TYPED CATEGORICAL COMBINATORY LOGIC

P-L. Curien

CNRS-Université Paris VII, LITP, Tour 55-56 1er étage,
3 Place Jussieu, 75221 PARIS CEDEX 05

ABSTRACT

The subject of the paper is the connection between the typed λ-calculus and the cartesian closed categories, pointed out by several authors. Three languages and their theories, defined by equations, are shown to be equivalent: the typed λc-calculus (i.e. the λ-calculus with explicit products and projections) λc_K, the free cartesian closed category CCC_K, and a third intermediary language, the typed categorical combinatory logic CCL_K, introduced by the author. In contrast to CCC_K, CCL_K has the same types as λc_K, and roughly the terminal object in CCC_K is replaced by the application and couple operators in CCL_K. In CCL_K β-reductions as well as evaluations w.r.t. environments (the basis of most practical implementations of λ-calculus based languages) may be simulated in the well-known framework of a same term rewriting system. Finally the introduction of CCL_K allowed the author to understand the untyped underlying calculus, investigated in a companion paper. Another companion paper describes a general setting for equivalences between equational theories and their induced semantic equivalences, the equivalence between CCL_K and CCC_K is an instance of which.

1. Introducing categorical combinators

Categories and λ-calculus are alternative theories of functionality, based on composition of functions (more abstractly arrows), substitution of actual parameters to formal ones respectively. A part from the interest in itself to be able to connect two different formalisms, and to let benefit one from the other (see the end of section 2 for an example), there is an operational significance: roughly λ-calculus is well-suited for programming, and combinators (of Curry, or those introduced here) allow for implementations getting rid of some difficulties in the scope of variables. Indeed we intend to develop implementations of functional programming languages based on categorical combinators, which we introduce now, letting them arise from the known principle that a formal semantic description yields a compilation.

Suppose that x has value $\underline{3}$ (3 is underlined to stress that $\underline{3}$ is the representation of 3), and that we want to express the function associating $y(\underline{3})$ (also written simply $y\underline{3}$) with every function y. The λ-calculus provides the elegant notation $\lambda y.yx$ (M in the sequel) for that function.

Hence $N = \lambda f.f(fx)$ will designate the function associating with f the result of applying f to $f(3)$. One may like to relate those two so-called λ-expressions and to point out some modularity by showing that the second expression may be built from the first one and a third expression.

Indeed two constructions are involved in the informal definition of N: first we associate with $f = x \mapsto f(x)$ the function $f \circ f = x \mapsto f(f(x))$, and then we apply the function described by M. This can be summarized by

$$N' = M \circ (\lambda f.f \circ f)$$

where we have mixed the λ-notation with the notation for composing functions, the basic concept of the theory of categories. We may turn N' into a pure λ-expression (i.e. code the composition in terms of λ-notation): we obtain

$$P = \lambda f.(\lambda y.yx)((\lambda yz.y(yz))f)$$

It is an interesting exercise for readers not familiar with λ-calculus to experiment with the β-reduction, which is the formal application of a function to its argument, involving substitution of the formal argument at each occurrence of the formal parameter. Repeated application of this rule yields N back from P:

$$P \to \lambda f.(\lambda y.yx)(\lambda z.f(fz)) \to \lambda f.(\lambda z.f(fz))x \to \lambda f.f(fx)$$

It will be clear from the rest of the section that we could have made the other choice, i.e. express N' in a pure categorical notation.

Now we turn back to the initial goal: the formal description of the meaning of say $\lambda y.yx$ which we shall give in terms of the meanings of the sub-expressions x, y, yx. Those expressions clearly depend on the value of at least x. The values of the variables are kept in an environment (a pile if one thinks of an implementation). We represent the environment as follows: one considers one more variable z, D_x, D_y, D_z are the sets of possible values of x,y and z, and ".." is the rest of the environment which needs not be detailed for the present description, and × denotes the usual cartesian product of set theory.

$$Env = ((..\times D_z)\times D_x)\times D_y$$

(Represented as a tree, Env looks like a comb.)

The meaning of an expression depends on Env. For instance the meaning of x (y) is obtained by having access to the D_x (D_y) part of the environment, which may be done through the first and second projections, denoted by Fst and Snd. Whence the meanings of x and y, denoted by $[\![x]\!]$ and $[\![y]\!]$ (more generally $[\![M]\!]$ denotes the semantics of M):

$$[\![x]\!] = Snd \circ Fst$$
$$[\![y]\!] = Snd$$

The behaviour of $\circ$, Fst and Snd may be described by equations:

$$\begin{cases}(ass) & (x \circ y)z = x(yz) \\ (fst) & Fst(x,y) = x \\ (snd) & Snd(x,y) = y\end{cases}$$

(applying a function to its argument is denoted by simple juxtaposition; of course x,y,z are fresh variables and have nothing to do with those in $\lambda y.yx$)

Now we define $[\![yx]\!]$ from $[\![y]\!]$ and $[\![x]\!]$, which are functions from Env to D_y, from Env to D_x respectively. First we may form the pair $<[\![y]\!],[\![x]\!]>$, which is a function from Env to $D_y \times D_x$. To fix ideas we set $D_x = \mathbb{N}$ and $D_y = \mathbb{N} \Rightarrow \mathbb{N}$, the set of functions from $\mathbb{N}$ to $\mathbb{N}$ (which is coherent with the above value of x). Remarking that the semantics of yx has to be a function from Env to $\mathbb{N}$ we obtain

$$\left[\, [\![yx]\!] = App \circ <[\![y]\!],[\![x]\!]> \right.$$

where App is the application function from $(\mathbb{N} \Rightarrow \mathbb{N}) \times \mathbb{N}$ to $\mathbb{N}$. The following equations describe the behaviour of App, $< \; >$:

$$\begin{cases}(app) & App(x,y) = xy \\ (dpair) & <x,y>z = (xz,yz)\end{cases}$$

The meaning of $\lambda y.yx$ depends on Env for x, but not for y which is bound (see below). It is a function from Env to $D_y \Rightarrow \mathbb{N}$ whereas $[\![yx]\!]$ is a function from $Env' \times D_y$ to $\mathbb{N}$, where $Env' = (..\times D_z) \times D_x$. We are tempted to use the currying which transforms a function f with two arguments a and b into a function $\Lambda(f)$ of a having as result a function of b such that

$$(\Lambda(f)(a))(b) = f(a,b)$$

or in equational form

$$\left\{ (d\Lambda) \quad (\Lambda(x)y)z = x(y,z) \right.$$

So we would like to write

$$[\![\lambda y.yx]\!] = \Lambda([\![yx]\!])$$

relying on the intuition that $[\![yx]\!]$ is the function associating yx with x and y, and that $[\![\lambda y.yx]\!]$ is the function associating with x the function associating yx with y. But then we loose symmetry since $\Lambda([\![yx]\!])$ is not a function from Env to $\mathbb{N}$, but from Env' to $\mathbb{N}$. So we have to take some care to ensure that the semantics takes always its argument in Env. We define $[\![\lambda y.yx]\!]$ as the currying of a function in $Env \times D_y \Rightarrow \mathbb{N}$ which itself is the composition of $[\![yx]\!]$ and a function $Subst_y$ from $Env \times D_y$ to Env which associates with a couple (ρ,a) a modified environment $\rho[y \leftarrow a]$, where only the component y has been changed (to a). We leave the reader check that in the present case

$$Subst_y = <Fst \circ Fst, Snd>$$

yielding

$[\![\lambda y.yx]\!] = \Lambda([\![yx]\!] \circ Subst_y) = \Lambda((App \circ <Snd, Snd \circ Fst>) \circ <Fst \circ Fst, Snd>)$

Obviously the last expression may be simplified using the following equations (which the reader may "check" by applying both members to a same formal argument):

$$\begin{cases}(Ass) \ (x \circ y) \circ z = x \circ (y \circ z)\end{cases}$$

$$\begin{cases}(DPair) \ <x,y> \circ z = <x \circ z, y \circ z> \\ (Snd) \ Snd \circ <x,y> = y \\ (Fst) \ Fst \circ <x,y> = x\end{cases}$$

Using these rules we obtain

$[\![\lambda y.yx]\!] = \Lambda(App \circ <Snd, Snd \circ (Fst \circ Fst)>$

We have introduced all the categorical combinators but the identity constant, which will arise below.

Now we present another way of associating a categorical term, i.e. of the kind built above, with any λ-expression, i.e. a term built from variables by application (MN) and abstraction $(\lambda x.M)$. We shall use the following notation.

$Snd \circ Fst^n = n!$

(By Ass this is unambiguous.)

$App \circ <A,B> = S(A,B)$

So we have

(1) $[\![\lambda y.yx]\!] = \Lambda(S(0!, 2!))$

Now we manipulate a more involved term:

$Q = (\lambda x. (\lambda z.zx)y)((\lambda t.t)z)$

Q has a disguised form of M as a subterm, namely $\lambda z.zx$ (exercise: define $[\![\lambda z.zx]\!]$ as above and check $[\![\lambda z.zx]\!] = [\![\lambda y.yx]\!]$ using the above equations).

This observation that the name of bound variables is indifferent is the basis of a variable name free notation due do N. De Bruijn, which we describe now.

N. De Bruijn's idea is to replace bound variable names by a number recording where they are bound in an expression, which is the only important information about them. Free variables are included in this treatment by considering in our example

$R = \lambda zxy.Q$ ($\lambda zxy.Q$ is an abbreviation for $\lambda z. (\lambda x. (\lambda y. Q))$)

where the order z, x, y is consistent with the discussion above. The number associated with any occurrence u of a variable, a leaf in the tree representation of R, is the number of nodes labelled λv, with $v \neq u$, which are met in the path from that leaf to the root until a node λu is encountered. The result of that transformation is

$$R' = (\lambda.(\lambda.01)1)((\lambda.0)2)$$

Now we make only a textual tranformation and replace λ by Λ, "." by S, n by n! and obtain what we shall call the De Bruijn translation of Q and denote by $Q_{DB(z,x,y)}$:

$$Q_{DB(z,x,y)} = S(\Lambda(S(\Lambda(S(0!,1!)),1!)),S(\Lambda(0!),2!))$$

The reader may check that (1) above indeed defines $(\lambda y.yx)_{DB(x,y)} = (\lambda y.yx)_{DB(z,x,y)}$.

We end this introduction by suggesting that one may compute with the categorical expressions (that they may be called so will result from the precise connection with cartesian closed categories established in the next section).

Q reduces by β-reduction to yz, which is 5 if y is *succ* (the successor function) and z is 4. But first the outermost β-reduction yields

$$Q' = (\lambda u.u((\lambda t.t)z))y$$

(The bound variable z was renamed to avoid the free occurrence of z becoming bound after substitution.)

We show that $Q_{DB(z,x,y)}$ reduces to $Q'_{DB(z,x,y)}$; we shall need some more equations.

We decompose $Q_{DB(z,x,y)}$ as follows

$Q_{DB(z,x,y)} = App \circ \langle \Lambda(A), B \rangle$ where

$B = App \circ \langle \Lambda(Snd), Snd \circ Fst \circ Fst \rangle$

$A = App \circ \langle \Lambda(C), Snd \circ Fst \rangle$ wherc

$C = App \circ \langle Snd, Snd \circ Fst \rangle$

First we use the following rule, which may be "checked" as above

$$(Beta) \quad App \circ \langle \Lambda(x), y \rangle = x \circ \langle Id, y \rangle$$

We get

$Q_{DB(z,x,y)} = A \circ E$ where

$E = \langle Id, B \rangle$

Now we lift E down to the leaves of the tree representation of A. Combining *Ass* and *DPair* allows to distribute E along an S node:

$$A \circ E = App \circ \langle \Lambda(C) \circ E, (Snd \circ Fst) \circ E \rangle$$

The leaf corresponding to the free occurrence of y in Q has already been reached; we may use *Ass*, *DPair*, *Fst* and the right identity equation

$$(IdR) \quad x \circ Id = x$$

We obtain

$$App \circ \langle \Lambda(C) \circ E, (Snd \circ Fst) \circ E \rangle = App \circ \langle \Lambda(C) \circ E, Snd \rangle$$

Now we need an equation allowing to distribute E inside $\Lambda(C)$:

$$\begin{cases}(D\Lambda)\ \ \Lambda(x)\circ y = \Lambda(x\circ <y\circ Fst, Snd>)\end{cases}$$

We get

$App\circ <\Lambda(C)\circ E, Snd>$

$= App\circ <\Lambda(App\circ <Snd\circ <E\circ Fst, Snd>, (Snd\circ Fst)\circ <E\circ Fst, Snd>>), Snd>$

After some dressing

$$\begin{aligned} App\circ <\Lambda(C)\circ E, Snd> &= App\circ <\Lambda(App\circ <Snd, Snd\circ (E\circ Fst)>), Snd> \\ &= App\circ <\Lambda(App\circ <Snd, B\circ Fst>), Snd> \end{aligned}$$

remembering $E = <Id, B>$. Now we compute $B\circ Fst$ by distributing Fst in the same way:

$$\begin{aligned} B\circ Fst &= App\circ <\Lambda(Snd\circ <Fst\circ Fst, Snd>), Snd\circ Fst\circ Fst\circ Fst> \\ &= App\circ \Lambda(Snd, Snd\circ Fst^3) = S(\Lambda(0!), 3!) \end{aligned}$$

Finally

$Q_{DB(z,x,y)} = S(\Lambda(S(0!, S(\Lambda(0!), 3!))), 0!) = Q'_{DB(z,x,y)}$

We have simulated a β-reduction by categorical rewritings. These rewritings have been able to recompute the number associated with the free occurrence of y in Q which is 1 in $Q_{DB(z,x,y)}$ and 0 in $P_{DB(z,x,y)}$ because the node λx has disappeared; they also recompose the fact that the free occurrence of z becomes 2 in $Q_{DB(z,x,y)}$ and 3 in $Q'_{DB(z,x,y)}$ because a node λz is inserted in the sequence of nodes λv up to the root.

Now we compute Q completely in the environment suggested above, using the rules with only lower case letters (*ass* rather than *Ass*, etc..). This looks very much like usual implementations of applicative languages. We start from

$S(\Lambda(A), B)\rho$ where

$\rho = (((\rho', 4), x), succ)$

We get by *ass*, *dpair* and *app*

$S(\Lambda(A), B)\rho = (\Lambda(A)\rho)(B\rho)$

We use $d\Lambda$ and set

$\rho' = (\rho, B\rho)$

We get

$(\Lambda(A)\rho)(B\rho) = A\rho'$

We manipulate A similarly and get

$A\rho' = C\rho''$ where

$\rho'' = (\rho', 1!\rho')$

Then

$C\rho'' = (0!\rho'')(1!\rho'')$

Making the leftmost reductions, and remembering the definitions of of ρ'', ρ', we get

$(0!\rho'')(1!\rho'') = (1!\rho')(1!\rho'') = (0!\rho)(1!\rho'') = succ\,(1!\rho'')$

Now we reduce the argument of $succ$.

$succ\,(1!\rho'') = succ\,(0!\rho') = succ\,(B\rho) = succ\,(0!(\rho,2!\rho)) = succ\,(2!\rho) = succ\,(\underline{4}) = \underline{5}$

Summarizing, we have introduced categorical combinators and we have suggested that their world was full of computations corresponding to those known in the λ-calculus world (β-reduction, abstract interpretation machines based on environment manipulations). Moreover all these computations are described in the unified framework of a first order rewriting system, whereas the formalisms of β-conversion and P. Landin's SECD machine [Lan] are quite different. The rest of the paper describes the typed categorical combinators formally.

2. Typed categorical combinators

First we define λc_K and CCL_K formally.

2.1. Definition

The **K-typed λc-calculus λc_K** and the **K-typed categorical combinatory logic** CCL_K are defined as follows:

K is a set of **basic types;** each term has a type, which is a term of $T_{\times,\Rightarrow}(K)$, and if M has the type σ, we write

M^σ or M: σ.

We agree that × has precedence over ⇒, and we write

$\sigma_1 \times \sigma_2 .. \times \sigma_n = (..(\sigma_1 \times \sigma_2)..\times \sigma_n)$

The structure of terms is as follows:

For λc_K:

- If x is a variable and σ is a type, then $x: \sigma$ is a term
- if $M: \sigma \Rightarrow \tau$ and $N: \sigma$, then $MN: \tau$
- if $x: \sigma$ and $M: \tau$, then $\lambda x.M: \sigma \Rightarrow \tau$
- if $M: \sigma$ and $N: \tau$, then $(M,N): \sigma \times \tau$
- if $M: \sigma \times \tau$, then $fst(M): \sigma$
- if $M: \sigma \times \tau$, then $snd(M): \tau$

For CCL_K

- If x is a variable and σ is a type, then x: σ is a term
- if A: $\sigma_2 \Rightarrow \sigma_3$ and B: $\sigma_1 \Rightarrow \sigma_2$, then $A \circ B$: $\sigma_1 \Rightarrow \sigma_3$
- Id: $\sigma \Rightarrow \sigma$
- if A: $\sigma \Rightarrow \tau_1$ and B: $\sigma \Rightarrow \tau_2$, then $<A,B>$: $\sigma \Rightarrow \tau_1 \times \tau_2$
- Fst: $\sigma \times \tau \Rightarrow \sigma$ (we shall often write $Fst^{\sigma,\tau}$)
- Snd: $\sigma \times \tau \Rightarrow \tau$ (we shall often write $Snd^{\sigma,\tau}$)
- if A: $\sigma_1 \times \sigma_2 \Rightarrow \sigma_3$, then $\Lambda(A)$: $\sigma_1 \Rightarrow (\sigma_2 \Rightarrow \sigma_3)$
- App: $(\sigma \Rightarrow \tau) \times \sigma \Rightarrow \tau$ (we shall often write $App^{\sigma,\tau}$)
- if A: $\sigma \Rightarrow \tau$ and B: σ, then AB: τ
- if A: σ and B: τ, then (A,B): $\sigma \times \tau$

Hence CCL_K is an algebra of first order terms. The theories are:

$\beta\eta SP_K$

$(beta)$	$(\lambda x^\sigma.M^\tau)N^\sigma = M[x \leftarrow N]$
(η)	$\lambda x^\sigma.M^{\sigma\Rightarrow\tau}x = M$ if $x \not\in FV(M)$
(fst)	$fst(M^\sigma,N^\tau) = M$
(snd)	$snd(M^\sigma,N^\tau) = N$
(SP)	$(fst(M^{\sigma\times\tau}),snd(M)) = M$

AA_K:

(Ass)	$(x^{\sigma_3\Rightarrow\sigma_4} \circ y^{\sigma_2\Rightarrow\sigma_3}) \circ z^{\sigma_1\Rightarrow\sigma_2} = x \circ (y \circ z)$
(IdL)	$Id^{\tau\Rightarrow\tau} \circ x^{\sigma\Rightarrow\tau} = x^{\sigma\Rightarrow\tau}$
(IdR)	$x^{\sigma\Rightarrow\tau} \circ Id^{\sigma\Rightarrow\sigma} = x$
(Fst)	$Fst^{\tau_1,\tau_2} \circ <x^{\sigma\Rightarrow\tau_1}, y^{\sigma\Rightarrow\tau_2}> = x$
(Snd)	$Snd^{\tau_1,\tau_2} \circ <x^{\sigma\Rightarrow\tau_1}, y^{\sigma\Rightarrow\tau_2}> = y$
$(DPair)$	$<x^{\sigma_1\Rightarrow\tau_1}, y^{\sigma_1\Rightarrow\tau_2}> \circ z^{\sigma\Rightarrow\sigma_1} = <x \circ y, x \circ z>$
$(Beta)$	$App^{\sigma_2,\sigma_3} \circ <\Lambda(x^{\sigma_1\times\sigma_2\Rightarrow\sigma_3}), y^{\sigma_1\Rightarrow\sigma_2}> = x \circ <Id^{\sigma_1\Rightarrow\sigma_1}, y>$
$(D\Lambda)$	$\Lambda(x^{\sigma_1\times\sigma_2\Rightarrow\sigma_3}) \circ y^{\sigma\Rightarrow\sigma_1} = \Lambda(x \circ <y \circ Fst^{\sigma,\sigma_2}, Snd^{\sigma,\sigma_2}>)$
(AI)	$\Lambda(App^{\sigma,\tau}) = Id^{(\sigma\Rightarrow\tau)\Rightarrow(\sigma\Rightarrow\tau)}$
(FSI)	$<Fst^{\sigma,\tau}, Snd^{\sigma,\tau}> = Id^{\sigma\times\tau\Rightarrow\sigma\times\tau}$
(ass)	$(x^{\sigma_1\Rightarrow\sigma_2} \circ y^{\sigma\Rightarrow\sigma_1}) z^{\sigma} = x(yz)$
(fst)	$Fst^{\sigma_1,\sigma_2}(x^{\sigma_1}, y^{\sigma_2}) = x$
(snd)	$Snd^{\sigma_1,\sigma_2}(x^{\sigma_1}, y^{\sigma_2}) = y$
$(dpair)$	$<x^{\sigma\Rightarrow\tau_1}, y^{\sigma\Rightarrow\tau_2}> z^{\sigma} = (xz, yz)$
(app)	$App^{\sigma,\tau}(x^{\sigma\Rightarrow\tau}, y^{\sigma}) = xy$
$(Quote\,1)$	$\Lambda(Fst^{\sigma,\sigma_2}) x^{\sigma} \circ y^{\sigma_1\Rightarrow\sigma_2} = \Lambda(Fst^{\sigma,\sigma_1}) x$
$(Quote\,2)$	$App^{\sigma_2,\sigma_3} \circ <x^{\sigma\Rightarrow(\sigma_2\Rightarrow\sigma_3)} \circ \Lambda(Fst^{\sigma,\sigma_1}) y^{\sigma}, z^{\sigma_1\Rightarrow\sigma_2}> = xy \circ z$

Some of these equations must be applied with caution. For instance we only can replace Id by $\Lambda(App)$, $<Fst, Snd>$ if Id is of type $(\sigma\Rightarrow\tau)\Rightarrow(\sigma\Rightarrow\tau)$, $\sigma\times\tau\Rightarrow\sigma\times\tau$ respectively.

The following lemma states some equational consequences of AA_K.

2.2. Lemma

The following equations are consequences of AA_K.

$(Quote\,3)$	$\Lambda(x^{\sigma\times\sigma_1\Rightarrow\sigma_2}) y^{\sigma} = x \circ <\Lambda(Fst^{\sigma,\sigma_1}) y, Id^{\sigma_1}>$
(id)	$Id^{\sigma\Rightarrow\sigma} x^{\sigma} = x$
$(d\Lambda)$	$\Lambda(x^{\sigma_1\times\sigma_2\Rightarrow\sigma_3}) y^{\sigma_1} z^{\sigma_2} = x(y, z)$

The system AA_K is equivalent to the system obtained by replacing $Quote\,2$ by the two following equations:

$(Quote\,2a)$	$\Lambda(Fst^{\sigma_1\Rightarrow\sigma_2,\sigma}) x^{\sigma_1\Rightarrow\sigma_2} = \Lambda(x \circ Snd^{\sigma,\sigma_1})$
$(Quote\,2b)$	$\Lambda(Fst^{\sigma_2,\sigma})(x^{\sigma_1\Rightarrow\sigma_2} y^{\sigma_1}) = x \circ \Lambda(Fst^{\sigma_1,\sigma}) y$

Proof: We only prove $Quote\,2b$ from $Quote\,2$.

$$\Lambda(Fst)(xy) =_{ass} (\Lambda(Fst) \circ x) y =_{D\Lambda,Fst} \Lambda(x \circ Fst) y$$
$$=_{Quote\,3} x \circ Fst \circ <\Lambda(Fst) y, Id> =_{Fst} x \circ \Lambda(Fst) y \quad \blacksquare$$

Now we define formally last section's De Bruijn's translation as well as the translations between λc_K and CCL_K.

2.3. Definition

Let $M: \sigma \in \lambda c_K$, and $x_0: \sigma_0, \ldots, x_n: \sigma_n$ be s.t. $FV(M) \subseteq \{x_0, \ldots, x_n\}$. We define $M_{DB_K(x_0, \ldots, x_n)}$ as follows:

$$
\boxed{
\begin{array}{l}
x^{\sigma}_{DB_K(x_0^{\sigma_0}, \ldots, x_n^{\sigma_n})} = Snd^{\sigma\times\sigma_n \cdots \times\sigma_{i+1}, \sigma_i} \circ Fst^{\sigma\times\sigma_n \cdots \times\sigma_i, \sigma_{i-1}} \circ \cdots \circ Fst^{\sigma\times\sigma_n \cdots \times\sigma_1, \sigma_0} \\
\text{where } i \text{ is minimum s.t. } x = x_i \\
(\lambda x.M)^{\sigma\Rightarrow\tau}_{DB_K(x_0, \ldots, x_n)} = \Lambda(M_{DB_K(x, x_0, \ldots, x_n)}) \\
(M^{\sigma\Rightarrow\tau} N^{\sigma})_{DB_K} = App^{\sigma,\tau} \circ \langle M_{DB_K}, N_{DB_K} \rangle \\
(M, N)^{\sigma\times\tau}_{DB_K} = \langle M_{DB_K}, N_{DB_K} \rangle \\
fst\,(M^{\sigma\times\tau})_{DB_K} = Fst^{\sigma,\tau} \circ M_{DB_K} \\
snd\,(M^{\sigma\times\tau})_{DB_K} = Snd^{\sigma,\tau} \circ M_{DB_K}
\end{array}
}
$$

One has

$$M^{\tau}_{DB_K(x_0^{\sigma_0}, \ldots, x_n^{\sigma_n})} : \sigma\times\sigma_n \cdots \times\sigma_0 \Rightarrow \tau$$

($\sigma_0, \ldots, \sigma_n, \tau$ are determined by $M, x_0, \ldots, x_n$ while σ is any type).

We define

$$M_{CCL_K} = M^{\tau}_{DB_K(x_0^{\sigma_0}, \ldots, x_n^{\sigma_n})}(y^{\sigma}, x_n^{\sigma_n}, \ldots, x_0^{\sigma_0})$$

where y is different from all x_i and has the type σ in M_{DB_K} (we apply the De Bruijn's translation to the environment formally).

We define the translation in the reverse direction by

$$
\boxed{
\begin{array}{l}
x^{\sigma}_{\lambda c_K} = x^{\sigma} \\
Id^{\sigma\Rightarrow\sigma}_{\lambda c_K} = \lambda x^{\sigma}.x \\
Fst^{\sigma,\tau}_{\lambda c_K} = \lambda x^{\sigma\times\tau}.fst\,(x) \\
Snd^{\sigma,\tau}_{\lambda c_K} = \lambda x^{\sigma\times\tau}.snd\,(x) \\
App^{\sigma,\tau}_{\lambda c_K} = \lambda x^{(\sigma\Rightarrow\tau)\times\sigma}.fst\,(x) snd\,(x) \\
(A^{\sigma_2\Rightarrow\sigma_3} \circ B^{\sigma_1\Rightarrow\sigma_2})_{\lambda c_K} = \lambda x^{\sigma_1}.A_{\lambda c_K}(B_{\lambda c_K} x) \\
(A^{\sigma\Rightarrow\tau} B^{\sigma})_{\lambda c_K} = A_{\lambda c_K} B_{\lambda c_K} \\
\langle A^{\sigma\Rightarrow\tau_1}, B^{\sigma\Rightarrow\tau_2} \rangle_{\lambda c_K} = \lambda x^{\sigma}.(A_{\lambda c_K} x, B_{\lambda c_K} x) \\
(A^{\sigma}, B^{\tau})_{\lambda c_K} = (A_{\lambda c_K}, B_{\lambda c_K}) \\
\Lambda(A^{\sigma_1\times\sigma_2\Rightarrow\sigma_3})_{\lambda c_K} = \lambda x^{\sigma_1} y^{\sigma_2}.A_{\lambda c_K}(x, y)
\end{array}
}
$$

Clearly

$M^{\tau}_{CCL_K}: \tau$ and $A^{\tau}_{\lambda c_K}: \tau$

We suppose that x,y do not appear in A,B $\bullet$

In [CuCCL,CuTh] the untyped version of these calculi and translations is defined. The main difference is that in the untyped case the application and couple operators are defined and not primitive. There we proved a First equivalence theorem of which the Second theorem below is just a typed copy (we refer to [CuCCL] for a sketchy proof and to [CuTh] for a full proof).

2.4. Second equivalence theorem

For any terms $M,N \in \lambda c_K$, $A,B \in \mathbf{CCL}_K$, the following holds:

(1) $M_{CCL_K \lambda c_K} =_{\beta\eta SP_K} M$

(2) $A_{\lambda c_K CCL_K} =_{AA_K} A$

(3) $A =_{AA_K} B \;\Rightarrow\; A_{\lambda c_K} =_{\beta\eta SP_K} B_{\lambda c_K}$

(4) $M =_{\beta\eta SP_K} N \;\Rightarrow\; M_{CCL_K} =_{AA_K} N_{CCL_K}$ $\blacksquare$

Actually neither η nor SP are needed in (1) (see [CuTh]).

Now we introduce $\mathbf{CCC}_K$.

2.5. Definition

Let K be a set of **basic objects.** The types are now couples written $\sigma \rightarrow \tau$ of terms σ,τ of $T_{\times,\Rightarrow}(K \cup \{\varepsilon\})$ where ε, called terminal object, is different from all the elements of K. The elements of $T_{\times,\Rightarrow}(K \cup \{\varepsilon\})$ are the **objects.**

The **free cartesian closed category** $\mathbf{CCC}_K$ is defined as follows:

- if x is a variable and σ,τ are objects, then $x: \sigma \rightarrow \tau$ is a term
- if $f: \sigma_2 \rightarrow \sigma_3$ and $g: \sigma_1 \rightarrow \sigma_2$ are terms, then $f \circ g: \sigma_1 \rightarrow \sigma_3$ is a term
- $Id: \sigma \rightarrow \sigma$ is a term
- if $f: \sigma \rightarrow \tau_1$ and $g: \sigma \rightarrow \tau_2$ are terms, then $\langle f, g\rangle: \sigma \rightarrow \tau_1 \times \tau_2$ is a term
- $Fst: \sigma \times \tau \rightarrow \sigma$ is a term
- $Snd: \sigma \times \tau \rightarrow \tau$ is a term
- $1: \sigma \rightarrow \varepsilon$ is a term
- if $f: \sigma_1 \times \sigma_2 \rightarrow \sigma_3$ is a term, then $\Lambda(f): \sigma_1 \rightarrow (\sigma_2 \Rightarrow \sigma_3)$ is a term
- $App: (\sigma \Rightarrow \tau) \times \sigma \rightarrow \tau$ is a term

We use as above the notation $Fst^{\sigma,\tau}$, $Snd^{\sigma,\tau}$ and $App^{\sigma,\tau}$, and we also write Id^{σ} for $Id: \sigma \rightarrow \sigma$ and 1^{σ} for $1: \sigma \rightarrow \varepsilon$.

CCC_K is the set of equations $CCL\beta\eta SP + Ter$ where

$(Ter)\ 1^{\sigma \rightarrow \varepsilon} = x^{\sigma \rightarrow \varepsilon}$ $\bullet$

$CCL\beta\eta SP$ consists of the equations from Ass until FSI included in AA_K above (in the types some $\Rightarrow$ have to be replaced by $\rightarrow$) (more on $CCL\beta\eta SP$ in [CuCCL]).

Here typing is critical since *Ter* without types would reduce to: "everything equals 0". The difference to the definition 2.1 is the absence of application and couple operators, and the presence of a family of constants 1, the unique arrows to the terminal object.

Now we establish the equivalence of CCL_K, AA_K and CCC_K, CCC_K. First we have to connect the types of both theories. We shall use the well-known isomorphism between $A \rightarrow B$ and $1 \rightarrow (A \Rightarrow B)$ in a cartesian closed category (A,B are any objects, 1 is the terminal object), which is as follows in our setting:

$$\left[\begin{array}{l}(x^{\sigma \rightarrow \tau})^+ = \Lambda(x \circ Snd^{\varepsilon,\sigma}) \\ (x^{\varepsilon \rightarrow \sigma \Rightarrow \tau})^- = App^{\sigma,\tau} \circ <x \circ 1^{\sigma \rightarrow \varepsilon}, Id^\sigma>\end{array}\right.$$

One proves easily the following equations:

$$\left[((x^{\sigma \rightarrow \tau})^+)^- =_{CCC_K} x \text{ and } ((x^{\varepsilon \rightarrow \sigma \Rightarrow \tau})^-)^+ =_{CCC_K} x\right.$$

2.6. Definition

With every object σ we associate

$$\left\{\sigma^* \in T_{\times,\Rightarrow}(K) \cup \{\varepsilon\} \ , \ \sigma^-: \sigma \rightarrow \sigma^* \in CCC_K \ , \ \sigma^+: \sigma^* \rightarrow \sigma \in CCC_K\right.$$

defined as follows:

- $\sigma^* = \sigma$, $\sigma^+ = \sigma^- = Id^\sigma$ if $\sigma \in K \cup \{\varepsilon\}$

For the product we proceed by cases:

- $\sigma_1^*, \sigma_2^* \neq \varepsilon$:
  $(\sigma_1 \times \sigma_2)^* = \sigma_1^* \times \sigma_2^*$, $(\sigma_1 \times \sigma_2)^+ = <\sigma_1^+ \circ Fst, \sigma_2^+ \circ Snd>$, $(\sigma_1 \times \sigma_2)^- = <\sigma_1^- \circ Fst, \sigma_2^- \circ Snd>$
- $\sigma_1^* \neq \varepsilon$, $\sigma_2^* = \varepsilon$
  $(\sigma_1 \times \sigma_2)^* = \sigma_1^*$, $(\sigma_1 \times \sigma_2)^+ = <Id, \sigma_2^+ \circ 1^{\sigma_1}> \circ \sigma_1^+$, $(\sigma_1 \times \sigma_2)^- = \sigma_1^- \circ Fst$
- $\sigma_1^* = \varepsilon$, $\sigma_2^* \neq \varepsilon$: symmetric
- $\sigma_1^*, \sigma_2^* = \varepsilon$
  $(\sigma_1 \times \sigma_2)^* = \varepsilon$, $(\sigma_1 \times \sigma_2)^+ = <\sigma_1^+, \sigma_2^+>$, $(\sigma_1 \times \sigma_2)^- = 1$

Now the exponentiation:

- $\sigma_1^*, \sigma_2^* \neq \varepsilon$
  $(\sigma_1 \Rightarrow \sigma_2)^* = \sigma_1^* \Rightarrow \sigma_2^*$
  $(\sigma_1 \Rightarrow \sigma_2)^+ = \Lambda(\sigma_2^+ \circ App \circ <Fst, \sigma_1^- \circ Snd>)$, $(\sigma_1 \Rightarrow \sigma_2)^- = \Lambda(\sigma_2^- \circ App \circ <Fst, \sigma_1^+ \circ Snd>)$
- $\sigma_1^* = \varepsilon$, $\sigma_2^* \neq \varepsilon$
  $(\sigma_1 \Rightarrow \sigma_2)^* = \sigma_2^*$

$(\sigma_1\Rightarrow\sigma_2)^+ = \Lambda(Fst)\circ\sigma_2^+$, $(\sigma_1\Rightarrow\sigma_2)^- = \sigma_2^-\circ App\circ <Id,\sigma_1^+\circ 1^{\sigma_1\Rightarrow\sigma_2}>$

- $\sigma_2^*=\varepsilon$

$(\sigma_1\Rightarrow\sigma_2)^* = \varepsilon$, $(\sigma_1\Rightarrow\sigma_2)^+ = \Lambda(\sigma_2^+\circ 1^{\varepsilon\times\sigma_1})$, $(\sigma_1\Rightarrow\sigma_2)^- = 1$ ∎

We omitted many types, and shall do so in the sequel. σ^* can be viewed as a canonical representent for σ when identifying $\sigma\times\varepsilon$, $\varepsilon\times\sigma$, $\varepsilon\Rightarrow\sigma$ with σ, and $\sigma\Rightarrow\varepsilon$ with ε. This is justified by the following lemma:

2.7. Lemma

For any $\sigma \in T_{\times,\Rightarrow}(K\cup\{\varepsilon\})$ the following holds:

$\left[\sigma^+\circ\sigma^- =_{CCC_K} Id^\sigma \text{ and } \sigma^-\circ\sigma^+ =_{CCC_K} Id^{\sigma^*}\right.$

Proof: We only check one case.

$$\begin{aligned} <Id,\sigma_2^+\circ 1^{\sigma_1}>\circ\sigma_1^+\circ\sigma_1^-\circ Fst^{\sigma_1\times\sigma_2} &=_{rec,Ter} <Fst,\sigma_2^+\circ 1^{\sigma_1\times\sigma_2}> \\ &=_{Ter} <Fst,\sigma_2^+\circ 1^{\sigma_2}\circ Snd^{\sigma_1\times\sigma_2}> \\ &=_{rec} <Fst,Snd> = Id \end{aligned}$$ ∎

Now we define the translations between CCL_K and CCC_K

2.8. Definition

With any term A: σ of CCL_K we associate a term A_{CCC_K}: $\varepsilon\rightarrow\sigma$ of CCC_K defined as follows:

$x^\sigma_{CCC_K} = x^{\varepsilon\rightarrow\sigma}$
$A_{CCC_K} = A^+$, if $A = Id, Fst, Snd, App$
$(A\circ B)_{CCC_K} = (A^-_{CCC_K}\circ B^-_{CCC_K})^+$
$<A,B>_{CCC_K} = <A^-_{CCC_K}, B^-_{CCC_K}>^+$
$\Lambda(A)_{CCC_K} = \Lambda(A^-_{CCC_K})^+$
$(AB)_{CCC_K} = A^-_{CCC_K}\circ B_{CCC_K}$
$(A,B)_{CCC_K} = <A_{CCC_K}, B_{CCC_K}>$

Conversely with any term $f:\sigma\rightarrow\tau$ of CCC_K s.t. $(\sigma\Rightarrow\tau)^*\neq\varepsilon$ (i.e. $\tau^*\neq\varepsilon$), we associate a term f_{CCL_K}: $(\sigma\Rightarrow\tau)^*$ of CCL_K defined by

$x^{\sigma\to\tau}_{CCL_K} = x^{(\sigma\Rightarrow\tau)^*}$

$Id^{\sigma}_{CCL_K} = Id^{\sigma^*\Rightarrow\sigma^*}$

$Fst^{\sigma,\tau}_{CCL_K} = Fst^{\sigma^*,\tau^*}$ if $\tau^*\neq\varepsilon$ $\quad = Id^{\sigma^*\Rightarrow\sigma^*}$ if $\tau^*=\varepsilon$

Symmetrically for *Snd*

$App^{\sigma,\tau}_{CCL_K} = App^{\sigma^*,\tau^*}$ if $\sigma^*\neq\varepsilon$ $\quad = Id^{\tau^*\Rightarrow\tau^*}$ if $\sigma^*=\varepsilon$

$(f^{\sigma_2\to\sigma_3}\circ g^{\sigma_1\to\sigma_2})_{CCL_K} = f_{CCL_K}\circ g_{CCL_K}$ if $\sigma_1^*,\sigma_2^*\neq\varepsilon$

$= f_{CCL_K} g_{CCL_K}$ if $\sigma_1^*=\varepsilon$, $\sigma_2^*\neq\varepsilon$

$= \Lambda(Fst^{\sigma_3^*,\sigma_1^*}) f_{CCL_K}$ if $\sigma_1^*\neq\varepsilon$, $\sigma_2^*=\varepsilon$

$= f_{CCL_K}$ if $\sigma_1^*,\sigma_2^*=\varepsilon$

$\langle f^{\sigma\to\tau_1}, g^{\sigma\to\tau_2}\rangle_{CCL_K} = \langle f_{CCL_K}, g_{CCL_K}\rangle$ if $\sigma^*,\tau_1^*,\tau_2^*\neq\varepsilon$

$= (f_{CCL_K}, g_{CCL_K})$ if $\sigma^*=\varepsilon$, $\tau_1^*,\tau_2^*\neq\varepsilon$

$= f_{CCL_K}$ if $\tau_1^*\neq\varepsilon$, $\tau_2^*=\varepsilon$

$= g_{CCL_K}$ if $\tau_1^*=\varepsilon$, $\tau_2^*\neq\varepsilon$

$\Lambda(f^{\sigma_1\times\sigma_2\to\sigma_3})_{CCL_K} = \Lambda(f_{CCL_K})$ if $\sigma_1^*,\sigma_2^*\neq\varepsilon$

$= f_{CCL_K}$ if $\sigma_1^*=\varepsilon$ or $\sigma_2^*=\varepsilon$.

Now we may state the Third equivalence theorem.

2.9. Third equivalence theorem

For all terms A,B of CCL_K and f,g of CCC_K of appropriate types, the following holds:

(1) $A =_{AA_K} B \;\Rightarrow\; A_{CCC_K} =_{CCC_K} B_{CCC_K}$

(2) $f^{\sigma\to\tau} =_{CCC_K} g^{\sigma\to\tau} \;\Rightarrow\; f_{CCL_K} =_{AA_K} g_{CCL_K}$ if $\tau^*\neq\varepsilon$

(3) $A_{CCC_K,CCL_K} =_{CCC_K} A$

(4) $f^{\sigma\to\tau}_{CCL_K,CCC_K} =_{AA_K} \overline{\sigma\to\tau}(f\,[x_0\leftarrow\underline{\sigma\Rightarrow\tau}(x_0),\ldots,x_n\leftarrow\underline{\sigma\Rightarrow\tau}(x_n)])$

where $V(A) = \{x_0,\ldots,x_n\}$ and $\overline{\sigma\to\tau}$, $\underline{\sigma\Rightarrow\tau}$ are defined by

$\overline{\sigma\to\tau}(f^{\sigma\to\tau}) = (\tau^-\circ f\circ\sigma^+)^+$ if $\sigma^*\neq\varepsilon$ $\quad = \tau^-\circ f\circ\sigma^+$ if $\sigma^*=\varepsilon$

$\underline{\sigma\Rightarrow\tau}\,(g^{\varepsilon\to(\sigma\Rightarrow\tau)^*}) = \tau^+\circ f^-\circ\sigma^-$ if $\sigma^*\neq\varepsilon$

$= \tau^+\circ f\circ\sigma^-$ if $\sigma^*=\varepsilon$

Proof: Tedious but easy. We only check (4) for *App*.

$App^{\sigma,\tau}_{CCL_K,CCC_K} = A$

$(\sigma^*\neq\varepsilon)$ $\quad (App^{\sigma^*,\tau^*})^+$

We have to check

$$\left[\tau^{-} \circ App^{\sigma,\tau} \circ ((\sigma \Rightarrow \tau) \times \sigma)^{+} = App^{\sigma^{*},\tau^{*}}\right.$$

Let

$$(\sigma \Rightarrow \tau)^{+} = \Lambda(B)$$

$$\begin{aligned}\tau^{-} \circ App^{\sigma,\tau} \circ ((\sigma \Rightarrow \tau) \times \sigma)^{+} &= \tau^{-} \circ App \circ \langle \Lambda(B) \circ Fst, \sigma^{+} \circ Snd \rangle \\ &= \tau^{-} \circ \tau^{+} \circ App \circ \langle Fst, \sigma^{-} \circ Snd \rangle \circ \langle Fst, \sigma^{+} \circ Snd \rangle \\ &= App \circ \langle Fst, \sigma^{-} \circ \sigma^{+} \circ Snd \rangle = App\end{aligned}$$

- $(\sigma^{*} = \varepsilon)\quad A = (Id^{\tau^{*}})^{+}$

We have to check

$$\left[\tau^{-} \circ App^{\sigma,\tau} \circ ((\sigma \Rightarrow \tau) \times \sigma)^{+} = Id^{\tau^{*}}\right.$$

$$\begin{aligned}\tau^{-} \circ App^{\sigma,\tau} \circ ((\sigma \Rightarrow \tau) \times \sigma)^{+} &= \tau^{-} \circ App \circ \langle Id, \sigma^{+} \circ 1 \rangle \circ \Lambda(Fst) \circ \tau^{+} \\ &= \tau^{-} \circ App \circ \langle \Lambda(Fst), .. \rangle \circ \tau^{+} \\ &= \tau^{-} \circ Fst \circ \langle Id, .. \rangle \circ \tau^{+} = Id^{\tau^{*}} \quad \blacksquare\end{aligned}$$

We end the section by pointing out that the two equivalence theorems of the section may be used to decide the equational equality in CCC_K (and also in CCL_K). Indeed the rewriting system obtained by orienting the rules of $\beta\eta SP_K$ from left to right is confluent (cf. [Pot]) and noetherian. We refer to [LamSco] for a proof of that property, which was actually established by J. Lambek and P.J. Scott for the same purpose. For concluding on decidability, we just have to remark

$$\left[f^{\sigma \rightarrow \tau} =_{CCC_K} g^{\sigma \rightarrow \tau} \text{ iff } f_{CCL_K} =_{AA_K} g_{CCL_K} \text{ iff } f_{CCL_K \lambda c_K} =_{\beta\eta SP_K} g_{CCL_K \lambda c_K}\right.$$

using

$$\overline{\sigma \rightarrow \tau}(\underline{\sigma \Rightarrow \tau}(x)) =_{CCC_K} x \quad \text{et} \quad \underline{\sigma \Rightarrow \tau}(\overline{\sigma \rightarrow \tau}(x)) =_{CCC_K} x$$

3. Conclusion

We have exhibited the connection between λ-calculus and cartesian closed categories, which goes back to [Lam,Sco] and quite independently to [BeSy,CuTh3], in a very syntactical and computational fashion. We refer to [CuTh,CuEq] for the semantic equivalences induced by the theorems in this paper.

It is very tempting to implement evaluators of categorical combinators. A result in [CuTh,CuTh] states that the evaluator last informally described in section 1, working by leftmost-outermost reductions, is complete with respect to the models of the underlying theory (namely CCL_K enriched with arithmetic combinators). Moreover the author devised a categorical abstract machine transforming categorical combinators into actual machine instructions. This machine will be described in a forthcoming paper with G. Cousineau, who significantly improved the original proposal.

Related (and independent) work appears in [PaGho,Poi,Dyb,LamSco]. [Poi],[LamSco] explicitely state an equivalence in the kind of this paper between (quite) λc_K and CCC_K, in a syntactic, a more semantic setting respectively. The differences of the present paper to these references are mainly the introduction of CCL_K, and the connection with De Bruijn's ideas, both contributing to an operational setting.

4. References

[BeSy] G. Berry, Some Syntactic and Categorical Constructions of Lambda-calculus models, Rapport INRIA 80 (1981).

[Bru] N.G. De Bruijn, Lambda-calculus Notation without Nameless Dummies, a Tool for Automatic Formula Manipulation, Indag Math. 34, 381-392 (1972).

[CuTh3] P-L. Curien, Algorithmes Séquentiels sur Structures de Données Concrètes, Thèse de Troisième Cycle, Université Paris VII (Mars 1979).

[CuTh] P-L. Curien, Combinateurs Catégoriques, Algorithmes Séquentiels et Programmation Applicative, Thèse d'Etat, Université Paris VII (Décembre 83), to be published in english as a monograph.

[CuCCL] P-L. Curien, Categorical Combinatory Logic, submitted to ICALP 85.

[CuEq] Syntactic Equivalences Inducing Semantic Equivalences, submitted to EUROCAL 1985.

[Dyb] P. Dybjer, Category-Theoretic Logics and Algebras of Programs, PhD Thesis, Chalmers University of Technology, Goteborg (1983).

[Lam] J. Lambek, From Lambda-calculus to Cartesian Closed Categories, in To H.B. Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism, ed. J.P. Seldin and J.R. Hindley, Academic Press (1980).

[LamSco] J. Lambek and P.J. Scott, Introduction to Higher Order Categorical Logic, to be published by Cambridge University Press (1984).

[Lan] P.J. Landin, The Mechanical Evaluation of Expressions, Computer Journal 6, 308-320 (1964).

[PaGhoTh] K. Parsaye-Ghomi, Higher Order Abstract Algebras, PhD Thesis, UCLA (1981).

[Poi] A. Poigné, Higher Order Data Structures, Cartesian Closure Versus λ-calculus, STACS 84, Lect. Notes in Comput. Sci.

[Pot] G. Pottinger, The Church-Rosser Theorem for the Typed λ-calculus with Extensional Pairing, preprint, Carnegie-Mellon University, Pittsburgh (March 1979).

[Sco4] D. Scott, Relating Theories of the Lambda-calculus, cf. [Lam].