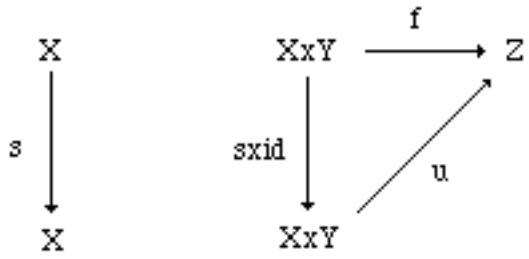
In order to give categorical meaning to this complex situation, we proceed as follows: we start with recovering applicative structures, in particular functionally complete ones, in Cartesian

categories (see 9.2.1-9.2.5); then we shift to the realm of Cartesian closed categories, where the existence of function spaces (exponents) allows a better understanding of the notion of functional completeness (9.2.6-9.2.7) and lambda-abstraction (9.2.8-9.2.12). In section 5, we will give a fully categorical characterization of models of these type-free calculi.

9.2.1 Definition *Let* $\mathbf{C}$ *be a Cartesian category,* T *its terminal object, and* U *an object in* $\mathbf{C}$*, with* $T < U$ *and* $u \in \mathbf{C}[U\times U,U]$*. The* ***applicative structure associated to u****,* $\underline{A}(u)$*, is given by* $\underline{A}(u) = (\mathbf{C}[T,U],\cdot)$*, where* $a\cdot b = u\circ\langle a,b\rangle$*.*

In a category with a terminal object T, $T < U$ simply generalizes the set-theoretic notion that U is "not empty". Clearly, $\underline{A}(u)$ is nontrivial (i.e., it contains at least two elements) iff $T < U$ is strict, i.e., is not an isomorphism.

9.2.2 Definition *Let* $\mathbf{C}$ *be a cartesian category. Then* $u \in \mathbf{C}[X\times Y,Z]$ *is* ***Kleene-universal*** *(K-universal) if* $\forall f \in \mathbf{C}[X\times Y,Z]\ \exists s \in \mathbf{C}[X,X]\ \ f = u\circ(s\times id)$ *, i.e.,*



Kleene-universality is a *weak* (co)universality property, since no unicity of s is required. It has an obvious recursion-theoretic meaning: indeed K-universality generalizes the s-m-n (iteration) theorem, with $X = Y = Z = \omega$ and with f a (total) recursive function, i.e., a morphism from $(\omega,id)\times(\omega,id)$ to (ω,id), in the category **EN** of numbered sets.

9.2.3 Definition *Let* $\mathbf{C}$ *be Cartesian and* $u \in \mathbf{C}[X\times X,X]$*. Then* $u^{(n)} \in \mathbf{C}[X\times X^n,X]$ *is inductively defined by* $u^{(0)} = id$*,* $u^{(n+1)} = u^{(n)}\circ(u\times id^n)$*, that is,*

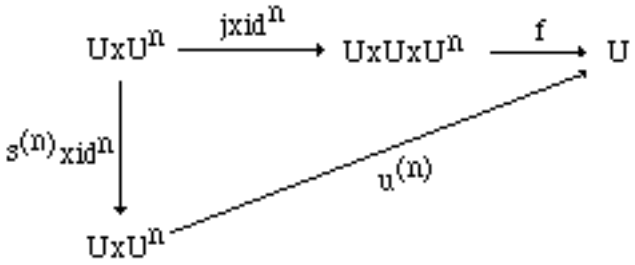
$$u^{(n+1)}\colon X\times X\times X^n \xrightarrow{u\times id^n} X\times X^n \xrightarrow{u^{(n)}} X\ ,\quad \text{where } X^{n+1} = X\times X^n.$$

It is easy to observe that $u^{(n)}$ corresponds exactly to the application of n+1 arguments, from left to right, e.g. $u^{(2)}\circ\langle a,b,c\rangle = u\circ(u\times id)\circ\langle a,b,c,\rangle = u\circ\langle u\circ\langle a.b\rangle,c\rangle$. We write $a\cdot b\cdot c$ for $(a\cdot b)\cdot c$.

9.2.4 Lemma *Let* $\mathbf{C}$ *be Cartesian. Assume that, for some* U *in* $\mathbf{C}$*,* $U\times U < U$ *and there is a K-universal* $u \in \mathbf{C}[U\times U,U]$*. Then* $\forall n\ u^{(n)} \in \mathbf{C}[U\times U^n,U]$ *is K-universal.*

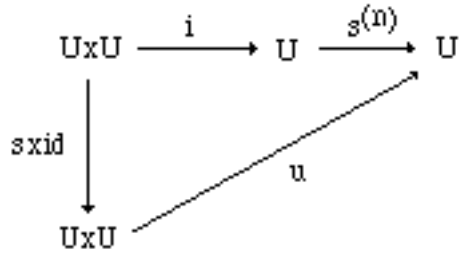
Proof By assumption, this is true for n = 1. Let $U\times U < U$ via (i,j) and $f\in\mathbf{C}[U\times U^{n+1},U]$. Then, by the inductive hypotesis, for some $s^{(n)}\in C[U,U]$ the following diagram commutes:

(1)



By assumption, for some $s\in\mathbf{C}[U,U]$ one also has

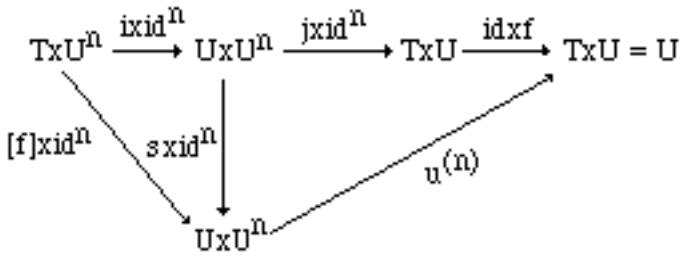
(2)



Then compute
$$\begin{aligned} f &= f\circ(j\times id^n)\circ(i\times id^n) \\ &= u^{(n)}\circ(s^{(n)}\times id^n)\circ(i\times id^n) && \text{by (1)} \\ &= u^{(n)}\circ(u\times id^n)\circ(s\times id^{n+1}) && \text{by (2)} \\ &= u^{(n+1)}\circ(s\times id^{n+1}).\ \blacklozenge \end{aligned}$$

9.2.5 Theorem *Let* ***C*** *be a Cartesian category. Assume that, for some object U, one has* $T < U$, $U\times U < U$ *and there exists a K-universal* $u\in\mathbf{C}[U\times U,U]$. *Then* $\underline{A}(u)$ *is a combinatory algebra.*

Proof Let $T < U$ via (i_T,j_T). Then, by lemma 9.2.4, $\forall n$, $\forall f\in\mathbf{C}[U^n,U]$ $\exists s\in\mathbf{C}[U,U]$ such that the following diagram commutes, with $[f] = s\circ i_T$ (we write i and j for i_T and j_T):



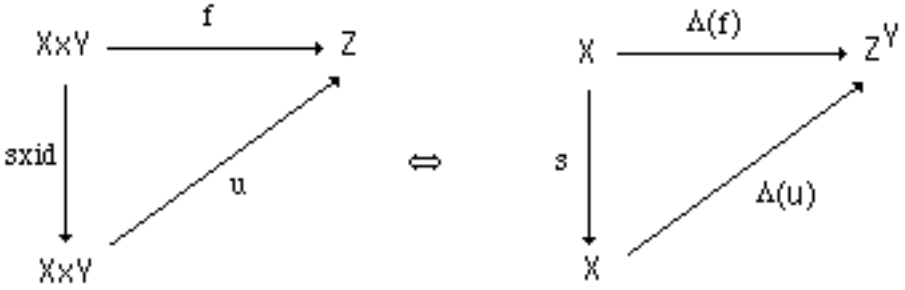
Thus, $u^{(n)}\circ[f]\times d^n = id_T\times f = f$. Since $u^{(n)}$ is the application, from left to right, of its n+1 arguments, [f] "represents" f, with respect to application. By 9.2.1, we only need to define $f\in\mathbf{C}[U^2,U]$ and $g\in\mathbf{C}[U^3,U]$ such that [f] and [g] represent K and S, respectively. For this purpose, take $f = pr^2_1\in\mathbf{C}[U^2,U]$ and $g = u\circ\langle u\circ\langle pr^3_1,pr^3_3\rangle,u\circ\langle pr^3_2,pr^3_3\rangle\rangle\in\mathbf{C}[U^3,U]$. ♦

Theorem 9.2.5 provides sufficient conditions in order to construct "functionally complete" objects. Theorem 9.5.6 will show that these conditions are also necessary.

A further insight, though, into functional completeness may be given in CCC's. The advantage of dealing with CCC's is that for any object X one may consider its "function space" or exponent X^X, as an object. As a matter of fact, functional completeness, in its various forms of increasing strength (combinatory algebras, λ-algebras, λ-models (see below)), expresses some sort of privileged relation between an object in a category and its "function space": its representability, say, in the sense expressed in the proof of theorem 9.2.5. In CCC's K-universal morphisms and principal morphisms are related as follows:

9.2.6 Proposition *Let* $\mathbf{C}$ *be a CCC and* Λ *be the isomorphism* $\mathbf{C}[X\times Y,Z] \cong \mathbf{C}[X,Z^Y]$. *Then* $u\in\mathbf{C}[X\times Y,Z]$ *is K-universal iff* $\Lambda(u)\in\mathbf{C}[X,Z^Y]$ *is principal.*

Proof The isomorphism Λ implies, by definition, the equivalence of the following diagrams:



and we are done. ♦

The connection between K-universal and principal morphisms should remind the reader that the latter correspond to (acceptable) Gödel numberings from ω to PR, the partial recursive functions, when these are viewed as objects in the category **EN** of numbered sets (see section 2.2). Note though that Kleene's $(\omega, \cdot)$ is a *partial* combinatory algebra and does not yield a model of Combinatory Logic. *Total* combinatory algebras turn out to be nontrivial constructions and may be obtained, for instance, in higher types, as it will be shown later. The categorical description, in terms of K-universal maps or principal morphisms, sheds some light on the connections between Gödel numberings and combinatory completeness. As a matter of fact, by proposition 9.2.6 one may then restate theorem 9.2.5 in terms of CCC's and principal morphisms.

9.2.7 Proposition *Let* $\mathbf{C}$ *be a CCC. Assume that, for some* U *in* $\mathbf{C}$, $T < U$, $U\times U < U$ *and there exists a principal* $p\in\mathbf{C}[U,U^U]$. *Then* $\underline{A}(\Lambda^{-1}(p))$ *is a combinatory algebra.*

The previous proposition suggests more explicitly the connection between the definition of application and the morphism eval in CCC's. Just observe that, by definition of "·", one has in a CCC

$$a\cdot b = \Lambda^{-1}(p)\circ\langle a,b\rangle = \mathrm{eval}\circ(p\times id)\circ\langle a,b\rangle = \mathrm{eval}\circ((p\circ a)\times b).$$

Informally, the equation above means "transform a into a morphism, by p, then apply it to b."

This process generalizes the way Gödel numberings associate functions to numbers. Similarly as for the partial recursive functions, there is in general no "canonical" way to go backwards, that is to choose uniformely and effectively a representative for each representable function. That is, this representative does not need to be unique and it is not possible to choose a representative for each representable function in a "uniform" way, i.e., by a morphism in the category. This is, though, possible in λ-models. We define them here in a first order manner, as particular combinatory algebras, with a suitable "choice" operator.

9.2.8 Definition *Let $A = (X, \cdot)$ be an applicative structure. Then*

i. A is a ***λ-model*** *if for some $k,s,\varepsilon \in X$ one has:*

$k.$ $\quad \forall x,y \quad kxy = x$;

$s.$ $\quad \forall x,y,z \quad sxyz = xz(yz)$;

$\varepsilon_1.$ $\quad \forall x,y \quad \varepsilon xy = xy$;

$\varepsilon_2.$ $\quad \forall x,y \quad (\forall z\ xz = yz) \Rightarrow \varepsilon x = \varepsilon y$;

$\varepsilon_3.$ $\quad \varepsilon\varepsilon = \varepsilon$.

ii. A is an ***extensional*** *λ-model if one also has $\forall x \ \varepsilon x = x$.*

ε has to be understood as a choice operator that picks up a canonical representative for each representable function. ε coincides with the canonical representative of the function it represents, by axiom (ε_3). In extensional λ-models, there is just one representative and $(\square\varepsilon_1)$, $(\square\varepsilon_3)$ are derived. Note that $A = (X, \cdot)$ is an extensional λ-model iff A is a combinatory algebra and $\forall x,y \ (\forall z\ xz = yz) \Rightarrow x = y$.

There exists an obvious formal system of combinators K, S, ε associated to the previous notion of λ-model, which we shall call **CLε** (we gave priority to the notion of model because we mainly focus here on semantical aspects). The interpretation of CLε in λ-models is straighforward. Note also that CLε may be easily and soundly translated into λ-calculus, by taking ε to λxy.xy.

Conversely, the combinator ε can be used to "clean" the translation of a lambda term by means of combinators described in definition 9.1.4 :

9.2.9 Definition *Given a λ-term M, the associated term $M_{CL\varepsilon}$ in CLε is inductively defined by*

$$x_{CL\varepsilon} = x$$
$$(MN)_{CL\varepsilon} = M_{CL\varepsilon}N_{CL\varepsilon}$$
$$(\lambda x.M)_{CL\varepsilon} = \varepsilon \cdot \langle x \rangle M_{CL\varepsilon}.$$

This "refinement" is completely worthless from an implementative point of view, since the reduction process is essentially unaffected by the combinator ε, as it is stated by equation (ε_1). On the contrary,

it is relevant in semantics, since it allows a simple definition of a sound interpretation of λ–terms in λ-models, as follows:

9.2.10 Definition *Let $A = (X, \cdot, k, s, \varepsilon)$ be a λ-model. The interpretation $[\![M]\!]\xi$ of a λ-term M in A with respect to an environment ξ is the semantics of the associated combinatorial term $M_{CL\varepsilon}$, i.e.,*

$$[\![M]\!]\xi = [M_{CL\varepsilon}]\xi.$$

We omit the soundness proof, which is technically straightforward and almoust evident from the previous discussions.

In the next two results we show how to derive λ-models from reflexive objects in categories with enough points.

9.2.11 Theorem *Let $\mathbf{C}$ be a CCC with enough points. Assume that, for some U in $\mathbf{C}$, one has $U^U < U$ via (ψ, ϕ). Then, for $\varepsilon = \psi\circ\Lambda(\psi\circ\phi\circ snd)$, $\underline{A} = (\underline{A}(\Lambda^{-1}(\phi)),\varepsilon)$ is a λ-model.*

Proof $\phi \in \mathbf{C}[U, U^U]$ is principal; moreover by 2.3.6, $T < U$ and $U\times U < U$. Thus, for $a\cdot b = eval\circ\langle(\phi\circ a), b\rangle = eval\circ(\phi\times id)\circ\langle a,b\rangle$ and some suitable K and S, $(\underline{A}(\Lambda^{-1}(\phi)),\cdot,K,S)$ is a combinatory algebra, i.e. (k) and (s) in 9.2.8 hold. Define now $\varepsilon = \psi\circ\Lambda(\psi\circ\phi\circ snd) : T\to U$ (that is, informally, $\varepsilon = \psi(\psi\circ\phi)$). Note first that, for any a,

(†) $\qquad \varepsilon \cdot a = \psi \circ \phi \circ a$

indeed:

$\varepsilon\cdot a$	$= (\psi\circ\Lambda(\psi\circ\phi\circ snd))\cdot a$	by def. of ε
	$= eval\circ\langle(\phi\circ\psi\circ\Lambda(\psi\circ\phi\circ snd)),a\rangle$	by def. of "·"
	$= eval\circ\langle(\Lambda(\psi\circ\phi\circ snd)),a\rangle$	since $\phi\circ\psi = id$
	$= eval\circ(\Lambda(\psi\circ\phi\circ snd)\times id)\circ\langle id\times a\rangle$	
	$= \psi\circ\phi\circ snd\circ\langle id\times a\rangle$	
	$= \psi\circ\phi\circ a$	

Then one has:

ε_1.	$\varepsilon\cdot a\cdot b$	$= (\psi\circ\phi\circ a)\cdot b$	by (†)
		$= eval\circ\langle(\phi\circ\psi\circ\phi\circ a),b\rangle$	by def. of "·"
		$= eval\circ\langle(\phi\circ a),b\rangle$	since $\phi\circ\psi = id$
		$= a\cdot b$	by def. of "·"

ε_2. Suppose that $\forall z\ az = bz$. Then, since

$$a\cdot z = eval\circ\langle(\phi\circ a),z\rangle = eval\circ((\phi\circ a)\times id)\circ\langle id,z\rangle,$$
$$b\cdot z = eval\circ\langle(\phi\circ b),z\rangle = eval\circ((\phi\circ b)\times id)\circ\langle id,z\rangle,$$

and since $\mathbf{C}$ has enough points, we have $eval\circ((\phi\circ a)\times id) = eval\circ((\phi\circ b)\times id)$, and thus $\phi\circ a = \phi\circ b$. Then $\varepsilon\cdot a = \psi\circ\phi\circ a = \psi\circ\phi\circ b = \varepsilon\cdot b$.

ε_3. $\quad \varepsilon\cdot\varepsilon = \psi\circ\phi\circ\psi\circ\Lambda(\psi\circ\phi\circ snd) = \psi\circ\Lambda(\psi\circ\phi\circ snd) = \varepsilon$. ♦

The definition of ε should be clear. Just note that $\psi\circ\phi : U\to U^U\to U$, i.e., ϕ gives a morphism for each point a in U and ψ chooses a "canonical" one, representing $\phi(a)$, as $\phi\circ\psi = id$. Then $\varepsilon = \psi\circ\Lambda(\psi\circ\phi\circ snd): T\to U$, internalizes $\psi\circ\phi$ as a point in U.

9.2.12 Corollary *Let* $\mathbf{C}$ *be a CCC with enough points.*

i. If, for some U in $\mathbf{C}$, $U^U < U$ *via (i,j) and there exists* $u\in\mathbf{C}[U\times U,U]$ *K-universal, then also* $\underline{A}(u)$ *can be turned into a* λ*-model.*

ii. If $U^U \cong U$ *via* (ψ,ϕ), *then* $\underline{A}(u)$ *is an extensional* λ*-model.*

Proof

i. By theorems 9.2.6 and 9.2.11 and the definiton of principal morphism.

ii. $\varepsilon\cdot a = \psi\circ\phi\circ a = a$. ♦

9.3 Categorical Semantics of the λ-Calculus

In theorem 9.2.11 we proved that if $\mathbf{C}$ is a CCC with enough points, and $U\in Ob_{\mathbf{C}}$ is a reflexive object (i.e., $U^U<U$ via (ψ, ϕ)) then, for $\varepsilon=\psi\circ\Lambda(\psi\circ\phi\circ snd) : T\to U$, $(\underline{A}(\Lambda^{-1}(\phi)),\varepsilon)$ is a λ-model. We can thus give an interpretation of the lambda calculus as in definition 9.2.10.

In this section we define a more direct interpretation of the lambda calculus over such an object U, and relate the two interpretations.

9.3.1 Definition *Let* $\mathbf{C}$ *be a CCC with terminal object T. Let* $U\in Ob_{\mathbf{C}}$ *be a reflexive object via the retraction pair* $(\psi: U^U\to U, \phi: U\to U^U)$. *Let M be a* λ*-term with* $FV(M) \subseteq \Delta = \{x_1,\ldots,x_n\}$. *Define then* $[M]_\Delta\in\mathbf{C}[U^n,U]$, *where* $U^n = (\ldots(T\times U)\times\ldots)\times U$ *with n copies of U, as follows (we use the two projections fst and snd in a polymorphic fashion, and we omit the indexes):*

$$[x_i]_\Delta = snd\circ fst^{n-i} = pr^n_i$$
$$[MN]_\Delta = eval\circ\langle\phi\circ[M]_\Delta, [N]_\Delta\rangle$$
$$[\lambda x.M]_\Delta = \psi\circ\Lambda([M]_{\Delta\cup\{x\}}).$$

We do not prove the soundness of the interpretation; the reader interested in this result may consult the references.

Examples

1. Let $M = \lambda x.xx$.

$$[\lambda x.xx] = \psi\circ\Lambda([xx]_{\{x\}})$$

$= \psi \circ \Lambda(\text{eval} \circ < \phi \circ [x]_{\{x\}}, [x]_{\{x\}} >)$

$= \psi \circ \Lambda(\text{eval} \circ < \phi \circ \text{snd}, \text{snd} >) : T \to U$.

2. Consider the term $Y = \lambda x.(\lambda y.x(yy)(\lambda y.x(yy))$. This is a fixpoint operator, since for every M,

$YM = (\lambda x.(\lambda y.x(yy)(\lambda y.x(yy)))M$

$= (\lambda y.M(yy)(\lambda y.M(yy))$

$= M((\lambda y.M(yy))(\lambda y.M(yy)))$

$= M(YM)$

Let us interpret Y. We proceed by stages

$[\lambda y.x(yy)]_{\{x\}} = \psi \circ \Lambda([x(yy)]_{\{x,y\}})$

$= \psi \circ \Lambda(\text{eval} \circ < \phi \circ [x]_{\{x,y\}}, [yy]_{\{x,y\}} >)$

$= \psi \circ \Lambda(\text{eval} \circ < \phi \circ \text{snd} \circ \text{fst}, \text{eval} \circ < \phi \circ [y]_{\{x,y\}}, [y]_{\{x,y\}} > >)$

$= \psi \circ \Lambda(\text{eval} \circ < \phi \circ \text{snd} \circ \text{fst}, \text{eval} \circ < \phi \circ \text{snd}, \text{snd} > >)$

Let $P = \psi \circ \Lambda(\text{eval} \circ < \phi \circ \text{snd} \circ \text{fst}, \text{eval} \circ < \phi \circ \text{snd}, \text{snd} > >)$.

Then we have

$[Y] = [\lambda x.(\lambda y.x(yy)(\lambda y.x(yy))]$

$= \psi \circ \Lambda([(\lambda y.x(yy)(\lambda y.x(yy)))]_{\{x\}})$

$= \psi \circ \Lambda(\text{eval} \circ < \phi \circ P \circ \text{snd}, P \circ \text{snd} >): T \to U$

It is not difficult to prove that $\Lambda^{-1}(\phi \circ [Y]) \circ <!U^U, \psi> = \text{eval} \circ < \phi \circ P \circ \psi, P \circ \psi > : U^U \to U$ is a categorical fixpoint operator (see definition 8.8.2).

We now relate the two notions of categorical semantics given in 9.2.10 and 9.3.1. As a matter of fact they are essentially equivalent, the only difference being that the one in definition 9.3.1 does not need the concept of environment.

9.3.2 Definition *Let ξ: Var$\to \underline{A}(\Lambda^{-1}(\phi))$ be an environment . Let $\Delta = \{x_1,\ldots,x_n\}$ be a finite set of variables. Then*

$$\xi'_\Delta = <\ldots<id_T, \xi(x_1)>\ldots ,\xi(x_n)> : T \to T\times U\ldots\times U$$

We shall usually omit the subscript Δ in ξ'_Δ when it will be clear from the context.

9.3.3 Theorem *Let **C** be a CCC with enough points, $U^U < U$ via (ψ, ϕ)), and $\underline{A}$ be the associated λ-model as in proposition 9.2.11. Let $[\,]$ and $[\![\,]\!]$ be the interpretations respectively defined in 9.2.10 and 9.3.1. Then, for any term M with free variables in $\Delta = \{x_1,\ldots,x_n\}$ and any environment ξ: Var$\to \underline{A}(\Lambda^{-1}(\phi))$, one has*

$$[M]_\Delta \circ \xi'_\Delta = [\![M]\!]\xi.$$

Proof The proof is by induction on the structure of M.

- case M = x. Suppose $\xi(x) = a : T \to U$. Then $\xi'_\Delta = < \text{id}_T, a >$. We have:

$[x]_{\{x\}} \circ \xi'_\Delta = \text{snd} \circ < \text{id}_T, a >$

$$= a$$
$$= \xi(x)$$
$$= [\![x]\!]\xi$$

- case $M = PQ$

$$[PQ]_\Delta \circ \xi'\Delta = \text{eval} \circ \langle \phi \circ [P]_\Delta, [Q]_\Delta \rangle \circ \xi'\Delta$$
$$= \text{eval} \circ \langle \phi \circ [P]_\Delta \circ \xi'\Delta, [Q]_\Delta \circ \xi'\Delta \rangle$$
$$= \text{eval} \circ \langle \phi \circ [\![P]\!]\xi, [\![Q]\!]\xi \rangle \quad \text{by induction hypothesis}$$
$$= [\![P]\!]\xi\ [\![Q]\!]\xi \quad \text{by def. of application}$$
$$= [\![PQ]\!]\xi$$

- case $M = \lambda x_{n+1}.N$

Note first that, for any $a : T \to U$,

$$(*) \qquad [\langle x_{n+1} \rangle N CL\varepsilon]\xi \cdot a = [N]_{\Delta \cup \{x_{n+1}\}} \circ \langle \xi'\Delta, a \rangle.$$

Indeed:

$$[\langle x_{n+1} \rangle N CL\varepsilon]\xi \cdot a = [N CL\varepsilon]\xi(x_{n+1}=a) \quad \text{by lemma 14.1.3}$$
$$= [\![N]\!]\xi(x_{n+1}=a) \quad \text{by definition of } [\![\ ]\!]$$
$$= [N]_{\Delta \cup \{x_{n+1}\}} \circ \langle \xi'\Delta, a \rangle \quad \text{by induction hypothesis.}$$

Note now that:

$$[\langle x_{n+1} \rangle N CL\varepsilon]\xi \cdot a = \text{eval} \circ \langle \phi \circ [\langle x_{n+1} \rangle N CL\varepsilon]\xi, a \rangle$$
$$= \text{eval} \circ ((\phi \circ [\langle x_{n+1} \rangle N CL\varepsilon]\xi) \times \text{id}) \circ \langle \text{id}_T, a \rangle,$$

and

$$[N]_{\Delta \cup \{x_{n+1}\}} \circ \langle \xi'\Delta, a \rangle = [N]_{\Delta \cup \{x_{n+1}\}} \circ \xi'\Delta \times \text{id} \circ \langle \text{id}_T, a \rangle.$$

Since **C** has enough points, and all the points in $T \times U$ are of the kind $\langle \text{id}_T, a \rangle$, from (*) we have:

$$[N]_{\Delta \cup \{x_{n+1}\}} \circ \xi'\Delta \times \text{id} = \text{eval} \circ ((\phi \circ [\langle x_{n+1} \rangle N CL\varepsilon]\xi) \times \text{id}).$$

Applyng Λ to both the members, and composing with ψ, we get:

$$\psi \circ \Lambda([N]_{\Delta \cup \{x_{n+1}\}} \circ \xi'\Delta) = \psi \circ \phi \circ [\langle x_{n+1} \rangle N CL\varepsilon]\xi.$$

By definition, $\psi \circ \Lambda([N]_{\Delta \cup \{x_{n+1}\}} \circ \xi'\Delta = [\lambda x.N]_\Delta \circ \xi'\Delta$

Moreover,

$$[\![\lambda x.N]\!]\xi = [(\lambda x_{n+1}.N) CL\varepsilon]\xi$$
$$= [\varepsilon \cdot \langle x_{n+1} \rangle N CL\varepsilon]\xi$$
$$= \varepsilon \cdot [\langle x_{n+1} \rangle N CL\varepsilon]\xi$$
$$= \psi \circ \phi \circ [\langle x_{n+1} \rangle N CL\varepsilon]\xi.$$

This conludes the proof. ♦

9.4 The Categorical Abstract Machine

The categorical interpretation in definition 9.3.1 suggests a very simple and nevertheless efficient implementation of the lambda calculus. The implementation is based on a call-by-value, leftmost strategy of evaluation, and it is performed by an abstract environment machine called CAM (see refernces). The first step toward the implementation is the compilation of lambda calculus in a language of **categorical combinators.**

Note that $[MN]_\Delta = eval \circ < \phi \circ [M]_\Delta, [N]_\Delta> = \Lambda^{-1}(\phi) \circ < [M]_\Delta, [N]_\Delta>$. $\Lambda^{-1}(\phi)$: $U\times U \rightarrow U$ is just the application u of the underlyng combinatory algebra. We shall write **app** instead $\Lambda^{-1}(\phi)$. Moreover, let **cur**(f) = $\psi \circ \Lambda(f)$, and write f ; g instead of g ∘ f. Then the equations which define the semantic interpretation of the lambda calculus are rewritten as follows:

$[x_i]_\Delta$ = fst; . . . ; fst; snd where fst appears n-i times

$[MN]_\Delta = < [M]_\Delta, [N]_\Delta>$; app

$[\lambda x.M]_\Delta =$ cur($[M]_{\Delta\cup\{x\}}$).

This provides a "compilation" of the λ–calculus in a language where all the variables have been replaced with "access paths" to the information they refer to (note the use of the "dummy" enviroment Δ during the compilation).

One of the main characteristic of the categorical semantical approach is that we can essentially use the same language for representing both the code and the environment. An evaluation of the code C in an environment ξ is then the process of reduction of the term ξ ; C. The reduction is defined by a set of rewriting rules. The general idea is that the environment should correspond to a categorical term in some normal form (typically, a weak head normal form). The reductions preserve this property of the environment, executing one instruction (i.e. one categorical combinator) of the code, and updating at the same time the program pointer to the following instruction.

For fst and snd we have the following rules, whose meaning is clear:

$<\alpha,\beta>$; (fst ; C_1) ⇒ α ; C_1

$<\alpha,\beta>$; (snd ; C_1) ⇒ β ; C_1

In the left hand side of the previous rules, $<\alpha,\beta>$ is the environment and the rest is the code. We shall use parenthesis in such a way that the main semicolon in the expression will distinguish between the environment at its left, and the code at its right.

For cur(C_1) we use the associative law of composition and delay the evaluation to another time:

ξ ; (cur(C_1); C_2) ⇒ (ξ ; cur(C_1)) ; C_2

The structure (ξ ; cur(C_1)) corresponds to what is usually called a **closure**.

The right time for evaluating a term of the kind cur(C) is when it is applied to an actual parameter α. We then have:

$<(\xi$; cur(C_1)), α > ; (app; C_2) ⇒ $<\xi, \alpha>$; (C_1; C_2)

The previous rule is just a rewriting of the equation

$\Lambda^{-1}(\phi) \circ <\psi \circ \Lambda(C_1) \circ \xi, \alpha > = eval \circ <\Lambda(C_1) \circ \xi, \alpha > = C_1 \circ <\xi, \alpha>$

that proves the semantical soundness of the previous rule.
Finally, we must consider the evaluation of a term of the kind $\langle C_1,C_2\rangle; C_3$. We have the formal equation:

$$\xi ; (\langle C_1,C_2\rangle; C_3) = \langle \xi ; C_1, \xi ; C_2\rangle ; C_3$$

but we cannot simply use it for defining a reduction, since we want also to reduce $\xi ; C_1$ and $\xi ; C_2$. We must first carry out independently the reductions of $\xi ; C_1$ and $\xi ; C_2$, and then put them together again building the new environment.
A simple solution on a sequential machine may be given by using a stack and working as follows: first save the actual environment ξ by a *push* operation, then evaluate $\xi ; C_1$ (that yields a new environment ξ_1); next *swap* the environment ξ_1 with the head of the stack (i.e. with ξ); now we can evaluate $\xi ; C_2$ obtaining ξ_2; finally build a pair $\langle \xi_1,\xi_2\rangle$ with the head of the stack ξ_1 and the actual environment ξ_2 (that is a *cons* operation). An interesting and elegant property is that, if we just write at compile time $\langle C_1,C_2\rangle$ as "push; C_1; swap; C_2; cons", then the above behaviour is obtained by a sequential execution of this code.

9.4.1 Definition *The **compilation** by means of categorical combinators of a λ-term M in a "dummy" environment $\Delta = (...(nil, x_1),\ldots), x_n)$ is inductively defined as follows:*

$$[x]_{(\Delta,x)} = snd$$
$$[y]_{(\Delta,x)} = fst; [y]_{\Delta}$$
$$[MN]_{\Delta} = push; [M]_{\Delta}; swap; [N]_{\Delta}; cons; app$$
$$[\lambda x.M]_{\Delta} = cur([M]_{(\Delta,x)}) .$$

Examples

1. The closed term $M = \lambda x.xx$ has the following compilation:

$$\begin{aligned}[\lambda x.xx]_{nil} &= cur([xx]_{(nil,x)}) \\ &= cur(push; [x]_{(nil,x)}; swap; [x]_{(nil,x)}; cons; app) \\ &= cur(push; snd; swap; snd; cons; app).\end{aligned}$$

2. The term $(\lambda x.x)(\lambda x.x)$ is so compiled:

$$\begin{aligned}[(\lambda x.x)(\lambda x.x)]_{nil} &= push; [\lambda x.x]_{nil}; swap; [\lambda x.x]_{nil}; cons; app \\ &= push; cur([x]_{(nil,x)}); swap; cur([x]_{(nil,x)}); cons; app \\ &= push; cur(snd); swap; cur(snd); cons; app.\end{aligned}$$

9.4.2 Definition *The **reduction** of the compiled code is summarized by the following table:*

BEFORE			AFTER		
Environment	Code	Stack	Environment	Code	Stack
$<\alpha,\beta>$	fst; C	S	α	C	S
$<\alpha,\beta>$	snd; C	S	β	C	S
ξ	cur(C_1); C_2	S	ξ; cur(C_1)	C_2	S
$<\xi$;cur(C_1),$\alpha>$	app; C_2	S	$<\xi,\alpha>$	C_1;C_2	S
ξ	push; C	S	ξ	C	ξ.S
ξ_1	swap; C	ξ_2.S	ξ_2	C	ξ_1.S
ξ_1	cons; C	ξ_2.S	$<\xi_2, \xi_1>$	C	S.

Example The code "push; cur(snd); swap; cur(snd); cons; app" corresponding to the λ-term $(\lambda x.x)(\lambda x.x)$ gives rise to the following computation:

ENV. = nil
CODE = push; cur(snd); swap; cur(snd); cons; app
STACK = nil

ENV. = nil
CODE = cur(snd); swap; cur(snd); cons; app
STACK = nil . nil

ENV. = nil; cur(snd)
CODE = swap; cur(snd); cons; app
STACK = nil . nil

ENV. = nil
CODE = cur(snd); cons; app
STACK = nil; cur(snd) . nil

ENV. = nil; cur(snd)
CODE = cons; app
STACK = nil; cur(snd) . nil

ENV. = <nil; cur(snd), nil; cur(snd) >
CODE = app
STACK = nil

ENV. = <nil, nil; cur(snd) >
CODE = snd
STACK = nil

ENV. = nil; cur(snd)
CODE =
STACK = nil

Note that "cur(snd)" is the compilation of $\lambda x.x$.

9.5 From Applicative Structures to Categories

We want now to "go backwards", with respect to section 9.2, where we described how to find models of a type-free language within type structures, i.e., within CCC. Namely, we will see how to construct typed models, in particular a CCC, out of type-free structures. This has an important motivation from Programming Language Theory, since it is the semantic counterpart of the following relevant methodology in functional languages.

We already mentioned, as an example, that one of the main features of Edinburgh ML is its automatic treatment of type assignment. That is, the programmer may write programs without taking care of the tedious details of assigning types. The type checker decides whether the given program is typable and, if so, assigns a type to it (actually, the "most general type").

This effective interactive feature of ML provides a partial check for correctness, as one may automatically control whether type errors occur. This is similar to what physicists call "dimensional analysis" for equations when they verify, say, whether a force faces a force, etc. Of course, a lot must be settled. For example, the actual decidability of the type assignment and the existence of "type schemes" such that all types of a given program are instances of these. The identity function, for example, has type $A \to A$ for all instances of A.

As for the semantics, one must first be able to interpret the type-free language, as handled by the programmer, and then interpret types as objects of suitable CCC constructed over the type-free model. In other words, one must be able to obtain an interpretation of types out of a model for the type-free calculus. "Soundness" then means that a program, once it is assigned a type, is actually interpreted as an "element" of the interpretation of its type. Decidability and soundness have been positively clarified by a mathematical investigation of computability and programming, which goes beyond the scope of this book (see references).

Our present purpose is to survey the main "type structures" (categories) one may construct out of type-free models and to complete, in this way, the categorical understanding of typed versus type-free calculi, as required for the semantics of the type assignment process. Most of the work may be done over an arbitrary combinatory algebra (X, .), i.e., over an arbitrary model of Combinatory Logic. Indeed, it is even not required that "·" , the application in X, is a total operation. As already mentioned, if "·" is not always defined, (X, ·) is no longer a model of CL. However, the categories constructed below still have the same properties (products, exponents, whenever possible...), which the reader should check as an exercise.

9.5.1 Definition *Let* $A = (X,\cdot)$ *be an applicative structure.*

i. The set of ***monomials*** *over* A *is inductively defined by*

$x,y....x_1,x_2...$*(variables)...are monomials*

$a,b....a_1,a_2....$*(constants from* X*)... are monomials*

MN *is a monomial if* M *and* N *are monomials.*

Substitution of constants for variables, i.e. $M[\underline{a}/\underline{x}]$*, in monomials is defined by induction in the usual way.* $M_1M_2...M_n$ *stands for* $(...(M_1M_2)...M_n)$.

ii. $f: X^n \to X$ *is* ***algebraic*** *if* $f(\underline{a}) = M[\underline{a}/\underline{x}]$ *for some monomial* M *and any* $\underline{a} = (a_1,... ,a_n) \in X^n$ *and* $\underline{x}$ *of length* n*. (That is, the set* $\mathbf{P}^n(A) = \mathbf{P}[X^n,X]$ *of algebraic functions of n-arguments is defined by the monomials over* X*, with at most* n *variables, modulo extensional equality.)*

iii. Given $(X,\cdot)$*, call* $f : X^n \to X$ ***representable*** *if* $\exists a \in X\ \forall \underline{b} \in X^n\ f(\underline{b}) = a\cdot b_1\cdot...\cdot b_n$.

By using algebraic functions, one may define a simple category over an arbitrary applicative structure.

9.5.2 Definition *Let* $A = (X, \cdot)$ *be an applicative structure. The category* $\mathbf{P}_A$ *of* ***polynomials over*** A*, has as*

objects*:* $X^n \in \mathbf{P}_A$*, for all* $n \in \omega$*;*

morphisms*:* $f \in \mathbf{P}_A[X^n,X^m]$ *iff* $f: X^n \to X^m$ *and* $\forall i<m\ pr^m_i \circ f \in P^n$*, with* pr^m_i *i-th projection.*

(If there is no ambiguity write $\mathbf{P}[X^n,X^m]$ *for* $\mathbf{P}_A[X^n,X^m]$*).*

For example, f(x,y) = (xb(xax),yxa) for a,b∈X, is in $\mathbf{P}[X^2,X^2]$. By substitution, one may easily show that morphisms are closed under composition; moreover, $pr^n_i \in \mathbf{P}[X^n,X] = \mathbf{P}^n$ and, thus, $\mathbf{P}_A$ is a category.

Exercise (Curry-Shoenfinkel) Prove that exactly in combinatory algebras every algebraic function is representable (*hint:* use the argument which translates a λ-term λx.M into an S-K-term <x>M in the introduction).

If A is a combinatory algebra, then, by the exercise, $\mathbf{P}_A$ may be considered the category of representable morphisms.

9.5.3 Lemma *Let $A = (X,\cdot)$ be an applicative structure. Then $\boldsymbol{P}_A$ is a cartesian category with enough points. Moreover, if $\boldsymbol{C}$ is a CC with enough points and $\boldsymbol{C}$, U and $\underline{A}(u)$ are as in definition 9.2.1, then $\boldsymbol{P}_{\underline{A}(u)}$ is a full sub-cartesian category of $\boldsymbol{C}$.*

Proof. Set $X^n \times X^m = X^{n+m}$ and $T = X^0$ (= a singleton set) for the terminal object. The projections pr_i's are given above. Clearly, $\mathbf{P}_A$ has enough points. The rest easily follows from definition 9.2.1 and the assumption that $\mathbf{C}$ has enough points. The reader may complete the proof as an exercise. ♦

Given a category $\mathbf{C}$, U and $\underline{A}(u)$ as in lemma 9.5.3, we say that $g \in \mathbf{C}[U^n,U]$ **induces** $f : \underline{A}(u) \to \underline{A}(u)$ if $f(h) = g \circ h$ for all $h \in \mathbf{C}[T,U]$. It is straightforward to prove that all algebraic functions defined by a monomial in n variables over $\underline{A}(u)$, and no constants, are induced by morphisms in $\mathbf{C}[U^n,U]$. One only has to interpret variables as projections (see section 9.3) and argue by induction on the structure of the "algebraic term" defining the function. For example, for $f(x_1,x_2,x_3) = (x_3 \cdot x_1) \cdot x_2$ write

$u \circ \langle pr_3, pr_1 \rangle : U^3 \to U \times U \to U$, which is $x_3 \cdot x_1$, and then

$u \circ \langle u \circ \langle pr_3, pr_1 \rangle, pr_2 \rangle : U^3 \to U \times U \to U$, which induces f.

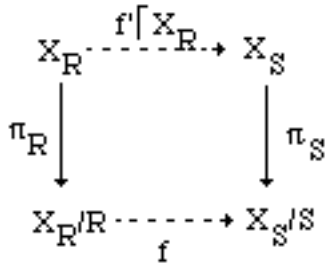
Next, we generalize a definition of category given over a specific applicative structure in example 3.4.1. The definition is slightly different (besides being more general). Since it is an important construction, it is worth seeing it again, under a different and more general viewpoint.

9.5.4 Definition *Let $A = (X,\cdot)$ be an applicative structure. Define then:*

*1. The category $\boldsymbol{PER}_A$ of **partial equivalence relations** given by:*

***objects**: $R \in \boldsymbol{PER}_A$ iff R is an equivalence relation on a subset X_R of X, i.e., X_R = dom R = range R.*

***morphisms**: for $R \in \boldsymbol{PER}_A$ let $\pi_R(n) = \{m \mid nRm\}$; then $f \in PER[R,S]$ iff $\exists f' \in \boldsymbol{P}[X,X]$ $f \circ \pi_R = \pi_S \circ f'$ on X_R, i.e., the following diagram commutes:*



*(we then say that f' **computes** f).*

2. The category $\boldsymbol{ER}_A$ *of (total)* ***equivalence relations*** *is given as above by using equivalence relations on* X *(i.e.,* $X_R = X$ *in 1.).*

$\mathbf{PER}_\mathrm{A}$ and $\mathbf{ER}_\mathrm{A}$ are clearly categories. Similarly as for $\mathbf{P}_\mathrm{A}$ we write (**P**)**ER**[R,S] for (**P**)$\mathbf{ER}_\mathrm{A}$[R,S] when unambiguous.

Exercise Let A be an applicative structure. Give a terminal object for $\mathbf{PER}_\mathrm{A}$ and $\mathbf{ER}_\mathrm{A}$, and prove that they have enough points. (*Hint*: recall that the constant functions are algebraic).

9.5.5 Proposition *Let* $A = (X,\cdot)$ *be an applicative structure. Then, if* $X\times X < X$ *in* $\boldsymbol{P}_A$, $\boldsymbol{ER}_A$ *and* $\boldsymbol{PER}_A$ *are CCs (with enough points). Moreover,* $\boldsymbol{P}_A$ *and* $\boldsymbol{ER}_A$ *are full sub-CC's of* $\boldsymbol{PER}_A$.

Proof Let $X\times X < X$ via $([-,-],\langle p_1,p_2\rangle)$. Then $R\times S$ may be defined componentwise, by

$$a(R\times S)b \text{ iff } (p_1(a))R(p_1(b)) \text{ and } (p_2(a))S(p_2(b)).$$

This turns $\mathbf{ER}_\mathrm{A}$ and $\mathbf{PER}_\mathrm{A}$ into CC's.

Observe now that $\forall n\ X^n < X$ via $([-,...,-],\langle p_1,...,p_n\rangle)$ in $\mathbf{P}_\mathrm{A}$, by iterating $X\times X < X$. Thus $\mathbf{P}_\mathrm{A}$ may be faithfully embedded in $\mathbf{PER}_\mathrm{A}$ by taking, for each X^n, the identity relation restricted to the image of X^n in X via $[-,...,-]$. Call this restricted identity id_n. Moreover, $\mathbf{P}_\mathrm{A}$ is full in $\mathbf{PER}_\mathrm{A}$, since $\mathbf{P}[X^n,X^m] \cong \mathbf{PER}[\mathrm{id}_n, \mathrm{id}_m]$, as sets, by the following isomorphism G (take m = 1, for the sake of simplicity). Let $g\in\mathbf{P}[X^n,X]$, then, for $x = [x_1,...,x_n]$, define $G(g)\in\mathbf{PER}[\mathrm{id}_n, \mathrm{id}]$ by $G(g)(x) = g(p_1(x),...,p_n(x)) = g(x_1,...,x_n)$. G(g) is computed, in the sense of 9.5.4, by $g\circ\langle p_1,... ,p_n\rangle : X \to X^n \to X$.

G is an isomorphism, whose reverse map is given as follows: if $h\in\mathbf{PER}[\mathrm{id}_n, \mathrm{id}]$ is computed by $h'\in\mathbf{P}[X,X]$, then $G^{-1}(h) = h'\circ[-,... ,-] : X^n \to X \to X$. By definition, $\mathbf{ER}_\mathrm{A}$ is a full sub-CC of $\mathbf{PER}_\mathrm{A}$, and both categories have enough points, by the exercise. ♦

The next theorem proves the converse of theorem 9.2.5 and, moreover, it shows that, by applying the construction in definition 9.2.1 and theorem 9.2.5 to a combinatory algebra, one gets back to the given combinatory algebra.

9.5.6 Theorem *Let* $A = (X,\cdot)$ *be a combinatory algebra and* $\boldsymbol{P}_A$ *be the category of polynomials over A. Then* $T < X$, $XxX < X$ *in* $\boldsymbol{P}_A$ *and, for* $u(x,y) = x\cdot y$, $u\in\boldsymbol{P}[X^2,X]$ *is K-universal in the category* $\boldsymbol{P}_A$. *Moreover,* $\underline{A}(u) = A$.

Proof $T < X$ trivially holds, for $X\neq\emptyset$. Clearly, XxX exists in $\mathbf{P}_\mathrm{A}$, by lemma 9.5.3. Let then $c,c_1,c_2\in X$ represent $\underline{\lambda}xyz.zxy$, $\underline{\lambda}xy.x$, $\underline{\lambda}xy.y$, respectively, in the sense of definition 9.5.1(iii). c is the element that codes pairs (they are commonly coded in this way in λ-calculus), while c_1, c_2 will be used to define projections. Thus, for $[x,y] = cxy$ and $p_i(x) = xc_i$, one has $[-,-]\in\mathbf{P}[X^2,X]$, $p_i\in\mathbf{P}[X,X]$ and $X\times X < X$ via $([-,-],\langle p_1,p_2\rangle)$. Finally, assume that $f\in\mathbf{P}[X^2,X]$ and that $a\in X$

represents f. Then $f = u\circ((\underline{\lambda}x.ax)\times id)$ and, hence, u is K-universal. It is easy to check from the definition that $\underline{A}(u) = A$. ♦

9.5.7 Corollary *Let $A = (X,\cdot)$ be an applicative structure. Then A is a combinatory algebra iff, in $\boldsymbol{P}_A$, one has $T < X$, $X\times X < X$ and, for $u(x,y) = x\cdot y$, u is K-universal.*
Proof ($\Rightarrow$) by theorem 9.5.6; ($\Leftarrow$) by theorem 9.2.6. ♦

As already pointed out, one needs CCC's in order to take care of λ−models. However, also combinatory algebras are tidely characterized within CCC's. The following immediate consequence of theorem 9.5.6 and proposition 9.2.6, plus proposition 9.2.7, fully characterizes the least requirement for functional completeness in CCC's.

9.5.8 Corollary *Let $A = (X,\cdot)$ be a combinatory algebra. Then $\boldsymbol{PER}_A$ is a CCC, where $T < X$, $X\times X < X$ and, for $u(x,y) = x\cdot y$, $\Lambda(u)\in\boldsymbol{PER}[X,X^X]$ is principal. Moreover $\underline{A}(u) = A$.*
Proof. In view of theorem 9.5.6 and proposition 9.5.5, we only need to define the object S^R in $\mathbf{PER}_A$, which represents **PER**[R,S]. Set then

$$aS^Rb \quad \text{iff} \quad \forall x,y\in X\ (xRy \Rightarrow (ax)S(by)).$$

Recall now that, by assumption, each function in **P**[X,X] is representable. The rest of the proof that $\mathbf{PER}_A$ is a CCC is an obvious generalization of example 3.4.1. ♦

9.5.9 Remark While completing the proof that $\mathbf{PER}_A$ is a CCC, in corollary 9.5.8, one may notice that it is only required, for all R, S and all $f\in$**PER**[R,S], that f has a representative in X. This is the point which allows the generalization to the partial case (see section 9.6, besides $\mathbf{PER}_\omega$ in example 3.4.1).

The next result proves the converse of corollary 9.1.12 and completes our categorical understanding of λ-models.

9.5.10 Theorem *Let $A = (X, \cdot, \varepsilon)$ be a λ-model. Then, in the CCC $\boldsymbol{PER}_A$, there exist $\psi\in\boldsymbol{PER}[X^X,X]$ and $\phi\in\boldsymbol{PER}[X, X^X]$ such that $X^X < X$ via (ψ, ϕ). Moreover, $(X,\cdot) \cong \underline{A}(\Lambda^{-1}(\phi))$, and $\varepsilon = \psi(\psi\circ\phi)$. If A is extensional, then $X^X \cong X$.*
Proof Let $f\in$**PER**[X,X] and $a\in X$ be a representative for f. Define then $\psi(f) = \varepsilon a$. By (ε_2), ψ is well defined. Recall also that X^X is the exponent representing **PER**[X,X] in $\mathbf{PER}_A$, hence $\psi\in$**PER**$[X^X,X]$.

As for φ, define $\phi(a) = \underline{\lambda}x.ax$ for any $a\in X$. Clearly $\phi\in$**PER**$[X,X^X]$. Compute then

$$\begin{aligned}\phi(\psi(f)) &= \underline{\lambda}x.\varepsilon ax \qquad \text{if a represents f}\\ &= f \qquad\qquad \text{by } (\varepsilon_1).\end{aligned}$$

Thus $X^X < X$, via (ψ, ϕ).

Finally, $(X,\cdot) \cong \underline{A}(\Lambda^{-1}(\phi))$, since $\phi(a)(b) = ab$. Moreover $\psi(\phi(a)) = \varepsilon a$ and, hence, ε represents $\psi\circ\phi$. Thus

$$\begin{aligned} \psi(\psi\circ\phi) &= \varepsilon\varepsilon && \text{by definition of } \psi \\ &= \varepsilon && \text{by } (\varepsilon 3). \end{aligned}$$

Finally, if $a = \varepsilon a = \psi(\phi(a))$ for all a, then $\psi\circ\phi = id$ and, hence, $X^X \cong X$. ♦

We conclude this part by summarizing the connections between type-free λ-calculus and categories with enough points obtained so far. This provides a unified framework for the topic.

9.5.11 Theorem *Let* ***C*** *be a CCC and* A *an object of* ***C****. Then*

1. $A^A \cong A \Rightarrow$ A *is an extensional λ-model;*

2. $A^A < A \Rightarrow$ A *is a λ-model;*

3. $\exists p \in \mathbf{C}[A, A^A]$ *principal,* $T < A$ *and* $A\times A < A \Rightarrow$ A *is a combinatory algebra.*

Conversely,

1. A *is an extensional λ-model* $\Rightarrow$ $A^A \cong A$ *in* $\boldsymbol{PER}_A$*;*

2. A *is a λ-model* $\Rightarrow$ $A^A < A$ *in* $\boldsymbol{PER}_A$*;*

3. A *is a combinatory algebra* $\Rightarrow$ $\exists p \in \boldsymbol{PER}_A[A, A^A]$ *principal,* $T < A$ *and* $A\times A < A$ *in* $\boldsymbol{PER}_A$.

9.5.12 Remark In the categorical semantics of lambda calculus, we have to deal with Cartesian (closed) categories, and thus with products and projections. Without much increasing the complexity of the semantics, it is thus possible to consider also a type-free λ-calculus with explicit pairing, λβηπ, in analogy to the typed case, see section 8.2. We leave to the reader the task of defining, as an exercise, this calculus and giving its semantics on a reflexive object U in a CCC **C**. As a matter of fact, one only needs to add $A\times A \cong A$ to (1) and (2) in theorem 9.5.11 in order to obtain characterizations of the models of $\lambda\beta(\eta)\pi$.

In conclusion, the diligent reader will notice that the models of λβηπ are exactly the CCC with e.p. and with a unique object nonisomorphic to the terminal one. It may be fair, then, to call λβηπ the "unityped" λ-calculus.

9.6 Typed and Applicative Structures: Applications and Examples

In the first part of this section, we sketch a recent application of the typed and type-free λ–calculus to category theory. In a sense, this application goes in the other direction with respect to our prevailing perspective, as, so far, we mostly applied categorical tools to the understanding of deductive systems and their calculus of proofs (λ–calculus).

The question we answer here may be simply stated, in categorical terms:

(1) which isomorphisms hold in all CCC's ?

The motivation is clear, as a simple and decidable equational theory of types will allow us to detect provably isomorphic types or, equivalently, valid isomorphisms in all models. In functional programming, for example, when retrieving programs from a library where they are collected according to their types, the search should be done "up to provable isomorphisms", as the same program may have been coded under isomorphic types; for example, a search program for lists of a given length, may be typed by $\mathrm{INT}\times\mathrm{LISTS} \to \mathrm{LISTS}$ or, equivalently, by $\mathrm{LISTS} \to (\mathrm{INT} \to \mathrm{LISTS})$ (see references). The result turns out to be an application of λ–calculus to categories, as we look at the problem from a proof-theoretic view point and, by the work done in chapter 8, we actually answer to the following equivalent question.

Consider the intuitionistic calculus of sequents, in section 8.3, and suppose that proofs of $A \vdash B$ and of $A \vdash B$ are given. Then one may ask

(2) in which cases the composition of $A \vdash B$ and $B \vdash A$ (and of $B \vdash A$ and $A \vdash B$) reduce, by cut-elimination, to the axiom $A \vdash A$ (and $B \vdash B$, respectively) ?

Clearly, (2) corresponds to (1) when types are understood as objects. The point is that the deductions in (2) are coded by λ-terms, M and N, say, as described in section 8.3. By this, and by a lot of (hinted) hacking on λ-terms, we will characterize valid isomorphisms by looking at the structure of M and N such that $M\circ N = I_A$ and $N\circ M = I_B$, by β–conversion.

The second part develops little general Category Theory since it essentially gives more examples of CCC's and of λ–models. The structures presented will be combinatory algebras and models of type-free λβ, since models of λβη may be derived from the general results in chapter 10. In particular, we introduce a few relevant categories of complete partial orders as well as their "effective" versions and hint how they relate to the various categories of quotients or **PER**'s that we largely used in the previous sections.

Part 1: Provable isomorphisms of types

Consider the following equational theory of types. It is given by axiom schemata plus the obvious inference rules that turn "=" into a congruence relation.

9.6.1 Definition Th *is axiomatized as follows, where* T *is a constant symbol:*

1. $A \times T = A$

2. $A \times B = B \times A$

3. $A \times (B \times C) = (A \times B) \times C$

4. $(A \times B) \to C = A \to (B \to C)$

5. $A \to (B \times C) = (A \to B) \times (A \to C)$
6. $A \to T = T$
7. $T \to A = A$.

In remark 3.3.3, we already asked the reader to prove that, when $\to$ and $\times$ are interpreted as cartesian product and exponent, the provable equations of Th hold as isomorphisms in any CCC. Note, though, that there are categorical models which realize Th, but are not CCC's. Take, say, a cartesian category and a bifunctor "$\to$" that is constant in the second argument.

We next hint how to prove the non trivial fact that Th characterizes exactly the valid isomorphisms in all CCC's, by using λ-calculus.

As pointed out in section 8, the typed λ-calculus is, at the same time:
a - the "theory" of CCC's;
b - the calculus of proofs of the intuitionistic calculus of sequents.

Thus the theorem is shown by observing that the isomorphic types in the (closed) term model of typed λ-calculus are provably equal in Th. This answers to (2) in the introduction above and, thus, to (1).

A term model is *closed*, when terms in it contain no free variables.

9.6.2 Definition *Given (an extension of) the typed λ-calculus, $\boldsymbol{\lambda}$, the (closed) term model is the type structure*

$|\boldsymbol{\lambda}| = \{|M: A| \ / \ M$ *is a (closed) term of type* $A\}$
where $|M: A| = \{ N \ / \ \boldsymbol{\lambda} \vdash M = N \}$.

Clearly, the type structure is non trivial, if a collection of ground or atomic types is given. The (pure) λ-calculus may be extended by adding fresh types and constants as well as consistent sets of equations. Consider now the extension of $\lambda\beta\eta\pi^t$ in section 8.3 by adding:
1 - a special atomic type T (the terminal object);
2 - an axiom schema

$$*A: A \to T$$

which gives a constant of that type;
3 - a rule

$$\frac{M: A \to T}{M = *A}$$

that gives the unicity of *A.

Call $\lambda\beta\eta\pi^{*t}$ this extended calculus and Tp^* its collection of types. The point is that the closed term model of $\lambda\beta\eta\pi^{*t}$ (and its extensions) forms a CCC, as the reader may check as an exercise. Then the provable equations of Th are realized in $|\lambda\beta\eta\pi^{*t}|$, as isomorphisms. We give an explicit name to these isomorphisms, as λ-terms provide the basic working tools.

9.6.3 Definition *Let $A,B\in Tp^*$. Then A and B are **provably isomorphic** ($A \cong_p B$) iff there exist closed λ-terms $M : A \to B$ and $N : B \to A$ such that $\lambda\beta\eta\pi^{*t} \vdash M \circ N = I_B$ and $\lambda\beta\eta\pi^{*t} \vdash N \circ M = I_A$, where I_A and I_B are the identities of type A and B. We then say that M and N are **invertible** terms in $\lambda\beta\eta\pi^{*t}$.*

9.6.4 Remark By general categorical facts, we then have the easy implication of the equivalence we want to show; namely, $Th \vdash A = B \Rightarrow A \cong_p B$. It may be worth for the reader to work out the details and construct the λ-terms which actually give the isomorphisms. Indeed, they include the "abstract" verification of cartesian closure; for example, "currying" is realized by $\lambda z.\lambda x.\lambda y.z\langle x,y\rangle$ with inverse $\lambda z.\lambda x.z(p1\ x)(p2\ x)$, that prove $(A \times B) \to C \cong_p A \to (B \to C)$; the term $\lambda z.\langle\lambda x.(p1\ (zx)),\lambda x.(p2\ (zx))\rangle$ with inverse $\lambda z.\lambda x.\langle(p1\ z)x,(p2\ z)\ x\rangle$ prove $A \to (B \times C) \cong_p (A \to B) \times (A \to C)$. The others are easily derived.

The proof of the other implication, i.e., $A \cong_p B \Rightarrow Th \vdash A = B$, roughly goes as follows. As a first step, types are reduced to "type normal forms", in a "type rewrite" system. This will eliminate terminal types and bring products at the outermost level. Then one needs to show that isomorphisms between type normal forms yield componentwise isomorphisms. This takes us to the pure typed λ-calculus (i.e., no products nor T's). Then a characterization of the invertible terms of the pure type-free calculus is easily applied in the typed case, as the invertible type-free terms happen to be typable. The syntactique structure of the invertible terms gives the result.

The axioms of Th suggest the following rewrite system R for types (essentially Th "from left to right ", with no commutativity):

9.6.5 Definition (Type rewriting R) *Let ">" be the transitive and substitutive **type-reduction** relation given by:*

1. $A \times T > A$
1'. $T \times A > A$
3. $A \times (B \times C) > (A \times B) \times C$
4. $(A \times B) \to C > A \to (B \to C)$
5. $A \to (B \times C) > (A \to B) \times (A \to C)$
6. $A \to T > T$
7. $T \to A > A$.

The system R yields an obvious notion of **normal form for types** (type normal form), i.e., when no type reduction can be applied. Note that 1, 1', 6 and 7 "eliminate the T's", while 4 and 5 "bring outside $\times$". It is then easy to observe that each type normal form is identical to T or has the structure $S_1\times ... \times S_n$ where each S_i does not contain T nor "$\times$". We write **nf(S)** for the normal form of S (there is exactly one, see 1.6) and say that a normal form is non trivial if it is not T.

9.6.6 Proposition R *is Church-Rosser and each type has a unique type normal form in* R.
Proof Easy exercise. ♦

By the implication discussed in remark 9.6.4, since $\mathsf{R} \vdash S > R$ implies $\mathsf{Th} \vdash S = R$, it is clear that any reduction $\mathsf{R} \vdash S > R$ is witnessed by an invertible term of type $S\to R$.

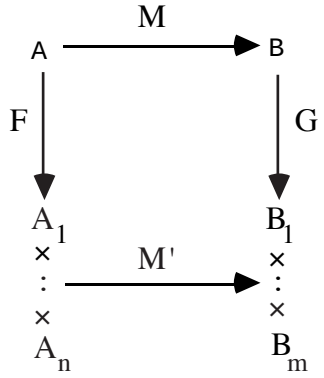
9.6.7 Corollary *Given types S and R, one has:*
1 - $\mathsf{Th} \vdash S = nf(S)$ *and, thus,*
2 - $\mathsf{Th} \vdash S = R \Leftrightarrow \mathsf{Th} \vdash nf(S) = nf(R)$.

In conclusion, when $\mathsf{Th} \vdash S = R$, either we have $nf(S) \equiv T \equiv nf(R)$, or $\mathsf{Th} \vdash nf(S) \equiv (S_1\times ... \times S_n) = (R_1\times ... \times R_m) \equiv nf(R)$. A crucial lemma below shows that, in this case, one also has $n = m$.

The assertion in the corollary can be reformulated for invertible terms in a very convenient way:

9.6.8 Proposition (commuting diagram) *Given types A and B, assume that F: A→nf(A) and G: B→nf(B) prove the reductions to type n.f.. Then a term M: A→B is invertible iff there exist an invertible term M': nf(A)→nf(B), such that* $M = G^{-1} \circ M' \circ F$.
Proof. ⇐) Set $M^{-1} \equiv (G^{-1}\circ M'\circ F)^{-1} \equiv F^{-1}\circ M'^{-1}\circ G$, then M is invertible.
⇒) Just set $M' = G\circ M\circ F^{-1}$. Then $M'^{-1} \equiv F\circ M^{-1}\circ G^{-1}$ and M' is invertible. ♦

A diagram easily represents the situation in the proposition:



We now state a few lemmas that should guide the reader through the basic ideas of this application of λ-calculus to category theory. Most technical proofs, indeed λ-calculus proofs, are omitted (and the reader should consult the references).

Recall first that, when $\mathsf{Th} \vdash S = R$, one has

$$\mathrm{nf}(S) \equiv T \equiv \mathrm{nf}(R), \text{ or } \mathsf{Th} \vdash \mathrm{nf}(S) \equiv (S_1 \times \ldots \times S_n) = (R_1 \times \ldots \times R_m) \equiv \mathrm{nf}(R).$$

Notice that, in the latter case, there cannot be any occurrence of T in either type. Indeed, a non trivial type normal form cannot be provably equated to T, as it can be easily pointed out by taking a non trivial model. Thus it may suffice to look at equations such as $(S_1 \times \ldots \times S_n) = (R_1 \times \ldots \times R_m)$ with no occurrences of T and, hence, to invertible terms with no occurrences of the type constant T in their types. We can show that these terms do not contain any occurrence of $*A$ either, for no type A, via the following lemma.

9.6.9 Lemma *Let M be a term of $\lambda\beta\eta\pi^{*t}$ in n.f..*

1 - (Terms of a product type) *If $M: A \times B$, then either $M \equiv \langle M1, M2 \rangle$, or there is $x:C$ such that $x \in FV(M)$ and $A \times B$ is a type subexpression of C.*

2 - (Every term, whose type contains no T, has no occurrence of $*A$ constants) *Assume that in M there is an occurrence of $*A$, for some type A. Then there is some occurrence of the type constant T in the type of M or in the type of some free variable of M.*

Proof. By induction on the structure of M. ♦

Note now that (the equational theory of) $\lambda\beta\eta\pi^{*t}$ is a conservative extension of (the equational theory of) $\lambda\beta\eta\pi^{t}$. Similarly for $\lambda\beta\eta\pi^{t}$ w.r.t. $\lambda\beta\eta^{t}$. Thus, invertibility in the extended theory, given by terms of a purer one, holds in the latter.

9.6.10 Proposition (Isomorphisms between type normal forms are given by terms in $\lambda\beta\eta\pi^{t}$) *Assume that S and R are non trivial type normal forms. If the closed terms M and N prove $S \cong_p R$ in $\lambda\beta\eta\pi^{*t}$, then their normal forms contain no occurrences of the constants $*A$. (Thus, M and N are actually in $\lambda\beta\eta\pi^{t}$).*

Proof By the previous lemma, as the terms are closed and no T occurs in their type. ♦

So we have factored out the class of constants *A, and we restricted the attention to $\lambda\beta\eta\pi^t$. By the next step, we reduce the problem to the pure calculus, i.e., we eliminate pairing as well, in a sense.

9.6.11 Proposition (Isomorphic type normal forms have equal lenght) *Let* $S \equiv S_1 \times ... \times S_m$ *and* $R \equiv R_1 \times ... \times R_n$ *be type normal forms. Then* $S \cong_p R$ *iff*
$n = m$ *and there exist* $M_1,...,M_n ; N_1,...,N_n$ *such that*

$x_1: S_1, ..x_n: S_n \vdash \langle M_1,...,M_n\rangle: (R_1 \times ... \times R_n)$

$y_1: R_1, ..y_n : R_n \vdash \langle N_1,...,N_n\rangle: (S_1 \times ... \times S_n)$

with $M_i[\underline{x} := \underline{N}] =_{\beta\eta} y_i$, *for* $1 \le i \le n$

$N_j[\underline{y} := \underline{M}] =_{\beta\eta} x_j$, *for* $1 \le j \le n$

and there exist permutations σ, π *over* n *such that*

$M_i = \lambda \underline{u}_i.x_{\sigma i}\, \underline{P}_i$ *and* $N_j = \lambda \underline{v}_j.y_{\pi j}\, \underline{Q}_j$

($\underline{M}$ *is a vector of terms; substitution of vectors of equal lenght is meant componentwise).*

Proof (Not obvious, see references). ♦

By induction, one may easily observe that terms of $\lambda\beta\eta\pi^t$ whose type is arrow-only belong to $\lambda\beta\eta^t$. Thus, one may look componentwise at terms that prove an isomorphism. The next point is to show that each component, indeed a term of $\lambda\beta\eta^t$, yields an isomorphism. This will be done by using a characterization of invertible terms in the pure calculus. The same result will be applied once more in order to obtain the result we aim at.

The characterization below has been given in the type-free calculus, as an answer to an old question of Church on the group of type-free terms. We follow the type-free notation, also for notational convenience.

9.6.12 Definition *Let* M *be a type-free term. Then* M *is a* ***finite hereditary permutation*** *(f.h.p.) iff either*

(i) $\lambda\beta\eta \vdash_u M = \lambda x.x$, *or*

(ii) $\lambda\beta\eta \vdash_u M = \lambda z.\, \lambda \underline{x}.z\underline{N}_\sigma$, *where if* $|\underline{x}| = n$ *then* σ *is a permutation over* n *and* $z\underline{N}_\sigma = zN_{\sigma 1}N_{\sigma 2}\ldots N_{\sigma n}$, *such that, for* $1 \le i \le n$, $\lambda x_i.N_i$ *is a finite hereditary permutation.*

For example, $\lambda z.\lambda x_1.\lambda x_2.zx_2x_1$ and $\lambda z.\lambda x_1.\lambda x_2.zx_2(\lambda x_3.\lambda x_4.x_1x_4x_3)$ are f.h.p.'s. The structure of f.h.p.'s is tidily desplayed by Böhm-trees. The **Böhm-tree** of a term M is (informally) given by:

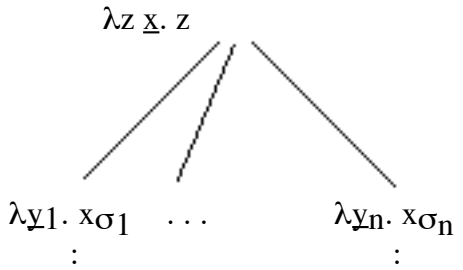
$BT(M) = \Omega$ if M has no head normal form

$BT(M) = \lambda x_1 \ldots x_n.\, y$ if $M =_\beta \lambda x_1 \ldots x_n.\, y\, M_1 \ldots M_p$

$$\begin{array}{c} / \ \ldots \ \backslash \\ BT(M_1) \quad BT(M_p) \end{array}$$

(see references).

It is easy to observe that a $BT(M)$ is finite and Ω-free iff M has a normal form. Then one may look at f.h.p.'s as Böhm-trees, as follows:



and so on, up to a finite depth (note that $\underline{y}_i$ may be an empty string of variables). Clearly, the f.h.p.'s are closed terms and possess a normal form. In particular, exactly the abstracted variables at level $n+1$ appear at level $n+2$, modulo some permutation of the order (note the special case of z at level 0). The importance of f.h.p.'s arises from the following classic theorem of λ-calculus. (Clearly, the notion of invertible term given in 9.6.3 easily translates to type-free λ-calculi).

9.6.13 Theorem *Let M be an untyped term possessing normal form. Then M is $\lambda\beta\eta$-invertible iff M is a f.h.p..*

Recall now that all typed terms possess a (unique) normal form (see references). Let then M be a typed λ-term and write $e(M)$ for the **erasure** of M, i.e. for M with all type labels erased.

Remark Observe that the erasures of all axioms and rules of the typed lambda calculus are themselves axioms and rules of the type-free lambda calculus. Then, if M and N are terms of $\lambda\beta\eta^t$ and $\lambda\beta\eta^t \vdash M = N$, one has $\lambda\beta\eta \vdash e(M) = e(N)$. Thus, in particular, if $M : \sigma \to \tau$ and $N : \tau \to \sigma$ are invertible terms in $\lambda\beta\eta^t$, $e(M)$ and $e(N)$ are f.h.p.'s.

Exercise Show that the f.h.p.'s are typable terms (*Hint*: Just follow the inductive definition and give z, for instance, type $A1 \to (A2..... \to B)$, where the Ai's are the types of the $N_{\sigma i}$.) Then, by the a small abuse of language, we may talk also of typed f.h.p.'s. Observe that these are exactly the typed invertible terms in definition 9.6.3.

The first application of 9.6.13 we need is the following.

9.6.14 Proposition *Let* $M_1, \ldots M_n, N_1, \ldots, N_n$ *and permutation* σ *be as in lemma 9.6.11. Then, for all* i, $\lambda x_{\sigma i}.M_i : S_{\sigma i} \to R_i$ *and* $\lambda y_i.N_{\sigma i} : R_i \to S_{\sigma i}$ *are invertible terms.*

Proof. For a suitable typing of the variables it is possible to build the following terms of $\lambda\beta\eta^t$ (we erase types for convenience):

$M = \lambda z x_1 \ldots x_n . z M_1 \ldots M_n$

$N = \lambda z y_1 \ldots y_n . z\ N_1 \ldots N_n$.

It is an easy computation to check, by the definition of the M_i's and of the N_i's, that M and N are invertible. Moreover, they are (by construction) in normal form, thus, by theorem 9.6.13, M and N are f.h.p.'s. This is enough to show that every M_i has only one occurrence of the x_i's (namely $x_{\sigma i}$); similarly for the N_i's.

Thus we obtain

$$M_i[\underline{x} := \underline{N}] \equiv M_i[\, x_{\sigma(i)} := N_{\sigma(i)}] =_{\beta\eta} y_i, \text{ for } 1 \le i \le n$$

$$N_i[\underline{y} := \underline{M}] \equiv N_i[\, y_{\pi(i)} := M_{\pi(i)}] =_{\beta\eta} x_i, \text{ for } 1 \le i \le n$$

and, hence, for each i, $\lambda x_{\sigma(i)}.M_i : S_{\sigma(i)} \to R_i$ and $\lambda y_i.N_{\sigma(i)} : R_i \to S_{\sigma(i)}$ are invertible. ♦

As a result of the work done so far, we can then focus on invertible terms whose types contain only "$\to$" i.e., investigate componentwise the isomorphisms of type normal forms. Of course, these isomorphisms will be given just by a fragment of theory Th.

Call S the subtheory of Th given by just one proper axiom (plus the usual axioms and rules for "="), namely

$$\text{(swap)} \qquad A \to (B \to C) = (B \to (A \to C)) .$$

S is a subtheory of Th by axioms 2 and 4 of Th.

9.6.15 Proposition *Let* A, B *be type expressions with no occurences of* T *nor* $\times$. *Then*

$$A \cong_p B \ \Rightarrow\ \mathsf{S} \vdash A = B.$$

Proof Suppose $A \cong_p B$ via M and N. As usual, we may assume without loss of generality that M and N are in normal form. By lemma 9.6.9 and the remark after 9.6.11, M and N actually live in $\lambda\beta\eta^t$ and, by theorem 9.6.13, they are f.h.p.'s. We prove $\mathsf{S} \vdash A = B$ by induction on the depth of the Böhm-tree of M.

Depth 1: $M \equiv \lambda z : C.\ z$. Thus $M : C \to C$, and $\mathsf{S} \vdash C = C$ by reflexivity.

Depth n+1: $M \equiv \lambda z : E.\ \lambda \underline{x} : \underline{D}.\ z\underline{N_\sigma}$. Recall $z\underline{N_\sigma} = z N_{\sigma 1} \ldots N_{\sigma n}$ where if the ith abstraction in $\lambda \underline{x} : \underline{D}$ is $\lambda x_i : D_i$ then the erasure of $\lambda x_i : D_i.N_{\sigma_i}$ is a f.h.p.. Thus $\lambda x_i : D_i.N_{\sigma_i}$ gives (half of) a provable isomorphism from D_i to some F_i. Hence the type of N_{σ_i} is F_i. In order to type check, we must have $E = (F_{\sigma 1} \to \ldots \to F_{\sigma n} \to B)$ for some B. Thus the type of M is $(F_{\sigma 1} \to \ldots \to F_{\sigma n} \to B) \to (D_1 \to \ldots \to D_n \to B)$. By induction, since the height of the Böhm tree of (the erasure of) each $\lambda x_i : D_i.\ N_{\sigma_i}$ is less than the height of the Böhm tree of M, ona has $\mathsf{S} \vdash D_i = F_i$ for $1 \le i \le n$. By a repeated use of the rules for "=", we get

$S \vdash (F_{\sigma 1} \to ... \to F_{\sigma n} \to B) = (D_{\sigma 1} \to ... \to D_{\sigma n} \to B)$.

Hence it suffices to show

$S \vdash (D_{\sigma 1} \to ... \to D_{\sigma n} \to B) = (D_1 \to ... \to D_n \to B)$.

This is quite simple to show by a repeated use of axiom (swap) above in conjunction with the rules. ♦

Clearly, also the converse of proposition 9.6.15 holds, since the "⇐" part in 9.6.15 is provable by a fragment of the proof hinted in 9.6.4. Thus one has:

$$S \vdash A = B \iff A \cong_p B \text{ by terms in } \lambda\beta\eta^t.$$

The result we aim at, is just the extension of this fact to Th and $\lambda\beta\eta\pi*^t$.

9.6.16 Main Theorem *$S \cong_p R \iff Th \vdash S = R$*

Proof. In view of 9.6.4, we only need to prove $S \cong_p R \Rightarrow Th \vdash S = R$. As we know, this is equivalent to proving $nf(S) \cong nf(R) \Rightarrow Th \vdash nf(S) = nf(R)$.

Now, by proposition 9.6.11, for $nf(S) \equiv (S_1 \times ... \times S_n)$ and $(R_1 \times ... \times R_m) \equiv nf(R)$, we have

$nf(S) \cong nf(R) \Rightarrow$ $n = m$ and there exist $M_1, ..., M_n, N_1, ..., N_n$

and a permutation σ such that $\lambda x_{\sigma i}.M_i : S_{\sigma i} \to R_i$ and $\lambda y_i.N_{\sigma i} : R_i \to S_{\sigma i}$.

By 9.6.14, these terms are invertible, for each i. Thus, by 9.6.15, $S \vdash R_i = S_{\sigma i}$ and, hence, by the rules, $Th \vdash S = R$. ♦

This concludes the proof of the main theorem of this part.

9.6.17 Corollary *Given types A and B, it is decidable whether they are (their interpretation yields) isomorphic (objects) in all CCC's.*

Proof *(Hint)* Reduce A and B to type normal form. Check that these have an equal number of factors. If so, observe that theory S does not change the lenght of types and perfom the required swaps to check the equality, in that theory, of each component. ♦

Exercise Check the complexity of the theory of provable isomorphisms.

Part 2: Higher type objects as models of the type-free λ-calculus

We give here some examples of categories and objects with the properties mentioned in the previous sections and discuss connections to Higher Type Recursion Theory, a highly developed topic to which denotational semantics of programming languages is greatly indebted. This theory suggested the early structures for a "generalized theory of computation," stressed the role of CCC's and, jointly

with category theory, set the basis for the construction of the early models of (type-free) λ-calculus and, thus, of functional programming languages.

We first mention a simple way to obtain lots of type-free models in CCC's with reflexive objects. Then we apply this construction to the main type structures for higher type recursion: the partial continuous and computable functionals in all finite higher types. Well-established results allow to recover from those structures the various hierarchies of total functionals, which actually started the topic (see references).

In section 2.4 we already gave two examples of reflexive objects in different CCC and, thus, of λ-models. When presenting the first, Pω, in 2.4.1, we promised to show the reflexivity of another very familiar Scott domain: the collection $P(R)$ of the partial (recursive) functions from ω to ω. Its reflexivity, see theorem 9.5.2, will be a consequence of a stronger property, with respect to a suitable category.

Exercise Let **C** be a CCC and T be a terminal object. Then $\forall X,Y \in Ob_{\mathbf{C}}$, if $T < Y$, one has $X < X^Y$.
(*Solution*: The retraction (i,j) is given by $i = \Lambda(pr_1) \in \mathbf{C}[X,X^Y]$ and $j = eval\circ(id\times t) \in \mathbf{C}[X^Y,X]$ for some fixed $t \in \mathbf{C}[T, X]$. Indeed, $j\circ i = j\circ(i\times d_T) = eval\circ(id\times t)\circ(i\times d_T) = eval\circ(i\times id)\circ(id\times t) = eval\circ(\Lambda(pr_1)\times id)\circ(id\times t) = pr_1\circ id\times t = id$.)

Let the set of type symbols, Tp, contain at least the atomic type 1. Then, for X in a CCC **C**, set

$$X^1 = X, \text{ and, for } A = X^\sigma \text{ and } B = X^\tau, \text{ set } X^{\sigma\to\tau} = B^A \text{ and } X^{\sigma\times\tau} = A\times B.$$

9.6.18 Lemma *Let U be a reflexive object in a CCC **C**. Then, for $\{U^\sigma\}_{\sigma\in Tp}$ as above*

$$\forall \sigma, \tau \in Tp \quad U^\sigma < U^\tau \text{ in } \mathbf{C}$$

In particular, then, $\forall \sigma \in Tp \; U^{\sigma\to\sigma} < U^\sigma$.

Proof Assume, by induction on the syntactic structure of types, that $U < U^\sigma$ and $U < U^\tau$. Clearly, $U\times U < U^{\sigma\times\tau}$. It is also easy to check that $U^U < U^{\sigma\to\tau}$. Similarly, from the inductive assumptions $U^\sigma < U$ and $U^\tau < U$, one has $U^{\sigma\to\tau} < U^U < U$, as U is a reflexive object, and $U^{\sigma\times\tau} < U\times U < U$, by proposition 2.3.6. Finally, $U < U\times U$ and $U < U^U$, by $T < U$ and the exercise. Therefore, $\forall \sigma,\tau \in Tp \; U^\sigma < U < U^\tau$ in **C**. ♦

As already shown, Scott domains, coherent domains and other categories of continuous functions have nontrivial reflexive objects. By the lemma, in these categories there are lots of λ–models, one in each higher type σ, over the reflexive object. We consider here yet another simple category with a reflexive object, namely the category **pcD** of ω-algebraic pair-consistent c.p.o.'s, and the object P of partial functions from integer to integer.

Call first a subset D of a p.o.set $(X,\leq)$ **pairwise consistent** if any pair of elements in D has an upper bound in X (and write $x \Uparrow y$ for $\exists z \in X\ x,y \leq z$). $(X,\leq)$ is **pair-consistent** if any pairwise consistent subset has a l.u.b. Call **pcD** the category of ω-algebraic pair-consistent c.p.o.'s (cf. examples in 2.4.1), with continuous maps as morphisms.

Exercise Prove that **pcD** is a full subCCC of the category **D** of Scott domains in 2.4.1.

9.6.19 Theorem *Let P be the set of the partial number-theoretic functions. Then for any object $\underline{X}$ in **pcD**, $\underline{X} < P$. In particular, P is reflexive.*

Proof. Let $\underline{X} = (X, X_0, \leq)$ be in **pcD**, and let $e: \omega \to X_0$ be an enumeration of the compact elements of $\underline{X}$. Define $\varphi: X \to P$ by setting, for all $n \in \omega$,

$$\varphi(x)(n) = \text{ if } e(n) \leq x \text{ then } 0 \text{ else if } \sim(x \Uparrow e(n)) \text{ then } 1 \text{ else } \perp .$$

Equivalently, $\varphi(x) \in P$ is uniquely determined by the ordered pair in which its domain splits

$$\langle \{ n \in \omega \mid \varphi(x)(n) = 0\}, \{n \in \omega \mid \varphi(x)(n) = 1\} \rangle = \langle \{n \in \omega \mid e(n) \leq x\}, \{n \in \omega \mid \sim(x \Uparrow e(n))\ \} \rangle .$$

It is easy to prove that φ is continuous.

In order to define $\psi: P \to X$, for any $f \in P$, set

$$X_f = \{e(i) \mid f(i) = 0 \ \underline{\text{and}}\ \forall j \leq i, \sim(e(i) \Uparrow e(j)) \to f(j) \neq 0\}.$$

First, X_f is pairwise consistent. Let i, j be such that $e(i), e(j) \in X_f$. Then $f(i) = f(j) = 0$. Suppose that $\sim(e(i) \Uparrow e(j))$. Then $i < j \Rightarrow f(i) \neq 0$ and $j \leq i \Rightarrow f(j) \neq 0$, which is impossible. Thus $e(i) \Uparrow e(j)$, so X_f is pairwise-consistent. By the consistency property of $\underline{X}$, we can define $\psi: P \to X$ by $\psi(f) = \sup_{\underline{X}}(X_f)$. It is a simple exercise to prove that ψ is continuous and that $\psi \circ \varphi(x) = x$ for all $x \in X$. Therefore $\underline{X}$ is a retract of P. ♦

It is clear that computability is at hand. As a matter of fact, the categories and results described so far can be "effectivized," in analogy to the examples in 2.4.1, e.g., the category **ED**. Denote by X_o the collection of the compact elements of $(X,\leq)$ in **pcD**.

9.6.20 Definition *Let $\underline{X} = (X, X_O, e_O, \leq)$ be in **pcD** and $e_O: \omega \to X_O$ (bijective). Then $\underline{X}$ is **effectively given** if*

1. *$e_O(n) \Uparrow e_O(m)$ is a decidable predicate*
2. *$\exists g \in R\ \forall n,m\ (\ e_O(n) \Uparrow e_O(m) \Rightarrow e_O(g(n,m)) = \sup\{e_O(n), e_O(m)\}\ )$.*

It is easy to show that if $\underline{X}$ and $\underline{Y}$ are effectively given, then also the space of continuous functions from $\underline{X}$ to $\underline{Y}$ is effectively given. Indeed, the category of effectively given ω-algebraic pair-consistent c.p.o.'s and continuous functions is cartesian closed (similarly to **ED**).

Recall that ideals are downward closed directed subsets of a poset $(X,\leq)$. As in the definition of the category **CD** of constructive domains in 2.4.1, the idea now is to take, within an effectively given $(X,X_o,e_o,\leq)$, only the l.u.b of those ideals of X_o, which are indexed over a recursively enumerable set. Call **computable** elements the l.u.b of the r.e. indexed ideals. Clearly, the computable elements of P are exactly the partial recursive functions, PR.

9.6.21 Definition *A sub-p.o.set X_c of an effectively given $\underline{X} = (X,X_o,e_o,\leq)$ is a* ***constructive and pair-consistent domain (ccd)*** *if for any ideal $D \subseteq X_o$ one has:*

D is principal in X_c iff $e_o^{-1}(D)$ is a recursively enumerable set.

Thus X_c contains exactly the computable elements of $\underline{X}$, e.g., $P_c = PR$. By the following exercise, this gives yet another interesting CCC.

Exercises

i. Let **CCD** be the category whose objects are ccd's and whose morphisms are the continuous and computable functions (computable as elements of the function spaces). Prove that **CCD** is cartesian closed.

ii. Prove 9.6.19 above for PR instead of P, i.e., prove that also PR is reflexive in **CCD**.

Consider now the type structure $\{PR^\sigma\}_{\sigma\in Tp}$ constructed over PR in **CCD**. These are known as the (higher type) partial recursive functionals. By the exercise and lemma 9.6.18, they yield a (type-free) λ-model in any finite type, as $PR^{\sigma\to\sigma} \lhd PR^\sigma$.

CCD tidily relates to categories defined in the previous section, provided that a minor generalization is made. So far, we have only been dealing with **total** applicative structures, i.e., where "$\cdot$" is everywhere defined, as combinatory algebras are total structures. There exist, though, interesting **partial** applicative structures: for example, Kleene's $K = (\omega,\cdot)$, where $n\cdot m = \phi_n(m)$ for some acceptable Gödel numbering $\phi: \omega\to PR$ of the partial recursive functions.

In general, given a *partial* applicative structure $B = (X,\cdot)$, i.e., "$\cdot$" may be a partial binary operation, one can define the categories $\mathbf{P}_B$, $\mathbf{ER}_B$ and $\mathbf{PER}_B$ as in definitions 9.5.2 and 9.5.4, with a minor caution. Since we deal here with categories of total morphisms, we consider only total polynomials in each $\mathbf{P}[X^n,X^m]$; in particular in $\mathbf{P}[X,X]$, when defining $\mathbf{ER}[R,S]$ in definition 9.5.4. As for $\mathbf{PER}[R,S]$, each $f\in\mathbf{PER}[R,S]$ is "computed," in the sense of 9.5.4, by a (possibly partial) $g\in\mathbf{P}[X,X]$ which must be total on domR, though. In conclusion, by the exercise before 9.5.3 and remark 9.5.9, if $X\times X \lhd X$ in $\mathbf{P}_B$, then proposition 9.5.5 applies similarly and $\mathbf{P}_B$ and $\mathbf{ER}_B$ are full sub-CC of the CC $\mathbf{PER}_B$. Moreover, if B is a partial combinatory algebra, then $\mathbf{PER}_B$ is a CCC by corollary 9.5.8 and what follows. The remaining results carry on similarly.

This remark has already been (implicitly) applied when defining **PER** over ω. As a matter of fact, Kleene's $K = (\omega,\cdot)$ is a partial combinatory algebra, as it contains (indices for) partial k and s. Thus, the properties of **PER** (= $\mathbf{PER}_K$ and **ER** = $\mathbf{ER}_K$) could be derived also by the work in this chapter.

Exercise The reader has already checked that the category **EN** of numbered sets (see the examples in section 2.2) is equivalent to $\mathbf{ER}_K$. He or she may try to give now an extension of **EN** which is cartesian closed and equivalent to $\mathbf{PER}_K$.

We already mentioned, in example 2.4.1, an important theorem, the Generalized Myhill-Shepherdson Theorem, which related constructive domains and **EN**. In this frame, it may be restated as "**CCD** is equivalent, as a category, to a full subCCC of $\mathbf{ER}_K$." Jointly to the exercise, this provides interesting embeddings, up to equivalence, of **CCD** into $\mathbf{ER}_K$ into $\mathbf{PER}_K$.

It should be clear why Kleene's K is only a partial combinatory algebra and not a total one. If ω^ω represents $\mathbf{PER}[\omega,\omega] = \mathbf{P}[\omega,\omega]$, then there is no principal $p \in \mathbf{PER}[\omega,\omega^\omega]$, as there is no Gödel-numbering or effective enumeration of $\mathbf{P}[\omega,\omega] = R$, the recursive functions.

One may try another way to turn K into a combinatory algebra. Consider first the category $\mathbf{CCD}_p$ of ccd's and partial morphisms, i.e., partial continuous maps with open domain. Let $_^\perp = _^\circ$ be the lifting functor defined in example 5.2.4. $\mathbf{CCD}[\omega^\perp,\omega^\perp]$, then, coincides with PR plus the everywhere constant function on $\omega^\perp$. From any acceptable Gödel-numbering of PR it is easy to construct a principal $p' \in \mathbf{CCD}[\omega^\perp,PR]$; however, $\omega^\perp \times \omega^\perp < \omega^\perp$ in **CCD** fails, by a simple continuity argument. Thus also $(\omega^\perp,\cdot)$ does not yield a combinatory algebra. However, by $PR^{\sigma\to\sigma} < PR^\sigma$ in **CCD** and 9.5.10, λ-models may be found at any finite higher type.

Note that p' above or the Gödel-numberings are principal morphisms which cannot be turned into retractions, by the latter observation or because, given a partial recursive function, there is no uniform effective choice of one of its indices, by the Rice theorem. More examples could be given by taking combinatory algebras which cannot be turned into λ-models. Indeed, a simple example is given by the term model of Combinatory Logic, i.e., by a "model" constructed by purely syntactic tools.

Remark *(Partial vs. total maps)* The reader may have noticed that there are various notions of partiality mentioned in these sections. As for the last, it poses no problem: a partial applicative structure has a (possibly) partial application. One may construct over it, though, categories of total maps, such as **EN**, $\mathbf{ER}_K$ (cf. the definition of morphisms in these categories).

Note now that we called $\{PR^\sigma\}_{\sigma\in Tp}$ the *partial* continuous and computable functionals, even though, at any type higher then 1, these are *total*, i.e., always defined, continuous (and computable) maps. Why are they called partial, in the literature? The intuition should be clear, but the categorical

notion of complete object (see definition 2.5.6) may provide a better or more rigorous understanding. As for the intuition, PR contains partial maps, in the ordinary sense, and each of the higher types contains a least element: the empty set in PR, the constantly empty map in PR^{PR} and so on. Intuitively, these least elements give the "undefined value" in the intended type. Categorically, this is understood by observing that, by this, each higher type, except for type 0 (i.e., ω) is a complete object in the intended category of partial maps, in the sense of section 2.5. The point is that these objects, by theorem 2.5.9, are exactly those such that the partial morphisms may be extended, and indeed viewed, as total ones.

Remark *(On total functionals)* Consider now ω and the total maps from ω to ω. We hint now at how to move to higher types and preserve totality of morphisms, in a categorical environment where it is possible to give a good notion of computability.

In sections 3.4 and 5.3 we presented the CCC (**sep-**)**FIL** of (separable) filter spaces and discussed some of its categorical properties. We also mentioned that this category is essentially nontopological, as some relevant types are not in the range of the embedding functor H: **Top**$\rightarrow$**FIL**, defined in example 5.3.7. In particular, **FIL**-ω, the sub-CCC of **sep**-**FIL** generated by ω, endowed with (the convergence induced by) the discrete topology, contains non topological types. The objects in **FIL**-ω are the sets of total continuous functionals.

On separable filter spaces, with an enumeration $\{U_i\}_{i\in\omega}$ of the base, one may consider the effectively given objects by generalizing the technique for domains in example 2.4.1 and definition 9.6.3. In short, one has to require that the base is decidable in the sense that

$$\{\, i \mid U_i = \varnothing \,\} \text{ and } \{\, (i_1,\ldots, i_m,k_1,\ldots,k_n) \mid U_{i1}\cap\ldots\cap U_{im} \subseteq U_{k1}\cap\ldots\cap U_{kn} \,\}$$

are recursive (uniformely in n and m). Then the computable elements are defined as limits of r.e. indexed filters. More precisely, let $\downarrow^s$ be the notion of convergence over filter spaces given in (s-conv.) in 5.3.7. Then set, for $(X,F)\in$**sep-FIL** with a decidable base,

$$x\in X \text{ is } \textbf{computable} \text{ iff } \exists\Phi\downarrow^s x \;\; \{\, i \mid U_i\in\Phi \,\} \text{ is r.e..}$$

In higher types, this defines total computable functionals.

Exercise Embedd the category **ED**, as topogical spaces, in 2.4.1 into **sep**-**FIL** by the functor H : **Top**$\rightarrow$**FIL** in 5.3.7 and describe how H and its left adjoint behave on computable elements. The computable elements of **FIL**-ω are the (Kleene-Kreisel) total continuous and computable functionals (see references).

As a final connection to the other categories of effective maps we used here, we just mention that a complex quotienting technique, which hereditarily defines total elements as the functions that take total elements to total elements, allows us to establish adjoint equivalences between **ED-**$\boldsymbol{\omega}^{\perp}$, the sub-CCC generated by $\boldsymbol{\omega}^{\perp}$ in **ED**, and $\mathbf{PER}_K$-ω, the sub-CCC generated by (ω,id) in $\mathbf{PER}_K$.

Similarly, the subCCC $\mathbf{PER}_{P\omega}$-ω in $\mathbf{PER}_{P\omega}$, the category of p.e.r.'s over the reflexive object $P\omega$, corresponds to **FIL-**ω. (See references).

References The original construction in Scott (1972) gives a non trivial isomorphism $A^A \cong A$ in a CCC of lattices. The general notion of categorical model of the type-free calculus $\lambda\beta\eta$ and $\lambda\beta$ are investigated in Berry (1979), Obtulowicz and Wiweger (1982), Koymans (1982), by using, though, the category of retractions, which does not need to have e.p., instead of **PER** (see Barendregt (1984) for a survey). By this one may deal with a weaker notion of model, the λ-algebras. λ-algebras, originally called pseudo-λ-models in Hindley and Longo (1980), are exactly the models of Combinatory Logic with β-equality (see also Barendregt (1984) or Hindley and Seldin (1986)). They are characterized as in theorem 9.5.11(2), by dropping the assumption that **C** has enough points and they are in between combinatory algebras and λ-models (in Barendregt and Koymans (1980) it is shown that the term model of CL cannot be turned into a λ-algebra.) However, retractions do not form a CCC on combinatory algebras and do not help in the categorical understanding of CL Thus we used **PER**, as in Longo and Moggi (1990), which we largely followed and where the models of CL were first characterized. The use of a choice operator ε in order to extend combinatory algebras to λ-models, which we followed here, is formalized in Meyer (1982).

Cousineau and al. (1985) and Curien (1986) introduce the categorical abstract machine.

The valid isomorphisms in Cartesian Closed categories are characterized in Bruce and DiCosmo and Longo (1990) (by a different proof, they where also given in Soloviev (1983)). The invertibility theorem 9.6.13, for the type-free λ-calculus, is due to Dezani (1976). Rittri (1989) applies isomorphisms in all CCC's to retrieval methods in libraries of programs.

Finally, pairwise consistent domains and the properties of $T\omega$ are investigated in Plotkin (1978). Filter spaces and related categories are used, for higher type recursion theory, in Hyland (1979). Ershov (1973-76) and Longo and Moggi (1984-84a) establish the relation between various classes of total and partial functionals.