Chapter 8

FORMULAE, TYPES, AND OBJECTS

During the last two decades, computer science has turned the Proof Theory of mathematical logic from a philosophical investigation of the foundations of human reasoning into an applied sector of mathematics. Many important logical systems of the beginning of the 1970s, which “fifteen years later” proved of great relevance for computer science, were invented by logicians such as Girard, Martin-Löf and Troelstra, who were mostly interested in constructive approaches to mathematics and foundational problems. Still in 1975, in the first pages of his book on Proof Theory, Takeuti introduced “the formalization of the proofs of mathematics, and the investigation of the structure of these proofs” as a mere “fruitful method in investigating mathematics.” Moreover, he himself remarked that “while set theory has already contributed essentially to the development of modern mathematics, it remains to be seen what influence proof theory will have on mathematics”.

If not on mathematics, Proof Theory has surely proved since that time his influence on theoretical computer science. The modern theory of functional languages and lambda calculus owes much of his actual settlement to mathematical logic, and what until a few years ago was known as the “denotational semantics of programming languages” has grown under the direct influence of Proof Theory, and together with the understanding of the logical aspects of Category Theory. But far from being only a rich field for applications, computer science has also been for mathematical logic an inexhaustible source of mutual enrichment. The leading theme of this stimulating relation, as for the topic presented in this chapter, has been the so called Curry-Howard correspondence, which exploits the connections between Proof Theory and Type Theory. This correspondence, also known by the suggestive name of “formulae-as-types analogy”, has been a main methodological tool both for understanding the relations between intuitionistic logic and typed lambda calculus, and for the design of new functional languages with powerful type systems. The main idea of the Curry-Howard correspondence, is that logical formulae can be interpreted as types in a suitable type theory; a proof of a formula is then associated with a λ-term of the proper type, and the reduction of a proof by cut-elimination corresponds to the normalization of the associated λ-term. As a consequence, if a formula is derivable in a certain logical system, then the corresponding type is inhabited in the associated type theory. The Curry-Howard correspondence works only for logical systems of Intuitionistic Logic. This restriction should be clear, since the constructive, procedural interpretation of the notion of proof was the very basis of Brower's approach to mathematics, which inspired Heyting's formalization of intuitionistic logic. Moreover, although the formulae-as-types analogy can also be applied to logical systems based on axioms and inference rules, such as that of Hilbert, just switching from λ-terms to combinators in the associated type theory, it has a more elegant application to systems of Natural Deduction. Indeed the procedural understanding of a logical proof is more clear in a system like

Natural Deduction, where one proceeds by the method of drawing inferences from assumptions, than in Hilbert's system, where one draw inferences from axioms. Furthermore, especially in Gentzen's variant of Natural Deduction, the inference rules of the calculus are closely related to the intuitive "operational" interpretation of the logical signs, and this fact allows one to proceed in the construction of the proof in a certain direct fashion, affording an interesting normal form for deductions which has no clear counterpart in Hilbert's system.

8.1 λ-Notation

Consider the following mathematical definition of the function $f: \mathbf{N} \to \mathbf{N}$:

$$f(x) = 5*x + 3 .$$

Note that it is not "f" that has been actually defined, but "f(x)," that is the result of the application of f to a formal parameter x which is supposed to range over the integers.

f(x) is not a function: it is a polynomial, that is, an operational description of the behavior of f when applied to x. The mechanism that allows us to "abstract" from the formal parameter x, and thus to pass from the knowledge of "how the function works" to the knowledge of "what the function is," is called **lambda abstraction (λ-abstraction)**. The function f in the example above is defined by the lambda abstraction of $5x + 3$ with respect to x, and is denoted by $\lambda x.\ 5*x+3$.

The complementary action to λ-abstraction is called **application**. The application of two terms M and N is usually represented by their justaposition (MN). From the computational point of view, λ-abstraction and application are related by the following rule, known as β-reduction:

$$(\lambda x.M)N \to [N/x]M$$

where [N/x]M means the substitution of N instead of x in the term M.
For example,

$$(\lambda x.\ 5*x+3)4 \ \to\ 5*4+3 \ =\ 23$$

The lambda notation was explicitly introduced by the logician Alonzo Church in the **lambda calculus** (λ-calculus), a formalism invented to develop a general theory of computable functions, and to extend that theory with logical notions providing a foundation for (part of) mathematics.

In computer science, the lambda notation made its first appearance in the programming language LISP; this was also the first language to use procedures as objects of the language. Like the early lambda calculus, LISP is an untyped language; that is, programs and data are not distinguished, but they are all elements of a unique untyped universe: the universe of λ-terms for the λ-calculus, the universe of S-expressions for LISP (see next chapter). Anyway, as every computer scientist knows from his or her own programming practice, types arise naturally, even starting from untyped universes, when objects are categorized according to their usage and behavior. The ultimate question is whether it wouldn't be better to consider types as an *a priori* schema of human knowledge, instead of an attempt to organize subsets of objects with uniform properties out of an untyped universe. In computer science, the debate about typed and untyped languages is very lively today, since it reflects the unresolvable conflict between reliability and flexibility of the language. Nowadays, the practical success of a language is often due to the the more or less successful attempt to compromise security and freedom; in this respect, the language ML is probably one of the most interesting examples. Anyway, since the first appearance of types in Algol 60, when typing of variables was introduced to check at compile time the connections of instances of use with associated declarations, typing has been considered more and more an essential discipline that must not only help, but guide the programmer in the design of code.

8.2 The Typed λ-Calculus with Explicit Pairs ($\lambda\beta\eta\pi^t$)

The collection Tp of type labels, over a ground set At of atomic type symbols, is inductively defined by

i. $At \subseteq Tp$;
ii. if $A,B \in Tp$, then $A \to B \in Tp$;
iii. if $A,B \in Tp$, then $A \times B \in Tp$.

For every type A there exists a denumerable number of **variables**, ranged over by lower case letters near the end of the alphabet. We use upper case letters M, N, P, . . . , as metavariables for terms. The fact that a term M has type A will be denoted with the expression "M: A."

The **well typed (λ-)terms** (w.t.t.) and their associated types, are defined according to the following formation rules:

1. every variable $x: A$ is a w.t.t.;
2. if $x: A$ is a variable, and $M: B$ is a w.t.t, then $\lambda x{:}A.M: A \to B$ is a w.t.t.;
3. if $M: A \to B$ is a w.t.t and $N: A$ is a w.t.t, then $MN: B$ is a w.t.t.;
4. if $M: A$ is a w.t.t and $N: B$ is a w.t.t, then $\langle M,N\rangle: A \times B$ is a w.t.t.;
5. if $M: A \times B$ is a w.t.t, then $fst(M): A$ is a w.t.t.;
6. if $M: A \times B$ is a w.t.t, then $snd(M): B$ is a w.t.t.

Given a w.t.t M: B, the set **FV(M)** of the **free variables** of M, is defined as follows:

1. if $M \equiv x$, then $FV(M) = \{x\}$;
2. if $M \equiv \lambda x{:}A.N$, then $FV(M) = FV(N)-\{x\}$;
3. if $M \equiv NP$, then $FV(M) = FV(N) \cup FV(P)$;
4. if $M \equiv \langle N,P\rangle$, then $FV(M) = FV(N) \cup FV(P)$;
5. if $M \equiv fst(N)$, then $FV(M) = FV(N)$;
6. if $M \equiv snd(N)$, then $FV(M) = FV(N)$.

The **substitution [M/x]N: B** of a proof M: A for a generic x: A in a proof N: B is defined in the following way:

1. if $N: B \equiv x: A$ then $[M/x]N: B \equiv M: A$;
2. if $N: B \equiv y: A$, $x \neq y$, then $[M/x]N: B \equiv N: B$;
3. if $N: B \equiv \lambda x:C.P : B$, then $[M/x]N: B \equiv \lambda x:C.P : B$;
4. if $N: B \equiv \lambda y:C.P : B$, $x \neq y$, $y \notin FV(M)$, then $[M/x]N: B \equiv \lambda y:C.[M/x]P : B$;
5. if $N: B \equiv PQ : B$, then $[M/x]N: B \equiv [M/x]P[M/x]Q : B$;
6. if $N: B \equiv \langle P,Q \rangle : B$, then $[M/x]N: B \equiv \langle [M/x]P,[M/x]Q \rangle : B$;
7. if $N: B \equiv fst(P) : B$, then $[M/x]N: B \equiv fst([M/x]P) : B$;
8. if $N: B \equiv snd(P) : B$, then $[M/x]N: B \equiv snd([M/x]P) : B$.

As an easy consequence of the definition above, it follows that if $x \notin FV(N)$, then $[M/x]N: B \equiv N: B$. Note also that, if $x \notin FV(N)$, then $[P/x][M/y]N: B \equiv [([P/x]M)/y]N: B$.

Given a sequence $\mathbf{M} = M_1,..., M_n$ of terms, and sequence $\mathbf{x} = x_1, . . . , x_n$ of variables, $[\mathbf{M}/\mathbf{x}]N$ denotes the simultaneous substitution of every term M_i for the variable x_i in the term N. We also use the notation $[M/\mathbf{x}]N$ to express the simultaneous substitution of the term M for all the variables in $\mathbf{x}$.

We consider an equational theory of proofs, defined as the minimal congruence relation "=" which satisfies the following axiom schemas:

(α) $\lambda x:A.M = \lambda y:A.[y/x]M$
$(\rightarrow\beta)$ $(\lambda x:A.M)N = [N/x]M$
$(\rightarrow\eta)$ $\lambda x:A.(Mx) = M$, if $x \notin FV(M)$
$(\times \beta_1)$ $fst(\langle M,N \rangle) = M$
$(\times \beta_2)$ $snd(\langle M,N \rangle) = N$
$(\times \eta)$ $\langle fst(P),snd(P) \rangle = P$.

The previous equations may be read from left to right : then one obtains a rewriting system, which defines the operational semantics of lambda calculus as a programming language. Explicitly, we define a **reduction relation** $\Rightarrow$ as the minimal relation between with respect to terms such that

$(\rightarrow\beta)$ $(\lambda x:A.M)N \Rightarrow [N/x]M$
$(\times \beta_1)$ $fst(\langle M,N \rangle) \Rightarrow M$
$(\times \beta_2)$ $snd(\langle M,N \rangle) \Rightarrow N$
$M \Rightarrow M'$ implies $MN \Rightarrow M'N$
$M \Rightarrow M'$ implies $NM \Rightarrow NM'$
$M \Rightarrow M'$ implies $\lambda x:A.M \Rightarrow \lambda x:A.M'$
$M \Rightarrow M'$ implies $fst(M) \Rightarrow fst(M')$
$M \Rightarrow M'$ implies $snd(M) \Rightarrow snd(M')$
$(\rightarrow\eta)$ $\lambda x:A.(Mx) \Rightarrow M$, if $x \notin FV(M)$
$(\times \eta)$ $\langle fst(P),snd(P) \rangle \Rightarrow P$.

The rewriting rules associated with $(\to\beta)$, $(\times\,\beta_1)$ and $(\times\,\beta_2)$ are called β-reductions, and those associated with $(\to\eta)$ and $(\times\,\eta)$ are called η-reductions. We put the η-reductions aside, because of their lesser computational interest. In the following section we shall see that from the point of view of the Curry-Howard correspondence both β– and η-reductions have an interpretation in terms of proof normalization. Also in this context, however, the β-rules play a major role.

A term is in **normal form** (respectively β-normal form), if it is no more reducible (respectively β-reducible). The (β-)reduction relation is confluent and noetherian.

Let us now look at some simple examples of programs written in the typed lambda calculus. As usual, when considering the formal expressiveness of a calculus, we compare it with the problem of representing integers and arithmetic functions.

Given a type σ, we call $\mathbf{N}_\sigma = (\sigma\to\sigma)\to(\sigma\to\sigma)$ the type of **σ-iterators** (**σ-numerals**). The reason for the name is that in the type $(\sigma\to\sigma)\to(\sigma\to\sigma)$ we have all the terms with the following structure:

$$0_\sigma \equiv \lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.y$$
$$1_\sigma \equiv \lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.xy$$
$$\dots$$
$$n_\sigma \equiv \lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.x^n y,\text{ where }\; x^n y = x(...(xy))\;\text{ “n times”.}$$

The effect of the term n_σ is to take a function x of type $\sigma\to\sigma$, a term y of type σ, and iterate the application of x to y n times. It is possible to show that if σ is an atomic type, then the terms of the form n_σ, together with the identity $\lambda x^{\sigma\to\sigma}.x^{\sigma\to\sigma}$ are the only closed terms in normal form of type σ.

We can now define the function $succ_\sigma \equiv \lambda n{:}\,\mathbf{N}_\sigma\,\lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,x(nxy)$.

The term $succ_\sigma$ has type $\mathbf{N}_\sigma\to\mathbf{N}_\sigma$ and its effect is to increase the iteration of one unit. For example,

$$\begin{aligned} succ_\sigma\, 0_\sigma &= (\lambda n{:}\,\mathbf{N}_\sigma\,\lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,x(nxy))\,0_\sigma \\ &= \lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,x(\,0_\sigma xy) \\ &= \lambda x^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,x(y) \\ &= 1_\sigma \end{aligned}$$

Define now $add_\sigma{:}\,\mathbf{N}_\sigma\times\mathbf{N}_\sigma\to\mathbf{N}_\sigma$ by $add_\sigma \equiv \lambda z{:}\,\mathbf{N}_\sigma\times\mathbf{N}_\sigma\,\lambda f^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,\mathrm{fst}\,z\,f\,(\mathrm{snd}\,z\,f\,y\,)$

For example, we have:

$$\begin{aligned} add_\sigma <1_\sigma,1_\sigma> &= (\lambda z{:}\,\mathbf{N}_\sigma\times\mathbf{N}_\sigma\,\lambda f^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,\mathrm{fst}\,z\,f\,(\mathrm{snd}\,z\,f\,y\,)\,)<1_\sigma,1_\sigma> \\ &= \lambda f^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,\mathrm{fst}<1_\sigma,1_\sigma>f\,(\mathrm{snd}<1_\sigma,1_\sigma>f\,y\,) \\ &= \lambda f^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,1_\sigma\,f\,(1_\sigma\,f\,y\,) \\ &= \lambda f^{\sigma\to\sigma}\,\lambda y^{\sigma}.\,f\,(\,f\,y\,) \\ &= 2_\sigma. \end{aligned}$$

Analogously, $mult_\sigma{:}\,\mathbf{N}_\sigma\times\mathbf{N}_\sigma\to\mathbf{N}_\sigma$ is given by

$$mult_\sigma \equiv \lambda z{:}\,\mathbf{N}_\sigma\times\mathbf{N}_\sigma\,\lambda f^{\sigma\to\sigma}\lambda y^{\sigma}.\,\mathrm{fst}\,z\,(\mathrm{snd}\,z\,f\,)\,y.$$

As an example of computation, we have

$$
\begin{aligned}
\mathrm{mult}_\sigma \langle 2_\sigma, 2_\sigma\rangle &= (\lambda z{:}\, \mathbf{N}_\sigma \times \mathbf{N}_\sigma\ \lambda f^{\sigma\to\sigma}\ \lambda y^\sigma.\ \mathrm{fst}\ z\ (\mathrm{snd}\ z\ f)\ y)\ \langle 2_\sigma, 2_\sigma\rangle \\
&= \lambda f^{\sigma\to\sigma}\ \lambda y^\sigma.\ \mathrm{fst}\ \langle 2_\sigma, 2_\sigma\rangle\ (\mathrm{snd}\ \langle 2_\sigma, 2_\sigma\rangle\ f)\ y \\
&= \lambda f^{\sigma\to\sigma}\ \lambda y^\sigma.\ 2_\sigma\ (2_\sigma\ f)\ y \\
&= \lambda f^{\sigma\to\sigma}\ \lambda y^\sigma.\ (2_\sigma\ f)(2_\sigma\ f\ y) \\
&= \lambda f^{\sigma\to\sigma}\ \lambda y^\sigma.\ f\,(f(f\,(f\ y)))) \\
&= 4_\sigma.
\end{aligned}
$$

As a final example, we present a test to 0_σ. That is, we want to define a term test 0_σ that takes two terms M and N of type σ and a σ-numeral n; if $n = 0_\sigma$ then the function yields M as result; otherwise, it outputs N. Here it is: test $0_\sigma \equiv \lambda z{:}\ \sigma\times\sigma\ \lambda n{:}\ \mathbf{N}_\sigma.\ n(\ \lambda y^\sigma.(\mathrm{snd}\ z)\ )(\mathrm{fst}\ z)$.

It is possible to prove (see references) that these are essentially all the arithmetic functions we can compute by the (pure!) simply typed λ-calculus, that is, all the functions that are defined by composition of the previous ones. For example the predecessor function is not representable.

The simply typed λ-calculus cannot be considered as a real programming language, but only as a paradigm for strongly typed (functional) languages, which allow us to point out their main common properties. In the last section of this chapter we shall consider an extension of the language by fixpoint operators. The combined use of these operators with a reasonable set of arithmetic primitives gives to the language its full computational power.

8.3 The Intuitionistic Calculus of Sequents

Among the many variants of intuitionistic logic expressed in systems of Natural Deduction, Gentzen's calculi of cequents has recently gained much popularity among the computer science community. In the first place, one of the main aspects of the procedural interpretations of proofs is that of making explicit the assumptions on which a formula occurrence in a deduction depends, and this is just what the notion of sequent is meant for. Moreover, since the calculi of sequents can be understood as metacalculi for the deducibility relation in the corresponding system of natural deduction, it seems to be the best instrument for the investigation of the structure and the properties of proofs. Note, however, that even when handling sequents, the dependencies of a formula by its premises can be quite hard to understand, if one adopts some common choices on the structure of the sequent, such as avoiding repetitions of formulae in the antecedent, or fixing a certain alphabetic order for them. This is very clear if we consider a sequent as an effective process for transforming proofs of the premises into a proof of the consequence. It seems not only reasonable, but fair to guarantee the ability of handling two or more different proofs for a same formula. In particular, this means allowing repetitions for a formula in the antecedent of a sequent, each one with an associated hypothetical distinct proof. From the same point of view, even the idea of an ordering among the formulae does not make any sense; what one really needs, are rather some explicit rules in the calculus which allow us “to move formulae around” (exchange rules). The presence of repetitions of formulae requires a new rule too, stating that two proofs of a formula can actually be assumed to be the same, and thus affording to discharge one of them (contraction rule). These rules, together with the so-called weakening rule, that allows for adding new assumptions to a sequent, are usually referred to as "structural rules." The reason for this name should be clear, since these rules have nothing to do with logical signs and their meanings, but only with the structures of the derivations in the calculi. This property of the structural rules has been often misunderstood, and regarded as a sign of their lesser logical interest: Takeuti called them "weak inferences," with respect to the "strong inferences" of introduction of the logical connectives. The very recent revival of the structural rules, and the analysis of their impact on the logical systems is mostly due to Girard, and to the invention of linear logic (see chapter 4).

The calculi of sequents for classical and intuitionistic logic were introduced by Gentzen at the beginning of the 1930s, who called them L systems. In these systems the elementary statements are statements of deducibility. Each statement $\Gamma \vdash B$ (called a **sequent**) has a conclusion B and a collection Γ, possibly void, of premises, and it is interpreted as stating the existence of a proof leading to the stated conclusion and having no uncanceled premises other than some of those stated.

We consider only the subcalculus that deals with the two connectives of conjunction and implication, and that will be referred to as "positive calculus." The principal connective of the system is implication; the role of conjunction is rather minor, but its analysis will help in clarifying some issues we will be concerned with in the rest of this chapter.

The following presentation of the positive intuitionistic calculus of sequents is slightly different from the usual one. In particular, every formula A will be equipped with an associated proof, formally represented by a lambda term $M: A$. This association between proofs (λ-terms) and formulae is defined in the inference rules of the calculus. If there is a derivation of the sequnt $\Gamma \vdash M: B$, the proof M keeps a trace of the derivation tree. On the other hand, the so-modified inference rules can be understood as metarules for proof manipulation. This understanding of the calculus of sequents as a metacalculus for the deducibility relation is one of the most peculiar aspect of this formal system.

8.3.1 Logical Alphabet

1. atomic propositions, ranged over by $A, B, C, \ldots$;
2. logical symbols: $\times, \rightarrow$.

8.3.2 W.F.Formulae (Types)

1. every atomic proposition is a formula;
2. if A, B are formulae, then $A\times B$ is a formula;
3. if A, B are formulae, then $A\rightarrow B$ is a formula.

There is a bijective correspondence between formulae and types of the typed lambda calculus with explicit pairs. We can go further, and associate every formula B with a λ-term M of the respective type, which intuitively represents its proofs. In particular, if $x_1: A_1,..., x_n: A_n$ are the free variables in M, then M: B is a proof of M depending on the hypothesis $A_1,..., A_n$. The previous approach is sound with the intuitionistic interpretation of the notion of proof as an effective procedure that shows the validity of the conclusion B, as soon as one has a proof of the validity of the premises. Thus, if M is a proof of B, possibly depending on a generic proof x: A, one gets a proof of $A \to B$ by "abstraction on x: A," that is $\lambda x{:}A.M: A \to B$. Juxtaposition of proofs corresponds intuitively to their sequential application. In particular, if we have a proof $M: A \to B$, and we apply it to a proof N: A, then we obtain a proof MN of B. A proof of $A \times B$ is given by a pair of distinct proofs for A and B. Moreover, having a proof of $A \times B$, one can select the two proofs of A and B by means of the projections fst and snd .

The inference rules of the calculus exactly formalize this process of constructiing complex proofs by means of simpler ones.

A (well-typed) term M: A, will be called a **proof** when it is regarded from a logical viewpoint. A variable x: A, will be called a **generic proof** of A.

An intuitionistic **sequent** has the following syntactic structure:

$$x_1: A_1, \ldots, x_n: A_n \vdash M: B$$

where $x_1: A_1, \ldots, x_n: A_n$ is a finite (possibly empty) list of *distinct* generic proofs, and M: B is a proof of B whose free variables are among $x_1, \ldots, x_n$. Every formula in the left-hand-side (l.h.s.) has an associated distinct variable; thus no confusion can arise between formulae, even if they have the same name.

The intuitive interpretation of the sequent $x_1: A_1, \ldots, x_n: A_n \vdash M: B$ is that of a process which builds a proof M of B, as soon as it has proofs for $A_1, \ldots, A_n$, that is, a function f of type $A_1 \times \ldots \times A_n \to B$, or equivalently, $A_1 \to (\ldots (A_n \to B) \ldots)$, which can be obtained by functional completeness from the polynomial M.

Since the name of a generic proof is not relevant, we assume that sequents which differ only by a renaming of variables are syntactically identified. This is consistent with the α-conversion rule of proofs, in view of the intuitive interpretation of sequents sketched above.

We use Greek capital letters Γ, Δ, . . . to denote finite sequences of generic proofs in the left hand side of a sequent. A sequent thus has the form $\Gamma \vdash M: B$.

An **inference** is an expression

$$\frac{S_1}{S} \qquad\qquad \frac{S_1 \quad S_2}{S}$$

where S_1, S_2, and S are sequents. S_1 and S_2 are called the upper sequents and S is called the lower sequent of the inference. Intuitively, this means that when S_1 (S_1 and S_2) is (are) asserted, one can infer S from it (them).

We always suppose that the variables in the l.h.s. of the upper sequents are distinct. This is not a problem, since the variables of the upper sequents S_1 and S_2 that contribute to form the l.h.s. of the lower sequent S can be conveniently renamed.

The logical system restricts the legal inferences to those obtained from the following rules.

8.3.3 Axioms and Rules

(axiom) $\quad x: A \vdash x: A$

(exchange) $$\frac{\Gamma, x: A, y: B, \Gamma_1 \vdash M: C}{\Gamma, y: B, x: A, \Gamma_1 \vdash M: C}$$

(weakening) $$\frac{\Gamma \vdash M: B}{\Gamma, x: A \vdash M: B}$$

(contraction) $$\frac{\Gamma, x: A, y: A \vdash M: B}{\Gamma, z: A \vdash [z/x][z/y]M: B}$$

(cut) $$\frac{\Gamma \vdash M: A \qquad \Gamma_1, x: A, \Gamma_2 \vdash N: B}{\Gamma_1, \Gamma, \Gamma_2 \vdash [M/x]N: B}$$

($\rightarrow$, r) $$\frac{\Gamma, x: A \vdash M: B}{\Gamma \vdash \lambda x{:}A.M: A \rightarrow B}$$

($\rightarrow$, l) $$\frac{\Gamma_1 \vdash M: A \qquad \Gamma_2, x: B \vdash N: C}{\Gamma_1, \Gamma_2, y: A \rightarrow B \vdash [yM/x]N: C}$$

($\times$, r) $$\frac{\Gamma \vdash M: A \qquad \Gamma \vdash N: B}{\Gamma \vdash \langle M,N\rangle : A \times B}$$

($\times$, l, 1) $$\frac{\Gamma, x: A \vdash M: C}{\Gamma, z: A \times B \vdash [fst(z)/x]M: C}$$

($\times$, l, 2) $$\frac{\Gamma, y: B \vdash M: C}{\Gamma, z: A \times B \vdash [snd(z)/x]M: C}$$

Remark Note the identification of the premises of the upper sequents in the rule ($\times$, r).

The formula A in the weakening rule is called the **weakening formula** (of the inference). Similarly, we define the **cut formula**. In the case of the contraction rule we distinguish between the **contracting formulae** x: A, y: A and the **contracted formula** z: A. The formulae $A\to B$ and $A\times B$ in the logical rules of introduction of the respective connectives, are called the **principal formulae** of the inference; A and B are called the **auxiliary** formulae.

Unlike the usual approach, we intend the previous definition to refer only to the specific occurrences of the formulae pointed out in the inference rules. For instance, consider the following application of the rule $(\to, l)$:

$$(\to, l)\qquad \frac{\Gamma_1 \vdash M: A \qquad w: A\to B,\ x: B \vdash N: C}{\Gamma_1,\ w: A\to B,\ y: A\to B \vdash [yM/x]N: C}$$

The (occurrence of the) formula $A\to B$ associated with the proof y is a principal formula of this inference, but that associated with w is not.

It is easy to check that, for every rule of inference, the proof in the right-hand side of the lower sequents is well formed, if those of the upper sequents are as well. A **derivation** D is a tree of sequents satisfying the following conditions:

1. the topmost sequents of D are axioms;
2. every sequent in D except the lowest one (the root) is an upper sequent of an inference whose lower sequent is also in D.

The lower sequent in a derivation D is called the **end-sequent** of D. A path in a derivation D from a leaf to the root is called a **thread**. A derivation without the cut rule is called **cut-free**.

Examples The following is a derivation:

$$\frac{\dfrac{x: A \vdash x: A}{z: A\times B \vdash fst(z): A} \qquad \dfrac{y: B \vdash y: B}{z: A\times B \vdash snd(z): B}}{z: A\times B \vdash \langle fst(z),snd(z)\rangle: A\times B}$$

The sequence

$x: A \vdash x: A$

$z: A\times B \vdash fst(z): A$

$z: A\times B \vdash \langle fst(z),snd(z)\rangle: A\times B$

is a thread.

The tidy display of the introduction of new formulae during a derivation in Gentzen's calculus of sequent, allows us to follow the entire "life" of a formula, from the moment it is "created" (by an axiom, a weakening, or as a principal formula of a logical rule) to the moment it is "consumed" (by a cut, or as an auxiliary formula in a logical rule). Note in particular that a step of contraction must not be regarded as a consumption, but as an identification of a formula A in the lower sequent, with two formulae of the same name in the upper sequent.

Given a thread T and an occurrence of a formula A in the final sequent of T, we call **rank** of A in T the number of consecutive sequents in T, counting upward, that contain the same occurrence of the formula A (or a formula which has been contracted to it).

For instance, consider again the thread

$x: A \vdash x: A$

$z: A\times B \vdash fst(z): A$

$z: A\times B \vdash \langle fst(z),snd(z)\rangle: A\times B$

of the previous example. The rank of the formula A×B in the l.h.s. of the end sequent is 2, while the rank of the formula A×B in the r.h.s. is 1.

8.4 The Cut-Elimination Theorem

One of the more interesting properties of Gentzen's calculus of sequent is the cut-elimination theorem, also known as Gentzen's Hauptsatz. This result proves that every derivation in the calculus can be effectively transformed in a "natural normal form" which does not contain any application of the cut rule. Moreover we show that this process of reduction of the derivations toward a normal form is in parallel to the β-reduction of the associated proofs.

The identification of the derivations reducible to a same normal form is the base of the so-called "semantics of proofs," and, *a posteriori*, it provides a justification for the equational theory we have introduced over the language of proofs (at least for the β-conversions).

The following presentation of the cut-elimination theorem is meant to study the relations between a derivation ending in the sequent $\Gamma \vdash M: A$ and the proof $M: A$. Indeed, it should be clear that given a proof $M: A$, with $FV(M) \subseteq \Gamma$, it is possible to obtain a derivation of the sequent $\Gamma \vdash M: A$ by building a sort of "parse tree" for the term $M: A$. Anyway, this correspondence between derivations and proofs is not a bijection: the problems arise with the structural rules, since their application is not reflected in the terms. In particular, it is not possible to recover the order in which the structural rules have been applied during the derivation.

The exact formalization of the equivalence of derivations that differ only in "structural details" is not at all trivial, and it is the main goal of Girard's research of "a geometry of interaction". From this respect, the language of proofs is a very handy formalism that allows us to abstract from such structural details.

Before the cut-elimination theorem, we state a first, simple result that relates cut-free derivations and proofs in β-normal form.

8.4.1 Proposition *Let $\Gamma \vdash M: A$ be the end sequent of a cut-free derivation. Then the proof $M: A$ is in β-normal form.*

Proof Exercise. *Hint:* Show, by induction on the length of the derivation, that the proof M: A cannot contain any of the following subterms:

1. $(\lambda x{:}B.N)P: C$
2. $\mathrm{fst}(\langle N,P\rangle)$
3. $\mathrm{snd}(\langle N,P\rangle)$. ♦

One of the aims of this chapter is to put in evidence the logical aspects underlying functional type theories. For this reason we state and prove Gentzen's Hauptsatz and stress the equivalence of the normalization of a derivation D ending in a sequent $\Gamma \vdash M: A$ and the β-reduction of M.

8.4.2 Theorem (The Cut-Elimination Theorem) *Let D be a derivation of the sequent $\Gamma \vdash M: A$. Then there exists a cut-free derivation D' of the sequent $\Gamma \vdash N: A$, where N is the β-normal form of M.*

It is difficult to appreciate at first sight the cut-elimination theorem in its whole meaning. Indeed, it looks like a mere result about the redundancy of one of the inference rules in the calculus, and one would expect, as a next step, to drop the cut rule from the logical system. This is not at all the case. The fact is that the cut rule is a very relevant and deep attempt to express one of the crucial methodological aspects of reasoning, that is, the decomposition of a complex problem in simpler subproblems: in order to prove $\Gamma \vdash B$, we can prove two lemmas $\Gamma \vdash A$ and $A \vdash B$. This is very clear from the computer science point of view, where the emphasis is not on the results - that is, the terms in normal form - but on the program's synthesis, and its reduction (computation) when applied to data.

An important consequence of the cut-elimination theorem is the following. One can easily see that in any rule of inference except a cut, the lower sequent is no less complicated than the upper sequent(s). More precisely, every formula occurring in an upper sequent is a subformula of some formula occurring in the lower sequent. Hence a proof without a cut contains only subformulae of the formulae occurring in the end sequent (**subformula property**). From this observation, the consistency of the logical system immediately follows. Suppose indeed that the empty sequent $\vdash$ were provable. Then by the cut-elimination theorem, it would be provable without a cut; but this is impossible, by the subformula property of cut-free proofs.

In order to prove theorem 8.4.2, it is convenient to introduce a new rule of inference called a mix. The mix rule is logically equivalent to the cut rule, but it helps deal with the contraction rules, whose behavior during the cut elimination is rather annoying. We shall discuss again this problem at the end of the proof.

A **mix** is an inference of the following form:

$$\frac{\Gamma \vdash M: A \qquad \Gamma_1 \vdash N: B}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{x}]N: B}$$

where $\mathbf{x}$ is a vector of variables in Γ_1, $\Gamma_1{}^*$ is obtained from Γ_1 by erasing all the assumption x: A with $x \in \mathbf{x}$, and $[M/\mathbf{x}]N$ denotes the simultaneous substitution of M for all variables in $\mathbf{x}$, inside the term N.

The previous notion of mix rule is slightly different from the usual one. In particular the formula A can still appear in the lower sequent $\Gamma_1{}^*$.

An occurrence of A in the upper sequents of a mix rule is called a **mix formula** (of the inference) if and only if it satisfies one of the two following conditions:

1. it is in the r.h.s. of the left upper sequent;
2. it is in the l.h.s. of the right upper sequent, and it is associated with a proof $x{:}A$, with $x \in \mathbf{x}$.

Clearly a cut is a particular case of mix, where the vector $\mathbf{x}$ contains only one variable. Moreover, every mix can be simply obtained by a sequence of exchanges and contractions, a cut, and another series of exchanges.

Since a proof without a mix is also a proof without a cut, theorem 8.4.2 is proved if we prove the following:

8.4.3 Theorem (The Mix-Elimination Theorem) *Let D be a derivation of the sequent $\Gamma \vdash M: A$. Then there exists a mix-free derivation D' of the sequent $\Gamma \vdash N: A$, where N is the β-normal form of M.*

Theorem 8.4.3 is easily obtained from the following lemma, by induction on the number of the mix rules occurring in the derivation D.

8.4.4 Lemma *Let D be a derivation of the sequent $\Gamma \vdash M: A$ that contains only one cut rule occurring as the last inference. Then there exists a cut-free derivation D' of the sequent $\Gamma \vdash N: A$, where N is the β-normal form of M.*

The rest of this section is devoted to the proof of this lemma.

We define two scales for measuring the complexity of a proof. The **grade** of a formula A (denoted by g(A)) is the number of logical symbols contained in A. The grade of a mix is the grade of the mix formula. When a derivation D has only one cut as the last inference, we define the grade of D (denoted by g(D)) to be the grade of this mix.

Let D be a derivation which contains a mix

$$\frac{\Gamma \vdash M\colon A \qquad \Gamma_1 \vdash N\colon B}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{x}]N\colon B}$$

as the last inference. We call a thread in D a left (right) thread if it contains the left (right) upper sequent of the mix. The **rank** of a thread is the number of consecutive sequents in D (counting upward from the upper sequents of the cut rule) that contain the mix formula or a formula which has been contracted to it. The rank of every thread is at least 1. The left (right) rank of a derivation D is the maximum among the ranks of the left (right) threads in D. The rank of a derivation D is the sum of its left and right ranks. The rank of a derivation is at least 2.

We prove the lemma by double induction on the grade g and rank r of the derivation D. The proof is subdivided into two main cases, namely $r = 2$ and $r > 2$.

Case 1 : r = 2

We distinguish cases according to the forms of the proofs of the upper sequents of the cut rule.

1.1. The left upper sequent S_1 is an initial sequent. In this case D is of the form

$$\frac{y\colon A \vdash y\colon A \qquad \Gamma \vdash N\colon B}{\Gamma_1{}^* \vdash [y/\mathbf{x}]N\colon B}$$

And we obtain the same proof by a series of contractions starting from the sequent $\Gamma \vdash N\colon B$.

1.2. The right upper sequent S_2 is an initial sequent. Similarly.

1.3. Neither S_1 nor S_2 is an initial sequent, and S_1 is the lower sequent of a structural inference. It is easy to check that, in case $r = 2$, this is not possible.

1.4. Neither S_1 nor S_2 is an initial sequent, and S_2 is the lower sequent of a structural inference. The structural inference must be a weakening, whose weakening formula is A. The derivation D has the structure

$$\frac{\Gamma, x{:}\,A \vdash M{:}\,A \qquad \dfrac{\Gamma_1 \vdash N{:}\,B}{x{:}\,A, \Gamma_1 \vdash N{:}\,B}}{\Gamma, \Gamma_1 \vdash [M/x]N{:}\,B}$$

Since $x \notin FV(N)$, $[M/x]N \equiv N$, and we obtain the same proof $N{:}\,B$ as follows:

$$\frac{\dfrac{\Gamma_1 \vdash N{:}\,B}{\text{some weakenings}}}{\Gamma, \Gamma_1 \vdash N{:}\,B}$$

1.5. Both S_1 and S_2 are the lower sequents of logical inferences. Since the left and right ranks of D are 1, the cut formulae on each side are the principal formulae of the logical inferences, and the mix is actually a cut between these two formulae.

We use induction on the grade and distinguish two cases according to the outermost logical symbol of A.

($\to$) the derivation D has the structure

$$\frac{\dfrac{\Gamma, x{:}\,A \vdash M{:}\,B}{\Gamma \vdash \lambda x{:}A.M{:}\,A\to B} \qquad \dfrac{\Gamma_1 \vdash N{:}\,A \qquad \Gamma_2, y{:}\,B \vdash P{:}\,C}{\Gamma_1, \Gamma_2, z{:}\,A\to B \vdash [zN/y]P{:}\,C}}{\Gamma, \Gamma_1, \Gamma_2 \vdash [\lambda x{:}A.M/z][zN/y]P{:}\,C}$$

where, by assumption, the proofs ending with $\Gamma, x{:}\,A \vdash M{:}\,B$, $\Gamma_1 \vdash N{:}\,A$ or $\Gamma_2, y{:}\,B \vdash P{:}\,C$ do not contain any cut. Note now that

$$[\lambda x{:}A.M/z][zN/x]P{:}\,C \equiv [(\lambda x{:}A.M)N/y]P{:}\,C = [\,([N/x]M)\,/y]P{:}\,C,$$

which also suggests how to build the new derivation.

First consider the derivation

$$\frac{\Gamma_1 \vdash N{:}\,A \qquad \Gamma, x{:}\,A \vdash M{:}\,B}{\Gamma_1, \Gamma \vdash [N/x]M{:}\,B}$$

This proof contains only one mix as its last inference. Furthermore, the grade of the mix formula is less than $g(A\to B)$. By induction hypothesis, there exists a derivation D' of the sequent $\Gamma_1, \Gamma \vdash M'{:}\,B$, where M' is the β-normal form of $[N/x]M$. Then, with another mix we get

$$\frac{\Gamma_1, \Gamma \vdash M': B \qquad \Gamma_2, y: B \vdash P: C}{\Gamma, \Gamma_1, \Gamma_2 \vdash [M'/z]P: C}$$

Again we can apply the induction hypothesis, obtaining the requested derivation.

(×) we only consider the case in which the right upper sequent is obtained by a (×,r,1), the other case being completely analogous. The derivation has the structure

$$\frac{\dfrac{\Gamma \vdash M: A \quad \Gamma \vdash N: B}{\Gamma \vdash \langle M,N\rangle : A\times B} \qquad \dfrac{\Gamma_1, x: A \vdash P: C}{\Gamma_1, z: A\times B \vdash [fst(z)/x]P: C}}{\Gamma, \Gamma_1 \vdash [\langle M,N\rangle/z][fst(z)/x]P: C}$$

where by assumption the proofs ending with $\Gamma \vdash M: A$, $\Gamma \vdash N: B$ or $\Gamma_1, x: A \vdash P: C$ do not contain any mix. We have

$[\langle M,N\rangle/z][fst(z)/x]P: C \equiv [fst(\langle M,N\rangle)/x]P: C = [M/x]P: C.$

Consider the derivation:

$$\frac{\Gamma \vdash M: A \qquad \Gamma_1, x: A \vdash P: C}{\Gamma, \Gamma_1, z: A\times B \vdash [M/x]P: C}$$

This proof contains only one mix as its last inference. Furthermore the grade of the cut formula is less than $g(A\times B)$. By induction hypothesis, we have the requested cut-free derivation D'.

Case 2 : r > 2

The induction hypothesis is that we can eliminate the cut from every derivation D' that contains only one cut as the last inference, and that satisfies either $g(D') < g(D)$, or $g(D') = g(D)$ and $rank(D') < rank(D)$.

There are two main cases, namely, $rank_r(D) > 1$ and $rank_l(D) > 1$ (with $rank_r(D) = 1$).

2.1. $rank_r(D) > 1$: we distinguish several subcases according to the logical inference whose lower sequent is S_2.

2.1.1. The sequent S_2 is the lower sequent of a weakening rule, whose weakening formula is not the cut formula. The derivation D has the structure

$$\dfrac{\Gamma \vdash M{:}\ A \qquad \dfrac{\Gamma_1 \vdash N{:}\ B}{y{:}\ C, \Gamma_1 \vdash N{:}\ B}}{\Gamma, y{:}\ C, \Gamma_1{}^* \vdash [M/\mathbf{x}]N{:}\ B}$$

Consider the derivation D'

$$\dfrac{\Gamma \vdash M{:}\ A \qquad \Gamma_1 \vdash N{:}\ B}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{x}]N{:}\ B}$$

where the grade of D' is the same of D, namely g(A). Moreover, the two derivations have the same left rank, while $\mathrm{rank}_r(D') = \mathrm{rank}_r(D) - 1$; thus we can apply the induction hypothesis. With a weakening and some exchanges we obtain the requested mix-free derivation.

2.1.2. The sequent S_2 is the lower sequent of an exchange rule. Similarly.

2.1.3. The sequent S_2 is the lower sequent of a contraction rule. This is the main source of problems with the cut rule (see discussion at the end of the proof). With the mix rule, everything is instead very easy. For instance, let us consider the case when the contracted formula is a cut formula (the other case is even simpler). The derivation has the structure

$$\dfrac{\Gamma \vdash M{:}\ A \qquad \dfrac{\Gamma_1, x{:}\ A, y{:}\ A \vdash N{:}\ B}{\Gamma_1, z{:}\ A \vdash [z/x][z/y]N{:}\ B}}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{x}][z/x][z/y]N{:}\ B}$$

where $z \in \mathbf{x}$.

Consider the derivation D'

$$\dfrac{\Gamma \vdash M{:}\ A \qquad \Gamma_1, x{:}\ A, y{:}\ A \vdash N{:}\ B}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{x}']N{:}\ B}$$

where $\mathbf{x}'$ is obtained from $\mathbf{x}$ by erasing z and adding x, y.

The grade of D' is the same of D, namely g(A). Moreover, the two derivations have the same left rank, while $\mathrm{rank}_r(D') = \mathrm{rank}_r(D) - 1$; thus we can apply the induction hypothesis, and we obtain a cut-free derivation D" of $\Gamma, \Gamma_1{}^* \vdash N'{:}\ B$, where N' is the β-normal form of $[M/\mathbf{x}']N{:}\ B$. Since $[M/\mathbf{x}][z/x][z/y]N{:}\ B \equiv [M/\mathbf{x}']N{:}\ B$, the proof is completed.

2.1.4. The sequent S_2 is the lower sequent of a logical inference J, whose principal formula is not a cut formula.

The last part of the derivation D looks like

$$\dfrac{\Gamma \vdash M: A \qquad \dfrac{\Gamma_1 \vdash N: B}{\Gamma_2 \vdash P: C}\,J}{\Gamma, \Gamma_2{}^* \vdash [M/\mathbf{x}]P: C}$$

where the derivations of $\Gamma \vdash M: A$ and $\Gamma_1 \vdash N: B$ contains no mixes, and Γ_1 contains all the cut formulae of Γ_2. Consider the following derivation D':

$$\dfrac{\Gamma \vdash M: A \qquad \Gamma_1 \vdash N: B}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{x}]N: B}$$

The grade of D' is the same as that of D, namely g(A). Moreover, the two derivations have the same left rank, while $rank_r(D') = rank_r(D) - 1$; thus we can apply the induction hypothesis, and we obtain a cut-free derivation D" of $\Gamma, \Gamma_1{}^* \vdash N': B$, where N' is the β-normal form of $[M/\mathbf{x}]N: B$.

Now we can apply the J rule to get:

$$\dfrac{\Gamma, \Gamma_1{}^* \vdash N': B}{\Gamma, \Gamma_2{}^* \vdash P': C}\,J$$

and clearly P' is the β-normal form of $[M/\mathbf{x}]P: C$ (the inference rules are sound w.r.t. equality of proofs).

2.1.5. The sequent S_2 is the lower sequent of a logical inference J, whose principal formula is a cut formula. We treat only the case of the arrow.

(→) the derivation D has the structure

$$\dfrac{\Gamma \vdash M: A\to B \qquad \dfrac{\Gamma_1 \vdash N: A \qquad \Gamma_2, y: B \vdash P: C}{\Gamma_1, \Gamma_2, z: A\to B \vdash [zN/y]P: C}}{\Gamma, \Gamma_1{}^*, \Gamma_2{}^* \vdash [M/\mathbf{x}][zN/y]P: C}$$

where $z \in \mathbf{x}$.

Let $\mathbf{v1}, \mathbf{v2}$ be the variables of $\mathbf{x}$, which are respectively in Γ_1 and Γ_2. Note that $\mathbf{v1}$ and $\mathbf{v2}$ cannot be both empty by the hypothesis about rank.

If $\mathbf{v1}, \mathbf{v2}$ are both nonempty, consider the following derivations D_1 and D_2:

D_1:

$$\frac{\Gamma \vdash M: A \to B \qquad \Gamma_1 \vdash N: A}{\Gamma, \Gamma_1{}^* \vdash [M/\mathbf{v_1}]N: A}$$

D_2:

$$\frac{\Gamma \vdash M: A \to B \qquad \Gamma_2, y: B \vdash P: C}{\Gamma, \Gamma_2{}^*, y: B \vdash [M/\mathbf{v_2}]P: C}$$

If $\mathbf{v_1}$ ($\mathbf{v_2}$) is empty, and thus $\Gamma_1 = \Gamma_1{}^*$ ($\Gamma_2 = \Gamma_2{}^*$), let D_1 (D_2) be as follows:

D_1:

$$\cfrac{\cfrac{\Gamma_1 \vdash N: A}{\text{weakenings}}}{\Gamma, \Gamma_1{}^* \vdash N: A}$$

D_2:

$$\cfrac{\cfrac{\Gamma_2, y: B \vdash P: C}{\text{weakenings}}}{\Gamma, \Gamma_2{}^*, y: B \vdash P: C}$$

The grade of D_1 and D_2 is the same as that of D, namely $g(A \to B)$. Moreover, $\mathrm{rank}_l(D_1) = \mathrm{rank}_l(D_1) = \mathrm{rank}_l(D)$, and $\mathrm{rank}_r(D_1) = \mathrm{rank}_r(D_1) = \mathrm{rank}_r(D) - 1$. Hence, by induction hypothesis, there exist two derivations D_1' and D_2' respectively ending in the sequents

$\Gamma, \Gamma_1{}^* \vdash N_1: A$ with N_1 β-normal form of $[M/\mathbf{v_1}]N: A$

$\Gamma, \Gamma_2{}^*, y: B \vdash P_1: C$ with P_1 β-normal form of $[M/\mathbf{v_2}]P: C$.

Let $\mathbf{w}$ be the vector of variables in Γ, and let $\mathbf{w'}, \mathbf{w''}$ be two vectors of fresh variables of the same length. Let $\Gamma' = [\mathbf{w'}/\mathbf{w}]\Gamma$, $\Gamma'' = [\mathbf{w''}/\mathbf{w}]\Gamma$ be the sequences of assumptions obtained by renaming the variables in Γ. Let also $N_1' \equiv [\mathbf{w'}/\mathbf{w}]N_1$ and $P_1' \equiv [\mathbf{w'}/\mathbf{w}]P_1$ be the results of the same operation on the two terms N_1 and P_1.

Consider the derivation D'

$$\cfrac{\Gamma \vdash M: A \to B \qquad \cfrac{\Gamma', \Gamma_1{}^* \vdash N_1': A \qquad \Gamma'', \Gamma_2{}^*, y: B \vdash P_1': C}{\Gamma', \Gamma_1{}^*, \Gamma'', \Gamma_2{}^*, z: A \to B \vdash [z(N_1')/y](P_1'): C}}{\Gamma, \Gamma', \Gamma_1{}^*, \Gamma'', \Gamma_2{}^* \vdash [M/z][zN_1'/y]P_1': C.}$$

The grade of D' is the same of D, namely, $g(A \to B)$. Moreover, $\mathrm{rank}_l(D') = \mathrm{rank}_l(D)$, and $\mathrm{rank}_r(D') = 1$, since $z: A \to B$ is the only cut formula. We can apply again the induction hypothesis, getting a mix-free derivation of the final sequent

$\Gamma, \Gamma', \Gamma_1{}^*, \Gamma'', \Gamma_2{}^* \vdash P_2: C$ with P_2 β-normal form of $[M/z][zN_1'/y]P_1'$.

Identifying by means of contractions (and exchanges) the variables in $\Gamma, \Gamma', \Gamma''$, we get a mix-free derivation of $\Gamma, \Gamma_1{}^*, \Gamma_2{}^* \vdash P_3: C$, where P_3 is a β-normal form of $[\mathbf{w}/\mathbf{w'}][\mathbf{w}/\mathbf{w''}]P_2$.

Note now that:

$$\begin{aligned} [\mathbf{w}/\mathbf{w'}][\mathbf{w}/\mathbf{w''}]P_2 &= [\mathbf{w}/\mathbf{w'}][\mathbf{w}/\mathbf{w''}][M/z][zN_1'/y]P_1' \\ &\equiv [M/z][z([\mathbf{w}/\mathbf{w'}]N_1')/y]([\mathbf{w}/\mathbf{w''}]P_1') \end{aligned}$$

$$\begin{aligned}
&\equiv [M/z][zN_1/y]P_1 \\
&= [M/z][z([M/\mathbf{v_1}]N)/y]([M/\mathbf{v_2}]P) \\
&\equiv [M/z][M/\mathbf{v_1}][M/\mathbf{v_2}][zN/y]P \\
&\equiv [M/\mathbf{x}][zN/y]P: C.
\end{aligned}$$

2.2. $\text{rank}_l(D) > 1$ (and $\text{rank}_r(D) = 1$)
This case is proved in the same way as is 2.1 above.

This completes the proof of lemma 8.4.4 and, hence, of the cut-elimination theorem. ♦

8.5 Categorical Semantics of Derivations

In this section we will study the categorical semantics of the intuitionistic proof theory developed in the last section. The main idea is that a formula A is interpreted as an object $\mathsf{I}(A)$ in a category **C**. A *provable* sequent $B_1,..., B_n$ |- A is then associated with a morphism $f: \mathsf{I}(B_1)\otimes...\otimes\mathsf{I}(B_n)\to\mathsf{I}(A)$, where $\otimes$: $\mathbf{C}\times\mathbf{C}\to\mathbf{C}$ is a suitable bifunctor that gives meaning to the comma in the l.h.s. of the sequent. Since the comma is associative and commutative, $\otimes$ is a symmetric tensor product. Moreover, we need some sort of projections for the weakening rule, and a diagonal map in the contraction rule. Actually, all these morphisms, together with a few reasonable equations, turn $\otimes$ into a categorical product.

Since C |- A⇒B iff C, A |- B, it is natural to interpret the implication ⇒ as the right adjoint to the functor $\otimes$.

Also the binary connective × is bifunctor of the category. In particular, the fact that C |- A×B iff (C |- A and C |- B) suggests that × should be interpreted as a categorical product. As a consequence $\otimes$ and × coincide to within isomorphism, that is consistent with the well-known fact that the comma in the l.h.s. of a sequent has the same logical meaning of a conjunction. Anyway, since they are different concept, we prefer to maintain the distinction at the semantical level.

8.5.1 Definition *A categorical model of the positive intuitionistic calculus is a pair* $(\mathbf{C},\mathsf{I}_{At})$*, where* **C** *is a Cartesian closed category (possibly with two specified products* $\otimes$ *and* × *), and* I_{At} *is a map that takes every atomic formula to an object of* **C**.

Notation In the rest of this chapter, we shall use the same symbols ⇒ and × for indicating both the connectives of the calculus and the associated functors of the category: the intended meaning is clear from the context.

The analysis of the cut-elimination theorem will provide a more convincing justification for the previous notion of model. Now we concentrate on the interpretation.

I is extended to all the formulae in the following way:

$\mathsf{I}(A) = \mathsf{I}_{At}(A)$ if A is atomic

$\mathsf{I}(A\Rightarrow C) = \mathsf{I}(A)\Rightarrow\mathsf{I}(C)$

$\mathsf{I}(A\times C) = \mathsf{I}(A)\times\mathsf{I}(C)$

The categorical interpretation does not give meanings to sequents, but to derivations. Every derivation D whose final sequent is $B_1, \ldots, B_n \vdash A$ is interpreted by a morphism

$\mathsf{I}(D) = f\colon \mathsf{I}(B_1)\otimes \ldots \otimes\mathsf{I}(B_n) \to \mathsf{I}(A)$

Before giving the details of the interpretation, we recall a few results about Cartesian closed categories, and fix the notation.

We call $\text{weak-l}_{A,B}\colon A\otimes B\to B$, $\text{weak-r}_{A,B}\colon A\otimes B\to A$ the projections associated with the product $\otimes$, and $\text{fst}_{A,B}\colon A\times B\to B$, $\text{snd}_{A,B}\colon A\times B\to A$ the projections associated with the product $\times$.

We have the natural isomorphisms $\text{assoc}_{A,B,C}\colon (A\otimes B)\otimes C\to A\otimes(B\otimes C)$ and $\text{exch}_{A,B}\colon A\otimes B\to B\otimes A$, given by the symmetric monoidal structure associated with $\otimes$. Moreover, we have a diagonal map $\Delta_A = \langle \text{id}_A,\text{id}_A\rangle\colon A\to A\otimes A$ and an the evaluation map $\text{eval}_{A,C}\colon (A\Rightarrow C)\otimes A\to C$.

The interpretation is given by induction on the length of the derivation. We define it in a somewhat informal but suggestive (and concise) way, associating with every inference rule of the calculus in 8.3.4 a respective semantic rule. This will be enough as for the categorical meaning of the intuitionistic system in section 8.3.

8.5.2 Interpretation

(axiom) $\quad \text{id}_A : A \to A$

(associativity) $$\frac{f\colon A\otimes(B\otimes C) \to D}{f \circ \text{assoc}_{A,B,C}\colon (A\otimes B)\otimes C \to D}$$

(exchange) $$\frac{f\colon B\otimes A \to C}{f \circ \text{exch}_{A,B}\colon A\otimes B \to C}$$

(weakening) $$\frac{f\colon B \to C}{f \circ \text{weak-l}_{A,B}\colon A\otimes B \to C}$$

(contraction)
$$\frac{f:\ (A\otimes A)\otimes B \to C}{f \circ \Delta_A \otimes id_B:\ A\otimes B \to C}$$

(cut)
$$\frac{f:\ B \to A \qquad g:\ A\otimes D \to C}{g \circ f\otimes id_D:\ B\otimes D \to C}$$

$(\to, r)$
$$\frac{f:\ B\otimes A \to C}{\Lambda(f):\ B \to A\Rightarrow C}$$

$(\to, l)$
$$\frac{f:\ E \to A \qquad g:\ C \to D}{g \circ eval_{A,C} \circ id\otimes f:\ (A\Rightarrow C)\otimes E \to D}$$

$(\times, r)$
$$\frac{f:\ C \to A \qquad g:\ C \to B}{<f,g>:\ C \to A\times B}$$

$(\times, l, 1)$
$$\frac{f:\ A\otimes C \to D}{f \circ fst_{A,B}\otimes id_C:\ (A\times B)\otimes C \to D}$$

$(\times, l, 2)$
$$\frac{f:\ B\otimes C \to D}{f \circ snd_{A,B}\otimes id_C:\ (A\times B)\otimes C \to D.}$$

8.6 The Cut-Elimination Theorem Revisited

In this section, we look at the cut elimination theorem from the categorical point of view. Our goal is to provide a more convincing justification of the categorical notion of model, not a further proof of cut-elimination. Indeed the notion of CCC imposes some identifications between derivations that are not evident at the logical level of the inference rules. The fact is that we are not interested in giving semantics to the provability relation among formulae, but more generally to the whole proof system, with its associated normalization procedures. The point is that the equalities, which define CCC's, reflect the identity of derivations up to normalization. In particular, we have the following result.

8.6.1 Theorem (The Cut-Elimination Theorem) *Let D be a derivation of the sequent A |- B. Then there exists a cut-free derivation D' whose final sequent is A |- B, and such that in every model* $\mathbf{I}(D) = \mathbf{I}(D')$.

We do not prove the previous result, but instead analyze in detail some examples of derivations identified by the cut-elimination process. Our aim is to show that the Cartesian closed structure is imposed by this identification of proofs up to normalization. The examples are instrumental to this.

8.6.2 Example If in the following derivation the left upper sequent A |- A is an initial sequent, the cut is eliminated by taking the derivation of the right upper sequent

$$\frac{A \vdash A \qquad\qquad A \vdash B}{A \vdash B}$$

At the semantic level, we have the situation:

$$\frac{id_A: A \to A \qquad f: A \to B}{f \circ id_A: A \to B}$$

thus a model must satisfy the equation $f \circ id_A = f$. Analogously, when the right upper sequent is an axiom, we derive the equation $id_A \circ f = f$

8.6.3 Example Consider a derivation whose final part looks like

$$\frac{D \vdash A \qquad \dfrac{B \vdash C}{A, B \vdash C}}{D, B \vdash C}$$

the cut is eliminated in the following way:

$$\frac{B \vdash C}{D, B \vdash C}$$

At the semantic level we have the following situation:

$$\frac{g: D \to A \qquad \dfrac{f: B \to C}{f \circ \text{weak-l}_{A,B}: A \otimes B \to C}}{f \circ \text{weak-l}_{A,B} \circ g \otimes id_B: D \otimes B \to C}$$

and

$$\frac{f: B \to C}{f \circ \text{weak-l}_{D,B}: D \otimes B \to C}$$

thus, for every $g: D \to A$, we must have

$$\text{weak-l}_{D,B} = \text{weak-l}_{A,B} \circ g \otimes id_B : D \otimes B \to C \qquad (*)$$

The previous equation is clearly true if $\otimes$ is a categorical product, and weak-$l_{D,B}$ is the associated right projection.

8.6.4 Example Consider a derivation whose final part looks like

$$\dfrac{D \vdash A \qquad \dfrac{A, A \vdash B}{A \vdash B}}{D \vdash B}$$

This is the annoying case of the contraction. With the help of a mix rule, the derivation reduces to

$$\dfrac{D \vdash A \qquad A, A \vdash B}{D \vdash B}$$

Regarding the mix as a simple abbreviation of a series of cuts and structural inferences, the previous derivation is logically equivalent to

$$\dfrac{\dfrac{D \vdash A \qquad \dfrac{D \vdash A \qquad A, A \vdash B}{D, A \vdash B}}{D, D \vdash B}}{D \vdash B}$$

At the semantic level we have

$$\dfrac{g: D \to A \qquad \dfrac{f: A\otimes A \to B}{f \circ \Delta_A : A \to B}}{f \circ \Delta_A \circ g: D \to B}$$

which reduces to

$$\dfrac{\dfrac{g: D \to A \qquad \dfrac{g: D \to A \qquad f: A\otimes A \to B}{f \circ g\otimes id_A : D\otimes A \to B}}{f \circ g\otimes id_A \circ id_A\otimes g : D\otimes D \to B}}{f \circ g\otimes g \circ \Delta_D : D\otimes D \to B}$$

This implies that in a model we expect

$$\Delta_A \circ g = g \otimes g \circ \Delta_D \qquad (+)$$

that is the naturality of Δ .

8.6.5 Example Consider the derivation

$$\dfrac{\dfrac{B, A \vdash C}{B \vdash A \Rightarrow C} \qquad \dfrac{E \vdash A \qquad C \vdash D}{(A \Rightarrow C), E \vdash D}}{B, E \vdash D}$$

The last cut is eliminated by introducing two other cuts of lesser grade, namely

$$\dfrac{E \vdash A \qquad \dfrac{B, A \vdash C \qquad C \vdash D}{B, A \vdash D}}{B, E \vdash D}$$

At the semantic level this gives

$$\dfrac{\dfrac{h:\ B \otimes A \to C}{\Lambda(h):\ B \to A \Rightarrow C} \qquad \dfrac{f: E \to A \qquad g: C \to D}{g \circ \mathrm{eval}_{A,C} \circ \mathrm{id} \otimes f:\ (A \Rightarrow C) \otimes E \to D}}{g \circ \mathrm{eval}_{A,C} \circ \mathrm{id} \otimes f \circ \Lambda(h) \otimes \mathrm{id}:\ B \otimes E \to D}$$

that reduces to

$$\dfrac{f: E \to A \qquad \dfrac{h:\ B \otimes A \to C \qquad g: C \to D}{g \circ h:\ B \otimes A \to D}}{g \circ h \circ \mathrm{id} \otimes f : B, E \to D}$$

Note that in a Cartesian closed category, by the β-axiom of exponents, one has

$$g \circ \mathrm{eval}_{A,C} \circ \mathrm{id} \otimes f \circ \Lambda(h) \otimes \mathrm{id} = g \circ \mathrm{eval}_{A,C} \circ \Lambda(h) \otimes \mathrm{id} \circ \mathrm{id} \otimes f = g \circ h \circ \mathrm{id} \otimes f.$$

This completes the observation that the Cartesian closure may be described in terms of equivalence of proofs, up to normalization.

8.7 Categorical Semantics of the Simply Typed Lambda Calculus

In sections 8.3 and 8.4, we have investigated the relation between typed lambda calculus and intuitionistic logic, and in sections 8.5 and 8.6 we have established the connection between intuitionistic logic and Cartesian closed categories. In this section we want now to "fill the triangle" among typed lambda calculus, intuitionistic logic and Cartesian closed categories, studying the missing edge - namely the categorical semantics of the Typed Lambda Calculus, over CCC's.

The main idea of the categorical semantics of the typed lambda calculus is to interpret types as objects and terms as morphisms of a category **C**. In particular, the reader has probably noted some analogy between the axioms of $\lambda\beta\eta\pi^t$ and the axioms defining products and exponents in Cartesian closed categories. The relation is quite evident in the case of products: $fst^{\sigma\times\tau\to\sigma}$ and $snd^{\sigma\times\tau\to\tau}$ are easily understood as the projections associated with the categorical product of σ and τ. Apart from the problem of switching from "application" to "composition," the axioms of $\lambda\beta\eta\pi^t$ regarding the product are essentially identical to the axioms in definition 1.3.3 of categorical product. Less immediate is the connection between β (and η) and the rules defining the exponents of a CCC: the main problem is that the definition in the calculus is based on the process of substitution, for which we do not have any immediate equivalent in Category Theory. Let us look a little closer at our tentative interpretation: the intuitive idea is that a term M^τ with free variables among $x_1{}^{\sigma_1}, \dots, x_n{}^{\sigma_n}$ should be interpreted as a morphism M^τ from $\Delta = \sigma_1\times\dots\times\sigma_n$ to τ. Suppose now that we have two terms M^τ and N^γ, such that the free variables of $[N^\gamma/x^\gamma]M^\tau$ are in $\Delta = \sigma_1\times\dots\times\sigma_n$. Then look at M^τ like an arrow from $\Delta\times\gamma$ to τ, and at N^γ like an arrow from Δ to γ. The effect of substituting N^γ for x^γ in M^τ is simply achieved by composing M^τ with $<id_\Delta, \mathit{N}^\gamma>$.

$$\begin{tikzcd}
\Delta \arrow[d, "<id_\Delta{,}N^\gamma>"] \\
\Delta\times\gamma \arrow[d, "M^\tau"] \\
\tau
\end{tikzcd}$$

These considerations are formalized in substitution lemma 8.7.6 below, which plays a central role in the semantics of functional languages. Note now that in every CCC, one has

$$(\beta_{cat})\quad eval \circ <\Lambda(f),g> = eval \circ \Lambda(f)\times id \circ <id,g> = f \circ <id,g>$$

which, by our interpretation of substitution, is just the categorical equivalent of β-conversion in λ-calculus.

We now begin a formal discussion on the categorical interpetation of typed λ-calculus, by outlining a simple and very intuitive, set-theoretic definition of model for $\lambda\beta\eta\pi^t$. This is just a definition and will be instrumental to the categorical characterization below. In particular, it will allow us to relate the category-theoretic approach to the Tarskian view in semantics. For a more direct connection between $\lambda\beta\eta\pi^t$ and CCC's, see the references.

The collection Tp of types over a set At of ground types has been defined in section 8.2. As types will be largely used as indexes (of terms), in this section we go back to the use of small greek letters σ,τ.... as metavariables for types.

8.7.1 Definition *Let* $\boldsymbol{C} = \{C_i \mid i\in At\}$ *be a collection of sets. Then* $\boldsymbol{TS_C} = \{C_\sigma\}_{\sigma\in Tp}$ *is a* ***type structure*** *over* $\boldsymbol{C}$ *iff, for all* $\sigma,\tau\in Tp$,

$$C_{\sigma\to\tau} \subseteq C_\sigma \to C_\tau \ (= \boldsymbol{Set}[C_\sigma,C_\tau])$$
$$C_{\sigma\times\tau} = C_\sigma\times C_\tau \, .$$

8.7.2 Definition *A type structure* $\boldsymbol{TS_A} = (\{A_\sigma\}_{\sigma\in Tp}, [\ ])$ *is a* ***model*** *of typed λ-calculus (with products) iff* $[\ ]$ *yields an interpretation for* $\lambda\beta\eta(\pi)^t$. *That is, for any* ***environment*** $h: Var \to \cup_{\sigma\in Tp} \{A_\sigma\}$ *with* $h(x^\sigma)\in A_\sigma$, *one has* $[M^\rho]_h\in A_\rho$, *for each* $\rho\in Tp$ *and for* $[\ ]$ *defined by*

Var.	$[x^\sigma]_h = h(x^\sigma)$
App.	$[M^{\sigma\to\tau}N^\sigma]_h = [M^{\sigma\to\tau}]_h\,([N^\sigma]_h)$
β.	$[\lambda x^\sigma.M^\tau]_h(a) = [M^\tau]_{h[a/x^\sigma]}$ for $a\in A_\sigma$
π.	$[fst^{\sigma\times\tau\to\sigma\tau}]_h = p_1$, $[snd^{\sigma\times\tau\to\sigma\tau}]_h = p_2$

where p_1, p_2 *are the set-theoretic projections (of proper types).*

This is a good definition of model, i.e., the axioms and rules are realized and, given a model $\mathbf{TS_A}$ $\lambda\beta\eta(\pi)^t$, one actually has

$$\lambda\beta\eta(\pi)^t \vdash M^\sigma = N^\sigma \ \Rightarrow\ \mathbf{TS_A} \models M^\sigma = N^\sigma.$$

Indeed, the reader will be asked to check the validity of axiom (β) in the exercise before 8.7.7. Note also that, in a model, axiom (η) and rule (ξ) trivially hold, since λ-terms are interpreted as functions in extenso and, hence, one has

$$\eta.\ \forall a\ \ [\lambda y^\sigma.M^{\sigma\to\tau}y^\sigma]_h(a) = [M^{\sigma\to\tau}]_h(a)\ \text{, for } y^\sigma\notin FV(M^{\sigma\to\tau}),\ \text{ and}$$
$$\xi.\ \forall a\ \ [M^\tau]_{h[a/x^\sigma]} = [N^\tau]_{h[a/x^\sigma]}\ \Rightarrow\ [\lambda x^\sigma.M^\tau]_h = [\lambda x^\sigma.N^\tau]_h\, ,$$

by (β) and by $[\lambda x^\sigma.P^\tau]_h\in A_{\sigma\to\tau} \subseteq A_\sigma\to A_\tau$.

The properties of projections and pairing, in case one considers $\lambda\beta\eta\pi^t$, are trivially realized by the definition of product.

Thus a type-structure $\mathbf{TS_A}$ is a model of $\lambda\beta\eta(\pi)^t$ iff

$$\lambda\beta\eta(\pi)^t \vdash M^\sigma = N^\sigma \ \Rightarrow\ \mathbf{TS_A} \models M^\sigma = N^\sigma.$$

Since the previous definition of model is set-theoretic, a suitable notion of "concreteness" for categories can help in studying categories as models. The minor loss of generality, in this case, is balanced by the gain in intuition.

In general, concrete categories are widely used in denotational semantics; they are (usually) defined by an "enough points" requirement.

8.7.3 Definition. *Let* $\mathbf{C}$ *be a category.* $t \in Ob_{\mathbf{C}}$ *is a* ***generator*** *iff for all* $a,b \in Ob_{\mathbf{C}}$ *and all* $f,g \in \mathbf{C}[a,b]$, $f \neq g \Rightarrow \exists h \in \mathbf{C}[t,a]\ f \circ h \neq g \circ h$.

$\mathbf{C}$ has ***enough points*** *if there exists a generator* t *that is terminal in the given category.*

The terminology derives from the idea that the arrows in $\mathbf{C}[t,a]$, when t is terminal, may be understood as the points (or elements) of a. Indeed, we can associate with each category $\mathbf{C}$ having enough points a category of sets $\mathbf{C}^{set}$ as follows. For $a,b \in Ob_{\mathbf{C}}$ and t terminal

objects: $|a| = \mathbf{C}[t,a]$

morphisms: $f \in \mathbf{C}^{set}[|a|,|b|]$ iff $\exists f' \in C[a,b]\ f = f' \circ \_$.

Observe that f' is unique once f is given.

8.7.4 Proposition. *If* $\mathbf{C}$ *is a CCC with enough points, then* $\mathbf{C}^{set}$ *is also a CCC with enough points.*

Proof If $c_{\sigma\to\tau}$ and $c_{\sigma\times\tau}$ are the exponent and product objects, in $\mathbf{C}$, of c_σ and c_τ, then $|c_{\sigma\to\tau}|$ and $|c_{\sigma\times\tau}|$ are the exponent and product objects, in $\mathbf{C}^{set}$, of $|c_\sigma|$ and $|c_\tau|$. As for the exponent, just set $eval^{set} = eval \circ \_$, and $\Lambda^{set}(f) = \Lambda(f') \circ \_$, where f' defines f as above. The rest is easy. ♦

Note that one obtains type structures from all categories with enough points That is, let $\mathbf{K}$ be a CCC with enough points and $\mathbf{C} = \{k_i \mid i \in At\} \subseteq Ob_{\mathbf{K}}$. Let also $k_{\sigma\to\tau}$ and $k_{\sigma\times\tau}$ be the exponent and product objects, in $\mathbf{K}$, of k_σ and k_τ. Then $\mathbf{K_C} = \{\mathbf{K}[T,k_\sigma] \mid \sigma \in Tp\}$ is the type structure generated by $\mathbf{K}$ and $\mathbf{C}$.

Examples. The simplest type structures are the "full" type-structures and the "term model" of $\lambda\beta\eta(\pi)^t$. That is, $\mathbf{Set_C}$, where $\mathbf{C}$ is a collection of sets, and $\mathbf{Term} = (\{Term_\sigma\}_{\sigma \in Tp}, [\ ])$, where $Term_\sigma$ is the set of terms of type σ modulo $\beta\eta$ convertibility.

8.7.5 Definition *Let* $\mathbf{C}$ *be a CCC, with terminal object* t *and projections* fst, snd. *Suppose one has a map* I *associating every atomic type* A *with an object of* $\mathbf{C}$. *Then the categorical interpretation is as follows:*

- ***Types:*** $[\sigma] = I(\sigma)$ *if* σ *is atomic*

$[\sigma\to\tau] = [\tau]^{[\sigma]}$

$[\sigma\times\tau] = [\sigma] \times [\tau]$

- ***Terms***: *let* M^σ *be a term of* $\lambda\beta\eta\pi^t$, *with* $FV(M^\sigma) \subseteq \Delta = \{x^{\sigma_1},\ldots,x^{\sigma_n}\}$, *and assume that* $A, A_1,\ldots,A_n$ *interpret* $\sigma, \sigma_1,\ldots,\sigma_n$ *(we omit typing, when unambiguous, and write* $p_i \in \mathbf{C}[t\times A_1\times\ldots\times A_n, A_i]$ *for the projection* $p_i : t\times A_1\times\ldots\times A_n \to A_i$ *). Then* $[M^\sigma]_\Delta$ *is the morphism in* $\mathbf{C}[t\times A_1\times\ldots\times A_n, A]$ *defined by*

$$[x^{\sigma_i}]_\Delta = p_i$$
$$[MN]_\Delta = eval \circ <[M]_\Delta, [N]_\Delta>$$
$$[\lambda x^\tau.M]_\Delta = \Lambda([M]_{\Delta\cup\{x^\tau\}})$$
$$[<M, N>]_\Delta = <[M]_\Delta, [N]_\Delta>$$
$$[fst\ (M)]_\Delta = fst \circ [M]_\Delta$$
$$[snd\ (M)]_\Delta = snd \circ [M]_\Delta .$$

8.7.6 Substitution Lemma

i. *If* $y^\sigma \notin FV(N)$, *then* $[N]_{\Delta\cup\{y^\sigma\}} = [N]_\Delta \circ fst$

ii. $[\ [N/x^\sigma]M\ ]_\Delta = [M]_{\Delta\cup\{x^\sigma\}} \circ <id,[N]_\Delta>$ *(types are omitted when unambiguous).*

Proof

i. By induction on M. The following is the typical case:

$$\begin{aligned}
[\lambda x^\tau.P]_{\Delta\cup\{y^\sigma\}} &= \Lambda([P]_{\Delta\cup\{y^\sigma, x^\tau\}}) \\
&= \Lambda([P]_{\Delta\cup\{x^\tau,y^\sigma\}}\circ\xi) && \text{for } \xi: (a_\Delta\times a_\sigma)\times a_\tau \cong (a_\Delta\times a_\tau)\times a_\sigma \\
&= \Lambda([P]_{\Delta\cup\{x^\tau\}}\circ fst\circ\xi) && \text{by induction} \\
&= \Lambda([P]_{\Delta\cup\{x^\tau\}}\circ fst\times id) && \text{as } fst\circ\xi = fst\times id \\
&= \Lambda([P]_{\Delta\cup\{x^\tau\}}) \circ fst && \text{by the naturality of } \Lambda.
\end{aligned}$$

ii. By induction on M:

$M \equiv x^\sigma$

$$\begin{aligned}
[x^\sigma]_{\Delta\cup\{x^\sigma\}} \circ <id,[N]_\Delta> &= snd \circ <id,[N]_\Delta> \\
&= [N]_\Delta \\
&= [\ [N/x^\sigma]\ x^\sigma\ ]_\Delta
\end{aligned}$$

$M \equiv x^{\sigma_i}$

$$\begin{aligned}
[x^{\sigma_i}]_{\Delta\cup\{x^\sigma\}} \circ <id,[N]_\Delta> &= p_i \circ fst_{T\times A1\times\ldots\times An, A} \circ <id, [N]_\Delta> \quad (\sigma_i\neq\sigma) \\
&= p_i \\
&= [x^{\sigma_i}]_\Delta
\end{aligned}$$

$M \equiv QP$

$$\begin{aligned}
[QP]_{\Delta\cup\{x^\sigma\}} \circ <id,[N]_\Delta> &= eval \circ <[Q]_{\Delta\cup\{x^\sigma\}},[P]_{\Delta\cup\{x^\sigma\}}> \circ <id,[N]_\Delta> \\
&= eval \circ <[Q]_{\Delta\cup\{x^\sigma\}}\circ<id,[N]_\Delta>,[P]_{\Delta\cup\{x^\sigma\}}\circ<id,[N]_\Delta>> \\
&= eval \circ <[\ [N/x^\sigma]\ Q\ ]_\Delta,[\ [N/x^\sigma]P\ ]_\Delta> \\
&= [\ [N/x^\sigma]\ PQ\ ]_\Delta
\end{aligned}$$

$M \equiv \lambda y^\tau.P$

$$
\begin{aligned}
[\,[N/x^\sigma]\,\lambda y^\tau.P\,]_\Delta &= \Lambda([\,[N/x^\sigma]P\,]_{\Delta\cup\{y^\tau\}}) \\
&= \Lambda([P]_{\Delta\cup\{y^\tau,x^\sigma\}}\circ <id,[N]_{\Delta\cup\{y^\tau\}}>) && \text{by induction} \\
&= \Lambda([P]_{\Delta\cup\{y^\tau,x^\sigma\}}\circ <id,[N]_\Delta\circ fst>) && \text{by (i)} \\
&= \Lambda([P]_{\Delta\cup\{x^\sigma,y^\tau\}})\circ <id,[N]_\Delta> && \text{by naturality (as in (i))} \\
&= [\lambda y^\tau.P]_{\Delta\cup\{x^\sigma\}}\circ <id,[N]_\Delta>.
\end{aligned}
$$

As for pairing and projection, the computation is easy and is left for exercise. ♦

The lemma may suffice to give an interpretation of $\lambda\beta\eta\pi^t$ over an arbitrary CCC. Indeed, nothing else is required from a purely categorical perspective, i.e., adopting a more suitable notion of model. In particular, it is worth noting that the previous categorical semantics of the typed lambda calculus really "fills the triangle" we talked about at the beginning of this section. We leave it as an exercise for the reader to verify that a term and its categorical interpretation with respect to the previous definition are actually associated with the same derivation in intuitionistic logic.

However, as many, since Tarski, are used to interpreting formal systems into mathematical structures by first assigning values to variables by means of environments as set-theoretic maps, we complete the construction of models (as defined at the beginning of the section) with the following exercise and theorem.

Exercise Prove a version of lemma 8.7.6 in the style of definition 8.7.2. That is, show that in any model and for any environment h, one has (we omit types for simplicity)

$$[M]_{h[[N]h/x^\sigma]} = [\,[N/x^\sigma]M\,]_h\,.$$

This gives axiom (β) in the model.

8.7.7 Theorem *Any CCC with enough points* **C** *and any collection* $\{a_i\}_{i\in At}$ *of objects in* **C** *yield a model of* $\lambda\beta\eta\pi^t$.

Proof Let **C** be the given category and let $A = \{|a_i| \,/\, i\in At\}$ be the collection of sets in $\mathbf{C}^{set}$ that interpret the atomic types. Then the model is given by the type structure $\mathbf{TS}_A = \mathbf{C}^{set}{}_A$, i.e., the higher types are interpreted by $|c_{\sigma\times\tau}|$ and $|c_{\sigma\to\tau}|$. As a matter of fact, let $h : Var \to \bigcup_{\sigma\in Tp}\{|c_\sigma|\}$ be an environment. Fix M^σ and let $FV(M^\sigma) \subseteq \Delta = \{x^{\sigma 1},\dots,x^{\sigma n}\}$. By definition, $h(x^{\sigma i})\in|c_{\sigma i}|$, then set $h_\Delta = <id,h(x^{\sigma 1}),...,h(x^{\sigma n})>\in\mathbf{C}[t,t\times c_{\sigma 1}\times...\times c_{\sigma n}]$ and define

$$[\![M^\sigma]\!]_h = [M^\sigma]_\Delta\circ h_\Delta\in |c_\sigma|.$$

It is now easy to check that the map F_h, defined by $F_h(e) = h[e/x]_\Delta$, for $e\in|c_\sigma|$, is in $\mathbf{C}^{set}[|c_\sigma|,|c_{\sigma 1}\times...\times c_{\sigma n}|\,]$. Thus, there exists $h'\in\mathbf{C}[c_\sigma,c_{\sigma 1}\times...\times c_{\sigma n}]$ such that $F_h = h'\circ\_$ and, hence, $h[e/x]_\Delta = h'\circ e$. This fact and the substitution lemma guarantee that $[\![\_]\!]$ is a well-defined interpretation map and, thus, that $\mathbf{C}^{set}{}_A$ is a model. ♦

We now need to prove the converse and construct a CCC out of every model of $\lambda\beta\eta\pi^t$. Observe first that by functional application, over a type structure, one may define algebraic functions as follows.

8.7.8 Definition *Given a type-structure* $\mathbf{TS_C} = \{C_\sigma\}_{\sigma\in Tp}$, *the* ***typed monomials*** *over* $\mathbf{TS_C}$ *are defined by*

constants $a^\sigma \in C_\sigma$, ... *are typed monomials, for all* $\sigma\in Tp$

variables x^σ, y^σ,.... *are typed monomials, for all* $\sigma\in Tp$

application $M^{\sigma\to\tau}(N^\sigma)$ *is a typed monomial, when* $M^{\sigma\to\tau}$ *and* N^σ *are typed monomials.*

A function $f : C_\sigma\to C_\tau$ *is* ***algebraic*** *if* $f(a^\sigma) = [a^\sigma/x^\sigma]M^\tau$ *for some typed monomial* M^τ.

The intuition is that algebraic functions will give the morphisms of a "rich enough" category as to interpret typed λ-terms. It is clear, for example, that projections are algebraic functions: just set send$((x^\sigma, y^\tau)) = x^\sigma$, then snd : $C_{\sigma\times\tau} \to C_\tau$ is the second projection.

8.7.9 Lemma *Given a type-structure* $\mathbf{TS_A} = \{A_\sigma\}_{\sigma\in Tp}$, *one obtains a category with enough points by taking the singleton set, the* A_σ*'s as objects and the algebraic functions as morphisms. Call this category* $\mathbf{CTS}_A$.

Proof Exercise. ♦

8.7.10 Theorem *Let* $\mathbf{TS_A} = \{A_\sigma\}_{\sigma\in Tp}$ *x be a model of* $\lambda\beta\eta\pi^t$ *and let* $\mathbf{CTS}_A$ *be as in the lemma. Then* $\mathbf{CTS}_A$ *is a CCC.*

Proof Products are defined by taking $A_\sigma\times A_\tau = A_{\sigma\times\tau}$. Then, if the typed monomial M defines f and N defines g, <M,N> defines <f,g>, as the formal pairing and projections behave as required in Cartesian categories.

As for exponents, set $A_\tau{}^{A_\sigma} = A_{\sigma\to\tau}$. Then

$$\text{eval} : A_{\tau\to\rho}\times A_\tau \to A_\rho \quad \text{is defined by } x \vdash \text{fst } x^{\tau\to\rho\times\tau}(\text{snd } x^{\tau\to\rho\times\tau}).$$

Moreover, if $f : A_\sigma\times A_\tau\to A_\rho$ is defined by M,

$$\Lambda(f) : A_\sigma\to A_\rho{}^{A_\tau} \quad \text{is defined by } \lambda x^\tau.M.$$

Finally observe that axiom (β) gives (β_{cat}) in definition 2.3.1, while (η) gives (η_{cat}). ♦

By the following exercise we let the careful reader relate the two interpretations of λ-terms given so far.

Exercise Compare in detail the categorical and set-theoretic meanings of terms. Namely, start with a $\mathbf{TS_A}$, which is a model, and construct a $\mathbf{CTS_A}$ as in 8.7.9. (Note that in this construction, lots of - irrelevant - morphisms may be lost). Then construct a new $\mathbf{TS'_A}$ as in 8.7.7 (this will be a "substructure" of $\mathbf{TS_A}$, up to an obvious identification). Consider now an interpretation $[\![-]\!]_h$ over

$\mathbf{TS_A}$, where the range of h, though, is restricted to $\mathbf{TS'_A}$. Define $[\ \text{-}\ ]_\Delta$ as in 8.7.5 over $\mathbf{CTS_A}$ and, finally, give $[\ \text{-}\ ]'_h$ over $\mathbf{TS'_A}$ by using h and $[\ \text{-}\ ]_\Delta$ as in 8.7.7. Prove then that $[\ \text{-}\ ]_h$ and $[\ \text{-}\ ]'_h$ coincide. Thus, we moved from $\mathbf{TS_A}$ to $\mathbf{CTS_A}$ and to $\mathbf{TS'_A}$, preserving the meaning of terms.

Conversely, by an even longer and more tedious proof, go from an interpretation $[\ \text{-}\ ]_\Delta$ over a Cartesian closed category $\mathbf{C}$, to a model $\mathbf{TS_A}$, where the interpretation $[\ \text{-}\ ]_h$ is given by 8.7.7. Construct then a $\mathbf{CTS_A}$ as in 8.7.9 and observe that it may be faithfully embedded into $\mathbf{C}$. Relate by this $[\ \text{-}\ ]_\Delta$, over $\mathbf{C}$, and the interpretation $[\ \text{-}\ ]'_\Delta$, given as in 8.7.5, over $\mathbf{CTS_A}$. (The connection between set-theoretic and categorical meaning of terms will be more closely investigated for the type-free case in the next chapter).

8.8 Fixpoint Operators and CCCs

The typed lambda calculus is strongly normalizable, that is:

1. every computation terminates, and
2. the computation is independent from the evaluation strategy.

The latter property is surely very attractive for a computational language since it greatly simplifies its semantics: the programmer, in the design of the code, does not have to fuss over the operational evolution of reduction, but can concentrate on the *denotation* of the program in its strongest meaning.

Considered by itself, the first property also seems to be quite interesting, but it has the rather annoying corollary that not all "computable" functions (e.g., Turing-computable) will be computable in the language. The problem is not only related to the abstract expressiveness of the language; in today's computer science, the idea of "general purpose" language is no longer considered as central as it was twenty years ago, and the loss of the ability to compute, for example, the Ackermann function, does not worry anybody: primitive recursive functions are surely enough for most interesting applications. What makes the general recursive formalism more attractive than the primitive recursive one, is that it is much easier to write code in the general formalism (also for computing a primitive recursive function).

Coming back to the typed lambda calculus, it is otherwise too poor for any application, so we face the problem of extending the language with more powerful constructs. In chapter 11, we shall study the "second order" or "polymorphic" extension, which still has the nice property of strong normalization together with a great formal expressiveness (although, as a programming language, it imposes on the programmer a completely new approach to the design of the code). We study for the moment a simple extension by means of **fixpoint operators**, which enables us to write recursive definitions of functions.

The reader should be aware, though, that a price must be paid when extending typed languages by fixpoint operators. The problem is related to the Curry-Howard analogy (see section 8.3). Remember that a (constructive) proof of a formula A corresponds with a closed λ-term of type A in type theory, and, thus, a formula is provable in the logic if and only if the corresponding type is "inhabited" in type theory. The existence of fixpoint operators has as a consequence that all types are inhabited, or, from the logical point of view, that all formulae are provable (see later). Thus fixpoint operators have no logical correspondent; nevertheless, the calculus they originate is not inconsistent from the point of view of the equational theory of programs (that is, not all terms happen to be provably equal). This is shown by the models below.

In conclusion, the programmer, in the present context, must decide whether to acquire the full expressive power and the elegance of programming by "recursive definitions" or to preserve the advantages of the "types as formulae" paradigm. This choice depends on the specific applications he or she has in mind.

8.8.1 Definition *A fixpoint operator of type* σ *is a term* Θ_σ *of type* $(\sigma\to\sigma)\to\sigma$ *such that , for every term* $M^{\sigma\to\sigma}$ *one has*

$$M^{\sigma\to\sigma}(\Theta_\sigma M^{\sigma\to\sigma}) = \Theta_\sigma M^{\sigma\to\sigma}.$$

By fixpoint operators one can simulate recursive definitions in the following way:

$$\text{letrec } f^\sigma \text{ be } M^\sigma[f] \text{ in } N^\gamma[f]$$

becomes

$$(\lambda h^\sigma.N^\gamma)\ (\Theta_\sigma(\lambda f^\sigma.M^\sigma))$$

Note that for every type σ we have at least an object of that type, obtained as a fixpoint of the identity

$$\Theta_\sigma\ (\lambda x^\sigma.x^\sigma).$$

We consider next the problem of giving an interpretation to fixpoint operators. As we need to interpret typed λ-calculi, this will be done in suitable CCC's.

8.8.2 Definition *Let* b *be an object of a CCC* ***C****. A* ***fixpoint operator*** *for* b *is a morphism* $Fix_b: b^b\to b$ *such that* $Fix_b = eval_{b,b}\circ\langle id,Fix_b\rangle$*. A category* ***has fixpoint operators*** *if each object* b *has a fixpoint operator* Fix_b *.*

In every CCC with fixpoint operators, it is easy to give meaning to the axiom

$$\text{fix.}\quad M^{\sigma\to\sigma}(\Theta_\sigma M^{\sigma\to\sigma}) = \Theta_\sigma M^{\sigma\to\sigma}$$

by letting

$$\{\Theta_\sigma\}_\Delta = \Lambda(Fix_\sigma\circ snd)\circ !\Delta,$$

where $snd: t\times\sigma^\sigma\to\sigma^\sigma$, and $!\Delta$ is the unique arrow from Δ to the terminal object t.

Indeed, one has the following:

$$
\begin{aligned}
\{ M(\Theta_\sigma M) \}_\Delta &= \mathrm{eval} \circ <\{M\}_\Delta, \{ (\Theta_\sigma M) \}_\Delta> \\
&= \mathrm{eval} \circ <\{M\}_\Delta, \mathrm{eval} \circ < \Lambda(\mathrm{Fix}_\sigma \circ \mathrm{snd}) \circ !\Delta, \{M\}_\Delta> > \\
&= \mathrm{eval} \circ <\{M\}_\Delta, \mathrm{Fix}_\sigma \circ \{M\}_\Delta > \\
&= \mathrm{eval} \circ < \mathrm{id}, \mathrm{Fix}_\sigma > \circ \{M\}_\Delta \\
&= \mathrm{Fix}_\sigma \circ \{M\}_\Delta \\
&= \mathrm{eval} \circ < \Lambda(\mathrm{Fix}_\sigma \circ \mathrm{snd}) \circ !\Delta, \{M\}_\Delta> \\
&= \{ (\Theta_\sigma M) \}_\Delta .
\end{aligned}
$$

It is not difficult to find CCC's with fixpoint operators. The most well-known example is probably the category **CPO**, with complete partial order for objects and continuous functions for morphisms (remember that, in **CPO**, $f: A\to B$ is continuous if and only if f is monotonic and for every directed subset D of A, $f(\cup(D)) = \cup f(D)$; every c.p.o.has a least element, $\perp = \cup\emptyset$).

Observe that, given a c.p.o. C and a continuous function f, we can form a chain

$$\{f^n(\perp_C)\}_{n\in\omega} = \perp_C \le f(\perp_C) \le f(f(\perp_C)) \le .... \le f^n(\perp_C) \le ....$$

starting from the bottom element $\perp_C$. The next two results develop this example.

8.8.3 Theorem *Let C be a CPO, let $\perp_C$ be its least element, and let $f: C\to C$ be a continuous function. Then $\cup\{f^n(\perp_C)\}_{n\in\omega}$ is the least fixed point of f.*

Proof: Note that $\cup\{f^n(\perp_C)\}_{n\in\omega} = \cup\{f^{n+1}(\perp_C)\}_{n\in\omega}$. Then

$$
\begin{aligned}
f(\cup\{f^n(\perp_C)\}_{n\in\omega}) &= \cup f\{f^n(\perp_C)\}_{n\in\omega}) \\
&= \cup\{f^{n+1}(\perp_C)\}_{n\in\omega}).
\end{aligned}
$$

Moreover, if c is another fixed point, then we prove by induction that, for all n, $f^n(\perp_C) \le c$. Indeed,

$$\perp_C \le c$$

$$f^n(\perp_C) \le c \Rightarrow f^{n+1}(\perp_C) \le f(c) = c.$$

Then, by definition of least upper bound, $\cup\{f^n(\perp_C)\}_{n\in\omega} \le c$. ♦

8.8.4 Definition *Let C be a c.p.o. Define then $Fix_C: C^C\to C$ the function that takes every continuous function $f: C\to C$ to $\cup\{f^n(\perp_C)\}_{n\in\omega}$.*

8.8.5 Proposition *$FIX_C: C^C\to C$ is continuous.*

Proof Exercise. ♦

In a more general setting, the existence of exponents, in CCC's, suggests the investigation of those “paradoxical” objects that “contain,” as a retract, their own function space; namely, the reflexive objects of section 2.3 (see also below). They will turn out to be rather relevant in the next chapter, where examples are given, and in chapter 10, where the idea will be generalized to fixpoint constructions over types, namely to the categorical counterpart of recursive definitions of data types.

Remember (see definition 1.4.2) that in a category **C**, $a < b$ via the retraction (i,j) iff $j \circ i = id_a$. In these assumptions, i turns out to be mono and j epic. Thus a retract a of b is a subobject of b, up to isomorphisms. An object V is reflexive iff $V^V < V$ (see definition 2.3.5). We next show how to construct another simple model of the typed lambda calculus with fixpoint operators, by using the category $\mathbf{Ret_V}$ of retracts over a reflexive object V. Recall that, given an object V in a category **C**, the category $\mathbf{Ret_V}$ is defined as follows (see definition 1.4.4):

$$Ob_{\mathbf{RetV}} = \{ f \in \mathbf{C}[V,V] \mid f \circ f = f \}$$
$$Mor_{\mathbf{RetV}} = \{ (f, k, g) \mid f, g \in Ob_{\mathbf{RetV}} , k \in \mathbf{C}[V,V] , k = g \circ k \circ f \}$$
$$dom((f, k, g)) = f , cod((f, k, g)) = g$$
$$id_f = (f, f, f)$$
$$(f, k, g) \circ (g', k', f) = (g', k \circ k', g)$$

Proposition 8.8.6 below proves that, if V is a reflexive object in a CCC **C**, then $\mathbf{Ret_V}$ is Cartesian closed too. In theorem 8.8.8 we will prove that, given a reflexive object V in an arbitrary CCC, every object f in $\mathbf{Ret_V}$ has a fixpoint operator Fix_f .

8.8.6 Proposition *If* **C** *is a CCC and V is a reflexive object in* **C**, *then* $\mathbf{Ret_V}$ *is a CCC.*

Proof By proposition 2.3.6, we know that the terminal object t and V×V are both retracts of V (by definition of reflexive object also $V^V<V$). Suppose the following:

$$\begin{array}{ll} t < V & \text{via in, out} \\ V\times V < V & \text{via in', out'} \\ V^V < V & \text{via in'', out''}. \end{array}$$

Let $1 = in \circ out : V \to V$. $1 \circ 1 = in \circ out \circ in \circ out = in \circ out = 1$. 1 is the terminal object of $\mathbf{Ret_V}$. If f is an object in $\mathbf{Ret_V}$, then $!f: f \to 1$ is $(f,1,1)$. Note that $(f,1,1)$ is a well-defined morphism, since $1 = 1 \circ 1 \circ f$ for the terminality of t. Moreover, if $(f,g,1)$ is another morphism, then $g = 1 \circ g \circ f = 1$, again for the terminality of t .

Given two objects f and g in $\mathbf{Ret_V}$, their product is $f\otimes g = in' \circ f\times g \circ out' : V \to V$. (In the present proof only, $\otimes$ has this meaning)

Note that

$$\begin{aligned} f\otimes g \circ f\otimes g &= in' \circ f\times g \circ out' \circ in' \circ f\times g \circ out' \\ &= in' \circ f\times g \circ f\times g \circ out' \\ &= in' \circ f\times g \circ out' \\ &= f\otimes g. \end{aligned}$$

The projections are $(f\otimes g, fst, f)$, $(f\otimes g, snd, g)$ where

$$fst = p_1 \circ out' \circ f\otimes g : V \to V$$
$$snd = p_2 \circ out' \circ f\otimes g : V \to V$$

and p_1, p_2 are the projections associated in **C** to the product V×V.

The pairing operation $\mathbf{<,>_{ret}}$ is defined as follows: given two morphisms (c,h,f) and (c,k,g) set

$$\mathbf{<}\ (c,h,f),\ (c,k,g)\ \mathbf{>_{ret}} = (c,\ in' \circ <h,k>, f \otimes g)$$

where <,> is the pairing in **C**.

This is a good definition, since

$$\begin{aligned} f\otimes g \circ in' \circ <h,k> \circ\ c &= in' \circ f\times g \circ out' \circ in' \circ <h \circ c, k \circ c> && \text{by def. of } f\otimes g \\ &= in' \circ f\times g \circ <h \circ c, k \circ c> \\ &= in' \circ <f \circ h \circ c, g \circ k \circ c> \\ &= in' \circ <h, k>. \end{aligned}$$

We have still to prove the equations associated with the product; we only prove that

$$(f\otimes g, fst, f) \circ \mathbf{<}\ (c,h,f), (c,k,g)\ \mathbf{>_{ret}} = (c,h,f)$$

and leave the other proofs as an exercise for the reader. We have the following:

$$\begin{aligned} &(f\otimes g, fst, f) \circ \mathbf{<}\ (c,h,f), (c,k,g)\ \mathbf{>_{ret}} = \\ &\qquad = (f\otimes g, fst, f) \circ (c, in' \circ <h,k>, f\otimes g) \\ &\qquad = (c, fst \circ in' \circ <h,k>, f) && \text{by composition in } \mathbf{Ret_V} \\ &\qquad = (c, p_1 \circ out' \circ f\otimes g \circ in' \circ <h,k>, f) && \text{by def. of fst} \\ &\qquad = (c, p_1 \circ out' \circ in' \circ f\times g \circ out' \circ in' \circ <h,k>, f) && \text{by def. of } f\otimes g \\ &\qquad = (c, p_1 \circ f\times g \circ <h,k>, f) = (c,h,f). \end{aligned}$$

The functor $\otimes$ is defined on morphisms in the usual way (namely, $f\times g = <p_1 \circ f, p_2 \circ g>$). Specifically, given two morphisms (c,h,f) and (d,k,g):

$$\begin{aligned} &(c,h,f) \otimes (d,k,g) = \mathbf{<}\ (c,h,f) \circ (c\otimes d, fst, c), (d,k,g) \circ (c\otimes d, snd, d)\ \mathbf{>_{ret}} \\ &\qquad = \mathbf{<}\ (c\otimes d, h \circ fst, f), (c\otimes d, k \circ snd, g)\ \mathbf{>_{ret}} && \text{by composition in } \mathbf{Ret_V} \\ &\qquad = (c\otimes d, in' \circ <h \circ fst, k \circ snd>, f\otimes g) && \text{by def. of } \mathbf{<,>_{ret}} \\ &\qquad = (c\otimes d, in' \circ <h \circ p_1 \circ out' \circ c\otimes d, k \circ p_2 \circ out' \circ c\otimes d>, f\otimes g) && \text{by def. of fst, snd} \\ &\qquad = (c\otimes d, in' \circ <h \circ p_1, k \circ p_2> \circ out' \circ c\otimes d, f\otimes g) \\ &\qquad = (c\otimes d, in' \circ h\times k \circ out' \circ c\otimes d, f\otimes g). \end{aligned}$$

We are now in a position to define the exponents. If $h, k : V\rightarrow V$, let $[h,k] = \Lambda(k \circ eval_{V,V} \circ (id\times h)) : V^V\rightarrow V^V$. Given two objects f and g in $\mathbf{Ret_V}$, their exponent is:

$$g^f = in'' \circ [f,g] \circ out'' : V\rightarrow V.$$

This is a good definition, since

$$\begin{aligned} g^f \circ g^f &= in'' \circ [f,g] \circ out'' \circ in'' \circ [f,g] \circ out'' && \text{by def. of } g^f \\ &= in'' \circ [f,g] \circ [f,g] \circ out'' \\ &= in'' \circ \Lambda(g \circ eval_{V,V} \circ (id\times f)) \circ \Lambda(g \circ eval_{V,V} \circ (id\times f)) \circ out'' && \text{by def. of } [f,g] \\ &= in'' \circ \Lambda(g \circ eval_{V,V} \circ (id\times f) \circ \Lambda(g \circ eval_{V,V} \circ (id\times f))\times id) \circ out'' \\ &= in'' \circ \Lambda(g \circ eval_{V,V} \circ \Lambda(g \circ eval_{V,V} \circ (id\times f))\times id \circ (id\times f)) \circ out'' \\ &= in'' \circ \Lambda(g \circ g \circ eval_{V,V} \circ (id\times f) \circ (id\times f)) \circ out'' \quad \text{by } (\beta) \\ &= in'' \circ \Lambda(g \circ eval_{V,V} \circ (id\times f)) \circ out'' && \text{since g, f are retractions} \\ &= in'' \circ [f,g] \circ out'' \quad \text{by def. of } [f,g] \\ &= g^f && \text{by def. of } g^f. \end{aligned}$$

The evaluation function is $(g^f \otimes f, ev, g)$ where

$$ev = eval_{V,V} \circ (out''\times id) \circ out' \circ g^f \otimes f$$

The currying operation $\mathbf{\Lambda_{ret}}$ is so defined: given a morphism $(c\otimes f,h,g)$,

$$\mathbf{\Lambda_{ret}}(c\otimes f,h,g) = (c, in''\circ \Lambda(h \circ in'), g^f).$$

This is a good definition of morphism in $\mathbf{Ret_V}$; indeed

$g^f \circ in''\circ \Lambda(h \circ in') \circ c =$

$= in'' \circ \Lambda(g \circ eval_{V,V} \circ (id\times f)) \circ out'' \circ in''\circ \Lambda(h \circ in') \circ c$	by def. of g^f
$= in'' \circ \Lambda(g \circ eval_{V,V} \circ (id\times f)) \circ \Lambda(h \circ in' \circ c\times id)out'' \circ in'' = id$	
$= in'' \circ \Lambda(g \circ eval_{V,V} \circ (id\times f) \circ \Lambda(h \circ in' \circ c\times id)\times id)$	
$= in'' \circ \Lambda(g \circ eval_{V,V} \circ \Lambda(h \circ in' \circ c\times id)\times id \circ (id\times f))$	
$= in'' \circ \Lambda(g \circ h \circ in' \circ c\times id \circ id\times f)$	by (β)
$= in'' \circ \Lambda(g \circ h \circ in' \circ c\times f)$	
$= in''\circ \Lambda(g \circ h \circ c\otimes f \circ in')$	by def. of $c\otimes f$
$= in''\circ \Lambda(h \circ in').$	

We only prove the axiom β, and leave as an exercise for the reader to prove η .

$(g^f\otimes f, ev, g) \circ (\mathbf{\Lambda_{ret}}(c\otimes f,h,g) \otimes (f, f, f)) =$

$= (g^f\otimes f, ev, g) \circ ((c, in''\circ \Lambda(h \circ in'), g^f) \otimes (f, f, f))$	by def. of $\mathbf{\Lambda_{ret}}$
$= (g^f\otimes f, ev, g) \circ (c\otimes f, in' \circ (in''\circ \Lambda(h \circ in'))\times f \circ out' \circ c\otimes f , g^f\otimes f)$	by def. of $\otimes$ on arrows
$= (c\otimes f, ev \circ in' \circ (in''\circ \Lambda(h \circ in'))\times f \circ out' \circ c\otimes f , g)$	by composition in $\mathbf{Ret_V}$
$= (c\otimes f, eval_{V,V} \circ (out''\times id) \circ out' \circ g^f\otimes f \circ in' \circ (in''\circ \Lambda(h \circ in'))\times f \circ out' \circ c\otimes f, g)$	by def. of ev
$= (c\otimes f, eval_{V,V} \circ (out''\times id) \circ g^f\times f \circ (in''\circ \Lambda(h \circ in'))\times f \circ out' \circ c\otimes f , g)$	by def. of $g^f\otimes f$
$= (c\otimes f, eval_{V,V} \circ (out'' \circ g^f \circ in''\circ \Lambda(h \circ in'))\times f \circ out' \circ c\otimes f , g)$	as $f \circ f = f$
$= (c\otimes f, eval_{V,V} \circ (\Lambda(g \circ eval_{V,V} \circ (id\times f)) \circ \Lambda(h \circ in'))\times f \circ out' \circ c\otimes f , g)$	by def. of g^f
$= (c\otimes f, eval_{V,V} \circ (\Lambda(g \circ eval_{V,V} \circ \Lambda(h \circ in')\times id \circ (id\times f))\times f \circ out' \circ c\otimes f , g)$	
$= (c\otimes f, eval_{V,V} \circ \Lambda(g \circ h \circ in'\circ id\times f)\times f \circ out' \circ c\otimes f , g)$	by β
$= (c\otimes f, g \circ h \circ in'\circ id\times f \circ c\times f \circ out' , g)$	by β
$= (c\otimes f, g \circ h \circ c\otimes f , g)$	
$= (c\otimes f,h,g).$	

This completes the proof that $\mathbf{Ret_V}$ is a CCC. ♦

8.8.7 Lemma *If V is a reflexive object in a CCC* $\mathbf{C}$, *then there is a fixpoint operator Fix: $V^V \to V$.*

Proof Let

$$F = eval \circ \langle id, in''\rangle : V^V \to V$$

$$H = \Lambda(eval \circ (id\times(F\circ out''))) : V^V \to V^V$$

$Fix = F\circ H$ is a fixpoint operator for V; indeed,

$$\begin{aligned} F\circ H &= \mathrm{eval} \circ \langle \mathrm{id}, \mathrm{in}'' \rangle \circ \Lambda(\, \mathrm{eval} \circ (\mathrm{id}\times(F\circ \mathrm{out}''))\,) \\ &= \mathrm{eval} \circ \Lambda(\, \mathrm{eval} \circ (\mathrm{id}\times(F\circ \mathrm{out}''))\,)\times \mathrm{id} \circ \langle \mathrm{id}, \mathrm{in}'' \circ \Lambda(\, \mathrm{eval} \circ (\mathrm{id}\times(F\circ \mathrm{out}''))\,)\,\rangle \\ &= \mathrm{eval} \circ \mathrm{id}\times(F\circ \mathrm{out}'') \circ \langle \mathrm{id}, \mathrm{in}'' \circ \Lambda(\, \mathrm{eval} \circ (\mathrm{id}\times(F\circ \mathrm{out}''))\,)\,\rangle \\ &= \mathrm{eval} \circ \langle \mathrm{id}, F \circ \Lambda(\, \mathrm{eval} \circ (\mathrm{id}\times(F\circ \mathrm{out}''))\,)\,\rangle \\ &= \mathrm{eval} \circ \langle \mathrm{id}, F\circ H \rangle. \ \blacklozenge \end{aligned}$$

8.8.8 Theorem *Every object f in $\mathbf{Ret_V}$ has a fixpoint operator Fix_f.*

Proof Let $\mathrm{Fix}: V^V \to V$ be a fixpoint operator for V. Define

$$\mathrm{Fix}_f = (f^f, \mathrm{Fix} \circ \mathrm{out}'' \circ f^f, f).$$

We leave it to the reader to prove that this is a good definition, that is:

$$(*)\quad \mathrm{Fix} \circ \mathrm{out}'' \circ f^f = f \circ \mathrm{Fix} \circ \mathrm{out}'' \circ f^f \circ f^f.$$

We must prove that

$$\mathrm{Fix}_f = (\, f^f \otimes f, \mathrm{ev}, f) \circ \mathbf{<} (f^f, f^f, f^f), \mathrm{Fix}_f \mathbf{>},$$

where $\mathrm{ev} = \mathrm{eval}_{V,V} \circ (\mathrm{out}''\times \mathrm{id}) \circ \mathrm{out}' \circ f^f \otimes f$.

Compute then

$$\begin{aligned} \mathrm{Fix}_f &= (f^f, \mathrm{eval}_{V,V} \circ \langle \mathrm{id},\mathrm{Fix}\rangle \circ \mathrm{out}'' \circ f^f, f) && \text{by def. of } \mathrm{Fix}_f \\ &= (f^f, \mathrm{eval}_{V,V} \circ \langle \mathrm{out}'' \circ f^f, f \circ \mathrm{Fix} \circ \mathrm{out}'' \circ f^f\rangle, f) && \text{by } (*) \\ &= (f^f, \mathrm{eval}_{V,V} \circ (\mathrm{out}''\times \mathrm{id}) \circ f^f\times f \circ \langle f^f, \mathrm{Fix} \circ \mathrm{out}'' \circ f^f\rangle, f) && \\ &= (f^f, \mathrm{eval}_{V,V} \circ (\mathrm{out}''\times \mathrm{id}) \circ \mathrm{out}' \circ f^f\otimes f \circ \mathrm{in}' \circ \langle f^f, \mathrm{Fix} \circ \mathrm{out}'' \circ f^f\rangle, f -) && \text{by def. of } f^f\otimes f \\ &= (f^f, \mathrm{ev} \circ \mathrm{in}' \circ \langle f^f, \mathrm{Fix} \circ \mathrm{out}'' \circ f^f\rangle, f) && \text{by def. of ev} \\ &= (\, f^f\otimes f, \mathrm{ev}, f) \circ (f^f, \mathrm{in}' \circ \langle f^f, \mathrm{Fix} \circ \mathrm{out}'' \circ f^f\rangle, f^f\otimes f) && \text{by composition in } \mathbf{Ret_V} \\ &= (\, f^f\otimes f, \mathrm{ev}, f) \circ \mathbf{<} (f^f, f^f, f^f), (f^f, \mathrm{Fix} \circ \mathrm{out}'' \circ f^f, f) \mathbf{>} && \text{by pairing } \mathbf{<,>} \text{ in } \mathbf{Ret_V} \\ &= (\, f^f\otimes f, \mathrm{ev}, f) \circ \mathbf{<} (f^f, f^f, f^f), \mathrm{Fix}_f \mathbf{>}. \ \blacklozenge \end{aligned}$$

References. For an introduction to Proof Theory, natural deduction, and intuitionisitic logic, the reader can consult Prawitz (1965) and Takeuti (1975). In particular, the latter inspired our presentation of the cut-elimination theorem. The “formulae as types” analogy is explained in Howard (1980), but the main ideas go back to work of Curry. The connections between λ-calculus and CCC’s were first explored by Lambek. The equivalence is shown in Lambek (1980) and Scott (1980). A full account of the relation between λ-calculus, CCC’s, and Proof Theory may be found in Lambek and Scott (1986). We tried here to complement that work by relating it to Tarskian semantics and emphasizing the role of “structures,” for the convenience of the reader who may begin with our approach and continue with further, more abstract readings. In section 8.7, we dealt only with CCC's, i.e., with the models of $\boldsymbol{\lambda\beta\eta\pi}^t$. Weaker calculi, that is, typed combinatory logic and $\boldsymbol{\lambda\beta\pi}^t$, are discussed in Hayashi (1985) and Martini (1988), which introduce various notions of “weak Cartesian closedness” for this purpose.