INFORMATION AND CONTROL 69, 188–254 (1986)

# Categorical Combinators

P.-L. CURIEN

*CNRS-Université Paris VII, LITP, Tour 55-56 1er étage,*
*3 Place Jussieu, 75221 Paris Cedex 05, France*

Our main aim is to present the connection between $\lambda$-calculus and Cartesian closed categories both in an untyped and purely syntactic setting. More specifically we establish a syntactic equivalence theorem between what we call categorical combinatory logic and $\lambda$-calculus with explicit products and projections, with $\beta$ and $\eta$-rules as well as with surjective pairing. "Combinatory logic" is of course inspired by Curry's combinatory logic, based on the well-known *S, K, I*. Our combinatory logic is "categorical" because its combinators and rules are obtained by extracting untyped information from Cartesian closed categories (looking at arrows only, thus forgetting about objects). Compiling $\lambda$-calculus into these combinators happens to be natural and provokes only $n \log n$ code expansion. Moreover categorical combinatory logic is entirely faithful to $\beta$-reduction where combinatory logic needs additional rather complex and unnatural axioms to be. The connection easily extends to the corresponding typed calculi, where typed categorical combinatory logic is a free Cartesian closed category where the notion of terminal object is replaced by the explicit manipulation of applying (a function to its argument) and coupling (arguments to build datas in products). Our syntactic equivalences induce equivalences at the model level. The paper is intended as a mathematical foundation for developing implementations of functional programming languages based on a "categorical abstract machine," as developed in a companion paper (Cousineau, Curien, and Mauny, *in* "Proceedings, ACM Conf. on Functional Programming Languages and Computer Architecture," Nancy, 1985). 

## 1. INTRODUCTION

The motivation for studying calculi of combinators is that implementing the $\beta$-rule of the $\lambda$-calculus, and more generally parameter passing mechanisms of programming languages involves some difficulties with the scope of variables, so that getting rid of variables at compile time may yield both efficient and safe interprets. Among the numerous approaches to eliminate variables, two of them are of particular interest:

— the use of Curry's combinators *S, K, I*, as suggested by Turner (1979);

— the so-called De Bruijn's (1972) notation (which is implicit in many closure based implementations of functional languages) for $\lambda$-expressions,



0019-9958/86 $3.00

replacing the names of the bound variables by their binding height, i.e., the number of $\lambda$'s between the variable and its binding $\lambda$ in the expression; actually, as we shall see, compilling $\lambda$-expressions into categorical code factorizes through getting its translation in De Bruijn's notation, and more specifically what remains to be done is just textual transformation.

Introducing categorical combinators starting from the $\lambda$-calculus is best done by using an intuition on types, and trying to describe the meaning of typed $\lambda$-expressions then quite naturally leads to categorical combinators. We refer to Curien (1985a). Here we shall use a much steeper way, more aking to a machine description. We take the risk of trying to convince the reader rather by the magic of "pushing symbols" than by a thorough semantic motivation. We suppose some acquaintance with $\lambda$-calculus, and equational theories (we refer to Barendregt, 1984; Huet and Oppen, 1980). Knowing the definition of a Cartesian closed category is not needed, but obviously would help intuition. Section 2 is devoted to the definition of untyped $\lambda c$-calculus (i.e., $\lambda$-calculus with explicit couples) and untyped categorical combinatory logic and the proof of their equivalence. For the sake of comparison Section 3 recalls the similar results of Curry's combinatory logic, which we call here classical. There is nothing original in this section, possibly except that we handle at a syntactic level notions which were introduced at the model level by various people (Meyer, 1982; Scott, 1976; Koymans, 1982). Sections 4 and 5 are short incursions into the typed case and models. Section 6 is a discussion.

The relative length of Section 2 enhances its importance. The material of Sections 2 and 4 has been presented at ICALP 85, CAAP 85 (Curien, 1985b, c), respectively. The paper is essentially extracted from the author's thesis, an improved version of which appeared as a monograph (Curien, 1985a), to which we refer for more motivation and more detail (especially on related work).

The rest of the section is a rapid introduction to categorical combinators viewed as machine instructions. In the style of Landin's (1964) SECD machine, imagine the computation of $\lambda$-expressions as some compound action of a code *applied* to an environment. Imagine moreover that the environment has a binary tree representation. Then the following can be said on the three constructors of the pure $\lambda$-calculus:

— variable: $x$ represents the access to the $x$ part of the environment: hence the action may be viewed as the *composition* of some actions of going or *projecting* from a node to its *first* or *second* son.

— application: the action of $MN$ may be conceptually decomposed as follows: first combine, or put aside, or *pair* the actions of $M$ and $N$, then perform the *application*.

— abstraction: $\lambda x.M$ cannot act directly on the environment in this kind of evaluation: the involved mechanism was called closure by Landin. Entering into it is out of the scope of the present paper, but is central to our categorical abstract machine (Cousineau, Curien, and Mauny, 1985). What we want to suggest may be phrased as follows: $\lambda x.M$ will only give rise to an action if a context $(\lambda x.M)\,N$ is reached. Then $M$ will act on a modified environment, a combination, or *couple* of the current environment and the result of the computation of $N$. So $M$ has two arguments, whereas $\lambda x.M$ has only one: the environment; the point is that the second argument has been abstracted, i.e., that some *currying* is involved.

We have introduced all the categorical combinators but one, the *identity* which we shall see does not arise when compiling $\lambda$-expressions into categorical combinatory logic, but when simulating a $\beta$-reduction on the categorical code.

Now our categorical kit is complete and the play can begin. Summarizing we have composing, identity, pairing, first and second projections, currying, and application. So far for the code itself representing $\lambda$-expressions as compound actions. The interface with the environment enlarges our kit to applying and coupling. In the untyped setting, applying and coupling are not primitive and may be defined. In the typed setting they can either be coded as degenerated cases of composition and pairing (with arrows starting from the terminal object), or added explicitly in a typed version of categorical combinatory logic. We insist on the difference between applying and application, between coupling and pairing. For the first think of the addition given as a binary operator compared to the addition given as a constant (as in ML) which can be handled as such, without reference to its arguments ($(\lambda x.x)+$), for the second notice that in naive set theoretic terms, coupling two functions $f: D \mapsto E$ and $g: D \mapsto F$ yields an element $(f, g)$ in the product $(D \Rightarrow E) \times (D \Rightarrow F)$ which is not a function type, while pairing them yields a function $\langle f, g \rangle$ in $D \Rightarrow (E \times F)$.

## 2. $\lambda$-Calculus and Pure Categorical Combinatory Logic

First we introduce the $\lambda c$-calculus, i.e., the $\lambda$-calculus with explicit couples and projections and corresponding conversions, forming with the well known $\beta$ and $\eta$-conversions a theory called $\beta\eta SP$. Then we introduce categorical combinatory logic as a first-order signature equipped with an equational theory $CCL\beta\eta SP$, called strong categorical combinatory logic.

The syntactic equivalence theorem establishes the equivalence of these two formalisms. It has two nice corollaries: first we show that $CCL\beta\eta SP$ is equivalent to a weaker theory $CCL$, called weak categorical combinatory

logic (which leads to the categorical abstract machine in (Cousineau, Curien, and Mauny, 1985)), plus an extensionality axiom; the other corollary is a functional (or combinatory) completeness result. Weakening an equation of $CCL\beta\eta SP$ yields an equivalence with the $\lambda c$-calculus without the surjective pairing (formal definition below). We end the discussion of pure categorical combinatory logic by a (poor) list of properties of the term rewriting systems associated with categorical equations. *We stress that the calculi in this section are typefree* (types appear in Sect. 4).

*$\lambda$-Calculus and De Bruijn's Notation*

We recall the formal definition of $\lambda$-calculus, define the De Bruijn's notation formally, and describe the $\beta$-conversion in this notation. We extend the $\lambda$-calculus into the $\lambda\mathbf{c}$-calculus, which has explicit products and projections and associated additional conversion rules.

2.1. DEFINITION. The **pure $\lambda$-calculus** $\lambda$ is built from a set *Var* of **variables**, which are the basic $\lambda$-expressions. The other $\lambda$-expressions (or terms) are built by

— **application**: if $M$, $N$ are terms, then $MN$ is a term

— **abstraction**: if $x$ is a variable and $M$ is a term, then $\lambda x.M$ is a term.

We use the following notation:

$$M_1 M_2 \ldots M_n = (\ldots.(M_1 M_2)\ldots M_{n-1})\, M_n$$

$$\lambda x_1 x_2 \ldots x_n.M = \lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)\ldots)).$$

Here is a formal definition of the sets $FV(M)$, $BV(M)$ of **free** and **bound** variables, defined by induction on $M$:

$$FV(x) = \{x\}, \qquad BV(x) = \{\ \}$$

$$FV(MN) = FV(M) \cup FV(N), \qquad BV(MN) = BV(M) \cup BV(N)$$

$$FV(\lambda x.M) = FV(M)/\{x\}, \qquad BV(\lambda x.M) = BV(M) \cup \{x\}.$$

Clearly $FV(M) \cup BV(M) = V(M)$, where $V(M)$ is the set of variables occurring in $M$. $FV(M) \cap BV(M)$ may be nonempty (see below).

One may add a set *Cons* of constants: then the basic $\lambda$-expressions are the constants and the variables, and the others are built by application and abstraction as above. For every constant $C$ we define

$$FV(C) = \{\ \}$$

and the calculus is denoted by $\lambda(Cons)$.

Now we introduce the De Bruin's notation formally.

2.2. DEFINITION. The **$\lambda$-calculus in De Bruijn's notation** is built as follows:

— any natural number is a term

— if $M$, $N$ are terms, then $MN$ is a term

— if $M$ is a term, $\lambda.M$ is a term.

De Bruijn's notation should be viewed as the *abstract* notation for $\lambda$-calculus, since it drops irrelevant details of variable names. We define the translation of a $\lambda$-expression into the De Bruijn's notation. The translation is relative to a list of variables, which may be considered as a formal environment.

2.3. DEFINITION. For any $M \in \lambda$ s.t. $FV(M) \subseteq \{x_0,..., x_n\}$ we define its **De Bruijn's translation** $M_{\mathrm{DB}(x_0,...,x_n)}$ by

$$x_{\mathrm{DB}(x_0,...,x_n)} = i \qquad \text{if } i \text{ is minimum s.t. } x = x_i$$

$$(\lambda x.M)_{\mathrm{DB}(x_0,...,x_n)} = \lambda.M_{\mathrm{DB}(x,x_0,...,x_n)}$$

$$(MN)_{\mathrm{DB}(x_0,...,x_n)} = M_{\mathrm{DB}(x_0,...,x_n)} N_{\mathrm{DB}(x_0,...,x_n)}.$$

When two terms $M$, $N$ are such that

$$M_{\mathrm{DB}(x_0,...,x_n)} = N_{\mathrm{DB}(x_0,...,x_n)}$$

for suitable $x_0,..., x_n$ (clearly the property does not depend on the particular choice of $x_0,..., x_n$), then we say that $M$, $N$ are **$\alpha$-equivalent**.

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

$$M_{\mathrm{DB}(y,x,z)} = (\lambda.(\lambda.01)\ 1)((\lambda.0)\ 2).$$

*The $\lambda$-expressions, and the operations on them* (particularly the substitution below) *are always defined modulo $\alpha$-equivalence.*

We leave the following key property as an exercise:

EXERCISE. Show that for any finite set of variables $X$ and any $\lambda$-expression $M$ one may find $N$ s.t. $BV(N) \cap X = \varnothing$ and $M, N$ are $\alpha$-equivalent.

This simple fact allows us to define the substitution, on which computations by $\beta$-conversions are based, without being involved in too much detail about renaming variables.

2.4. DEFINITION. The expression $M[x \leftarrow N]$, called "$M$ in which $N$ is **substituted** for all free occurrences of $x$," is defined as follows, by induction on $M$,

— $x[x \leftarrow N] = N$, $y[x \leftarrow N] = y$ (if $y \neq x$)

— $(M_1 M_2)[x \leftarrow N] = (M_1[x \leftarrow N])(M_2[x \leftarrow N])$

— $(\lambda y.M)[x \leftarrow N] = \lambda y.(M[x \leftarrow N])$ if $y \neq x$ and if $y \notin FV(N)$ (not a restriction by the exercise above).

EXERCISE. Check $M[x \leftarrow N] = M$ if $x \notin FV(M)$.

Now we may define the two conversions (reductions if they are orientated from left to right) of the $\lambda$-calculus. First we recall the classical notion of occurrence.

2.5. DEFINITION. The **occurrence** $u$ (where $u$ is a word on $\{0, 1, 2\}$) of a term $M$ is defined as follows:

$$M/\varepsilon = M$$

$$M/0u = N/u \qquad \text{if } M = \lambda x.N$$

$$M/1u = M_1/u, \qquad M/2u = M_2/u \qquad \text{if } M = M_1 M_2.$$

We denote by $M[u \leftarrow N]$ the term obtained by replacing the occurrence $u$ by $N$ in $M$.

Of course occurrences of expressions in De Bruijn's notation are defined in the same way.

2.6. DEFINITION. The $\beta$, $\eta$-reductions of the $\lambda$-calculus are defined by

($\beta$): $(\lambda x.M)\, N = M[x \leftarrow N]$

($\eta$): $\lambda x.Mx = M$ if $x \notin FV(M)$.

We denote by $\beta\eta$ the theory $\beta + \eta$. As in equational theories the rules of the definition extend to any context of a left or right member, i.e., if, for instance, $M/u = (\lambda x.M)\, N$ (we say that $M/u$ is a **redex**), then

$$M =_\beta M[u \leftarrow M[x \leftarrow N]].$$

The next definition describes the $\beta$-reduction in De Bruijn's notation.

2.7. DEFINITION. The following **substitution** and **lifting** operators for terms in De Bruijn's notation are defined as follows:

$$n[m \leftarrow N] = \begin{cases} n & \text{if } n < m \\ U_0^n(N) & \text{if } n = m \\ n-1 & \text{if } n > m \end{cases}$$

$$M_1 M_2[m \leftarrow N] = M_1[m \leftarrow N]\, M_2[m \leftarrow N]$$

$$(\lambda.M)[m \leftarrow N] = \lambda.(M[m+1 \leftarrow N])$$

$$U_i^m(j) = \begin{cases} j & \text{if } j < i \\ j+m & \text{if } j \geqslant i \end{cases}$$

$$U_i^m(N_1 N_2) = U_i^m(N_1)\, U_i^m(N_2)$$

$$U_i^m(\lambda.N) = \lambda.(U_{i+1}^m(N)).$$

This definition should be viewed as a careful treatment of how substitution can actually be performed mechanically.

EXAMPLE (taking $M$ as above).

$$\begin{aligned} M_{\mathrm{DB}(y,x,z)} &= (\lambda.(\lambda.01)\ 1)((\lambda.0)\ 2) \\ &= ((\lambda.01)\ 1)[0 \leftarrow (\lambda.0)\ 2] \\ &= ((\lambda.01)[0 \leftarrow (\lambda.0)\ 2])(1[0 \leftarrow (\lambda.0)\ 2]) \\ &= ((\lambda.01)[0 \leftarrow (\lambda.0)\ 2])\ 0 \\ &= (\lambda.((01)[0 \leftarrow (\lambda.0)\ 2]))\ 0 \\ &= (\lambda.(0[1 \leftarrow (\lambda.0)\ 2])(1[1 \leftarrow (\lambda.0)\ 2]))\ 0 \\ &= (\lambda.(0U_0^1((\lambda.0)\ 2)))\ 0 \\ &= (\lambda.(0(U_0^1(\lambda.0)\ U_0^1(2))))\ 0 \\ &= (\lambda.(0(U_0^1(\lambda.0)\ 3)))\ 0 \\ &= (\lambda.(0((\lambda.U_1^1(0))\ 3)))\ 0 \\ &= (\lambda.(0((\lambda.0)\ 3)))\ 0. \end{aligned}$$

That this machinery indeed performs $\beta$-reductions is left as an

EXERCISE. Show that for suitable $x_0, \ldots, x_n$, if $((\lambda x.M)\ N)_{\mathrm{DB}(x_0,\ldots,x_n)} = (\lambda.P)\ Q$, then

$$(M[x \leftarrow N])_{\mathrm{DB}(x_0,\ldots,x_n)} = P[0 \leftarrow Q].$$

Now we define the $\lambda$-calculus with explicit couples and projections.

2.8. DEFINITION. The **pure $\lambda c$-calculus $\lambda\mathbf{c}$** is defined by adding the following operators to the structure defined in 2.1:

— **couple**: if $M, N$ are terms then $(M, N)$ is a term

— **first projection**: if $M$ is a term then $\mathrm{fst}(M)$ is a term

— **second projection**: if $M$ is a term then $\mathrm{snd}(M)$ is a term.

We use the following notation:

$$(M_1, M_2,..., M_n) = (...(M_1, M_2),..., M_n).$$

In addition to the $\beta$ and $\eta$-conversions (Definition 2.6 extends easily to $\lambda\mathbf{c}$), we define the three following conversions:

$$\begin{aligned} &(\mathrm{fst}) && \mathrm{fst}((M, N)) = M \\ &(\mathrm{snd}) && \mathrm{snd}((M, N)) = N \\ &(SP) && (\mathrm{fst}(M), \mathrm{snd}(M)) = M \end{aligned}$$

(in the two first equalities the double parentheses stress the arities of fst and ( ); they will be often omitted however for simplicity).

We write $\beta P = \beta + \mathrm{fst} + \mathrm{snd}$, $\beta\eta P = \beta P + \eta$, and $\beta\eta SP = \beta\eta P + SP$.

The $\lambda c$-calculus may be coded inside the $\lambda$-calculus, where the conversions fst and snd, but not $SP$, may be simulated (proof omitted):

2.9. DEFINITION. With every term $M$ of $\lambda\mathbf{c}$ we associate a term $M_\lambda \in \lambda$ defined as follows:

$$\begin{aligned} x_\lambda &= x \\ (MN)_\lambda &= M_\lambda N_\lambda \\ (\lambda x.M)_\lambda &= \lambda x.M_\lambda \\ (M, N)_\lambda &= \lambda x.xM_\lambda N_\lambda, \qquad \text{where } x \notin FV(M) \cup FV(N), \\ \mathrm{fst}(M)_\lambda &= M_\lambda(\lambda xy.x) \\ \mathrm{snd}(M)_\lambda &= M_\lambda(\lambda xy.y). \end{aligned}$$

2.10. PROPOSITION. *For all $M, N \in \lambda\mathbf{c}$ the following holds*

$$M =_{\beta\eta P} N \Rightarrow M_\lambda =_{\beta\eta} N_\lambda.$$

The proposition remains valid without $\eta$ (the conversions fst and snd are simulated using $\beta$ only), but *cannot* be extended to handle *SP*, as was proved by Barendregt (1974). This justifies considering $\lambda$-calculus on its own.

*Strong Categorical Combinatory Logic*

We introduce the equational presentation of categorical combinatory logic. We introduce the applying and coupling operators, together with a set of rules involving them, called weak rules, as derived operators and rules.

2.11. DEFINITION. The **pure categorical combinatory logic CCL** is the algebra of terms built fro a set *Var* of variables over the following signature:

— Id, Fst, Snd, App, called respectively **identity**, **first projection**, **second projection**, and **application**, of arity 0

— $\Lambda$, called **currying**, of arity 1

— $\circ$, $\langle\ ,\ \rangle$, called respectively **composition** and **pairing**, of arity 2.

We use the following notation:

$$\langle A_1,..., A_n\rangle = \langle ...\langle A_1, A_2\rangle,..., A_n\rangle.$$

Now we state the equations which will allow to establish the correspondence between **CCL** and $\lambda$**c**.

2.12. DEFINITION. *CCL*$\beta\eta$*SP*, also called **strong categorical combinatory logic**, is the following set of equations:

| | |
|---|---|
| (Ass) | $(x \circ y) \circ z = x \circ (y \circ z)$ |
| (IdL) | $\mathrm{Id} \circ x = x$ |
| (IdR) | $x \circ \mathrm{Id} = x$ |
| (Fst) | $\mathrm{Fst} \circ \langle x, y\rangle = x$ |
| (Snd) | $\mathrm{Snd} \circ \langle x, y\rangle = y$ |
| (DPair) | $\langle x, y\rangle \circ z = \langle x \circ z, y \circ z\rangle$ |
| (Beta) | $\mathrm{App} \circ \langle \Lambda(x), y\rangle = x \circ \langle \mathrm{Id}, y\rangle$ |
| (D$\Lambda$) | $\Lambda(x) \circ y = \Lambda(x \circ \langle y \circ \mathrm{Fst}, \mathrm{Snd}\rangle)$ |
| (AI) | $\Lambda(\mathrm{App}) = \mathrm{Id}$ |
| (FSI) | $\langle \mathrm{Fst}, \mathrm{Snd}\rangle = \mathrm{Id}$ |

$CCL\beta\eta P$ is the system obtained by replacing the axiom FSI in $CLL\beta\eta SP$ by

$$\text{(FSA)} \quad \text{App} \circ \langle \text{Fst}, \text{Snd} \rangle = \text{App}$$

Finally we write $CCL\beta = CCL\beta\eta SP - \text{AI} - \text{FSI}$.

We strongly recommend the following exercise, which will give the reader some insight into these equations.

EXERCISE. Prove the equivalence between $CCL\beta\eta SP$ and the system

$$\text{Ass} + \text{IdL} + \text{IdR} + \text{Fst} + \text{Snd} + S\text{Pair} + \text{App} + S\Lambda$$

where

$$\text{(}S\text{Pair)} \quad \langle \text{Fst} \circ x, \text{Snd} \circ x \rangle = x$$

$$\text{(App)} \quad \text{App} \circ \langle \Lambda(x) \circ \text{Fst}, \text{Snd} \rangle = x$$

$$\text{(}S\Lambda\text{)} \quad \Lambda(\text{App} \circ \langle x \circ \text{Fst}, \text{Snd} \rangle) = x$$

which is the set of equations defining the Cartesian closed categories (see, e.g., Lambek, 1972) except that there are no types and no equation for the terminal object (see Sect. 4).

EXERCISE. Show that the following is a consequence of $CCL\beta$:

$$\text{(Beta')} \quad \text{App} \circ \langle \Lambda(x) \circ y, z \rangle = x \circ \langle y, z \rangle.$$

We show that we can define the applying and coupling operators. Intuitions trace back to the bijective correspondence between $1 \to D \Rightarrow E$ and $D \to E$, for all objects $D, E$ in a Cartesian closed category with terminal object 1 (see Sect. 4).

2.13. DEFINITION. We define the operations "$^>$", "$^<$" of arity 1, "$\cdot$" (**applying**, denoted by simple juxtaposition) and ( ) (**coupling**) of arity 2 as follows (for any $A, B$):

$$A^> = \Lambda(A \circ \text{Snd})$$

$$A^< = \text{App} \circ \langle A, \text{Id} \rangle$$

$$A \cdot B = (A \circ B^>)^< \text{ (in practice we shall write } AB)$$

$$(A, B) = \langle A^>, B^> \rangle^<.$$

We use the same notation as for the $\lambda c$-calculus operators of the same names; moreover we agree that applying has stronger precedence than composition.

Finally we denote by **RA** (right absorbing) the set of terms $B$ of **CCL** s.t.

$$B =_{CCL\beta\eta SP} B \circ \Lambda(\mathrm{Snd}).$$

Think of $^>$ and $^<$ as associating an arrow of $1 \to D$ with an element of $D$ and vice versa. Notice that applying has arity 2 while application has arity 0: it is somehow the same as having a binary addition, but also an addition constant in itself, which can be manipulated in other contexts than just applied to its arguments, as in the example of Section 1. Finally notice that, for any $A$ in **CCL**, $A^>$ is in **RA**.

2.14. LEMMA. *For all terms* $A \in$ **CCL**, $B \in$ **RA** the following holds

$$(A^>)^< =_{CCL\beta\eta SP} A \tag{1}$$

$$B =_{CCL\beta\eta SP} B \circ A \tag{2}$$

$$(B^<)^> =_{CCL\beta\eta SP} B. \tag{3}$$

*Moreover* $CCL\beta\eta SP \vdash CCL + \mathrm{Quote}$, *where CCL, also called* **weak categorical combinatory logic**, *is the following set of equations*:

(id) $\mathrm{Id}.x = x$

(ass) $(x \circ y).z = x.(y.z)$

(fst) $\mathrm{Fst}.(x, y) = x$

(snd) $\mathrm{Snd}.(x, y) = y$

(dpair) $\langle x, y \rangle.z = (x.z, y.z)$

(app) $\mathrm{App}.(x, y) = x.y$

(d$\Lambda$) $(\Lambda(x).y).z = x.(y, z)$

*and* Quote *is the following set of equations*:

(Quote$_1$) $(\Lambda(\mathrm{Fst}).x) \circ y = \Lambda(\mathrm{Fst}).x$

(Quote$_2$) $\mathrm{App} \circ \langle x \circ (\Lambda(\mathrm{Fst}).y), z \rangle = (x.y) \circ z$

(Quote$_3$) $\Lambda(x).y = x \circ \langle \Lambda(\mathrm{Fst}).y, \mathrm{Id} \rangle$.

*Proof.*

$$(A^{>})^{<} = \mathrm{App} \circ \langle \Lambda(A \circ \mathrm{Snd}), \mathrm{Id} \rangle =_{\mathrm{Beta}} (A \circ \mathrm{Snd}) \circ \langle \mathrm{Id}, \mathrm{Id} \rangle =_{\mathrm{Ass,Snd,IdR}} A$$

$$B \circ A =_{\mathrm{def}} (B \circ \Lambda(\mathrm{Snd})) \circ A =_{\mathrm{Ass},D\Lambda,\mathrm{Snd}} B \circ \Lambda(\mathrm{Snd}) =_{\mathrm{def}} B$$

$$(B^{<})^{>} = \Lambda((\mathrm{App} \circ \langle B, \mathrm{Id} \rangle) \circ \mathrm{Snd}) =_{\mathrm{Ass,DPair,IdL}} \Lambda(\mathrm{App} \circ \langle B \circ \mathrm{Snd}, \mathrm{Snd} \rangle)$$

$$=_{2,2} \Lambda(\mathrm{App} \circ \langle B \circ \mathrm{Fst}, \mathrm{Snd} \rangle) =_{S\Lambda} B.$$

For the other equations we first show

$$CCL\beta\eta SP,\ \mathrm{Quote}_2 \vdash \mathrm{Quote}_3$$

$$CCL\beta\eta SP,\ \mathrm{Quote}_3,\ \mathrm{ass},\ \mathrm{fst},\ \mathrm{snd},\ \mathrm{dpair},\ \mathrm{app} \vdash \mathrm{id},\ d\Lambda.$$

Here is for $\mathrm{Quote}_3$:

$$\Lambda(x)\ y =_{\mathrm{idR}} \Lambda(x)\ y \circ \mathrm{Id} =_{\mathrm{Quote}_2} \mathrm{App} \circ \langle \Lambda(x) \circ \Lambda(\mathrm{Fst})\ y, \mathrm{Id} \rangle$$

$$=_{\mathrm{Beta}} x \circ \langle \Lambda(\mathrm{Fst})\ y, \mathrm{Id} \rangle.$$

For id, we first get by $\mathrm{Quote}_3$,

$$\Lambda(\mathrm{Snd})\ x =_{\mathrm{Quote}_3} \mathrm{Snd} \circ \langle \ldots, \mathrm{Id} \rangle =_{\mathrm{Snd}} \mathrm{Id}. \tag{4}$$

Hence

$$\mathrm{Id}\, x =_{4,\mathrm{snd}} (\Lambda(\mathrm{Snd})(y,x))(\mathrm{Snd}(y,x)) =_{\mathrm{app,dpair,ass}} (\mathrm{App} \circ \langle \Lambda(\mathrm{Snd}), \mathrm{Snd} \rangle)(y, x)$$

$$=_{\mathrm{Beta,Snd,snd}} x.$$

For $d\Lambda$ we first show

$$\Lambda(\mathrm{Fst})\ y = \Lambda(\mathrm{Snd} \circ \mathrm{Fst})(z, y) \tag{5}$$

$$\Lambda(\mathrm{Fst})\ y =_{\mathrm{snd,ass}} (\Lambda(\mathrm{Fst}) \circ \mathrm{Snd})(z, y) =_{D\Lambda} \Lambda(\mathrm{Fst} \circ \langle \mathrm{Snd} \circ \mathrm{Fst}, \mathrm{Snd} \rangle)(z, y)$$

$$=_{\mathrm{Fst}} \Lambda(\mathrm{Snd} \circ \mathrm{Fst})(z, y).$$

By (5) we obtain the following instance of $\mathrm{d}\Lambda$:

$$\Lambda(\mathrm{Fst})\ yz =_{5,\mathrm{fst}} (\Lambda(\mathrm{Snd} \circ \mathrm{Fst})(z, y))(\mathrm{Fst}(z, y))$$

$$=_{\mathrm{app,dpair,ass}} (\mathrm{App} \circ \langle \Lambda(\mathrm{Snd} \circ \mathrm{Fst}), \mathrm{Fst} \rangle)(z, y) =_{\mathrm{Beta,Ass,Fst,IdR,snd}} y. \tag{6}$$

Finally we may prove $d\Lambda$ using $\text{Quote}_3$,

$$\Lambda(x)\ yz =_{\text{Quote}_3} (x \circ \langle \Lambda(\text{Fst})\ y, \text{Id} \rangle)\ z =_{\text{ass,dpair,6,id}} x(y, z).$$

We are left with the other equations of *CCL* and Quote:

$$x(yz) =_{\text{def}} (x \circ ((y \circ z^{>})^{<})^{>})^{<} =_3 (x \circ (y \circ z^{>}))^{<} =_{\text{Ass,def}} (x \circ y)\ z.$$

(3) may be applied since we verify $y \circ z^{>} \in \mathbf{RA}$ (same argument as to prove (2)):

$$\text{Fst}(x, y) = (\text{Fst} \circ (\langle x^{>}, y^{>} \rangle^{<})^{>})^{<} =_3 (\text{Fst} \circ \langle x^{>}, y^{>} \rangle)^{<} =_{\text{Fst},1} x$$

(likewise, $\langle x^{>}, y^{>} \rangle \in \mathbf{RA}$),

$$(xz, yz) =_{\text{def}} \langle ((x \circ z^{>})^{<})^{>}, ((y \circ z^{>})^{<})^{>} \rangle^{<} =_3 \langle x \circ z^{>}, y \circ z^{>} \rangle^{<}$$
$$=_{\text{DPair,def}} \langle x, y \rangle\ z$$
$$\text{App}(x, y) =_{\text{def}} (\text{App} \circ (\langle x^{>}, y^{>} \rangle^{<})^{>})^{<} =_{3,\text{def}} (\text{App} \circ \langle \Lambda(x \circ \text{Snd}), y^{>} \rangle)^{<}$$
$$=_{\text{Beta,Snd,def}} xy.$$

For the two remaining equations of Quote we first establish

$$\Lambda(\text{Fst})\ x = x^{>} \tag{7}$$
$$\Lambda(\text{Fst})\ x =_{\text{def}} \text{App} \circ \langle \Lambda(\text{Fst}) \circ x^{>}, \text{Id} \rangle =_{\text{Beta}',\text{Fst}} x \rangle$$
$$\Lambda(\text{Fst})\ x \circ y =_7 x^{>} \circ y =_2 x^{>} =_7 \Lambda(\text{Fst})\ x$$

since $x^{>} \in \mathbf{RA}$,

$$\text{App} \circ \langle x \circ \Lambda(\text{Fst})\ y, z \rangle =_7 \text{App} \circ \langle x \circ y^{>}, z \rangle$$
$$=_{2,\text{Ass,IdL,DPair}} \text{App} \circ (\langle x \circ y^{>}, \text{Id} \rangle \circ z)$$
$$=_{\text{Ass,def}} xy \circ z.$$

EXERCISE. Show that the two last equations in 2.13 are consequences of $CCL\beta\eta SP + CCL +$ Quote. In other words, there is only one way of defining applying and coupling in such a way that $CCL\beta\eta SP + CCL +$ Quote holds.

*The Equivalence Theorem*

We rephrase the translation $_{\text{DB}}$ in the categorical setting, show how to code the substitution and lifting in terms of categorical combinators, thus detaching a branch of the equivalence theorem: strong categorical rules allow us to simulate $\beta\eta SP$-conversions. Then we define the translations between $\lambda$c-calculus and categorical combinatory logic, and establish the full equivalence theorem.

2.15. DEFINITION. Let $M \in \lambda c$ and $x_0,..., x_n$ be s.t. $FV(M) \subseteq \{x_0,..., x_n\}$. $M_{\mathrm{DB}(x_0,...,x_n)}$ is defined by

$$x_{\mathrm{DB}(x_0,...,x_n)} = i! \qquad \text{if } i \text{ is minimum s.t. } x = x_i, \text{ where } i! = \mathrm{Snd} \circ \mathrm{Fst}^i$$

$$(\lambda x.M)_{\mathrm{DB}(x_0,...,x_n)} = \Lambda(M_{\mathrm{DB}(x,x_0,...,x_n)})$$

$$(MN)_{\mathrm{DB}(x_0,...,x_n)} = \mathrm{App} \circ \langle M_{\mathrm{DB}(x_0,...,x_n)}, N_{\mathrm{DB}(x_0,...,x_n)} \rangle$$

$$(M, N)_{\mathrm{DB}(x_0,...,x_n)} = \langle M_{\mathrm{DB}(x_0,...,x_n)}, N_{\mathrm{DB}(x_0,...,x_n)} \rangle$$

$$\mathrm{fst}(M)_{\mathrm{DB}(x_0,...,x_n)} = \mathrm{Fst} \circ M_{\mathrm{DB}(x_0,...,x_n)}$$

$$\mathrm{snd}(M)_{\mathrm{DB}(x_0,...,x_n)} = \mathrm{Snd} \circ M_{\mathrm{DB}(x_0,...,x_n)}.$$

EXERCISE. Show that the following properties hold:

$$M_{\mathrm{DB}(x_0,...,x_n)} = N_{\mathrm{DB}(x_0,...,x_{i-1},y,x_{i+1},...,x_n)} \qquad \text{where } N = M[x_i \leftarrow y]$$

$$M_{\mathrm{DB}(x,x_0,...,x_n)} = M_{\mathrm{DB}(x_0,...,x_n)} \circ \mathrm{Fst} \qquad \text{if } x \notin FV(M)$$

$$M[x_0 \leftarrow N^0,..., x_n \leftarrow N^n]_{\mathrm{DB}(y_0,...,y_m)} = M_{\mathrm{DB}(x_0,...,x_n)} \circ \langle z \circ \mathrm{Fst}^{m+1}, N^n_{\mathrm{DB}(y_0,...,y_n)},..., N^0_{\mathrm{DB}(y_0,...,y_m)} \rangle$$

where we suppose $FV(N^0) \cup ... \cup FV(N^n) \subseteq \{y_0,..., y_m\}$ and where $z$ is any variable (hint: for the second property one may proceed as in the proof of 2.17 below, whereas an induction on terms would involve tedious permutations).

The last property in the exercise is very significant: *it means that the categorical composition mirrors the $\lambda$-calculus substitution.* More technically the presence of $z$ in the equation corresponds to the intuition that the environment is made of its useful part, and "the rest," i.e., $z$.

Now we come to the simulation of substitution and lifting. We first show a lemma which will turn out to be the key of the simulation.

2.16. LEMMA. *Let* $A \in \mathbf{CCL}$. *We write* $P(A) = \langle A \circ \mathrm{Fst}, \mathrm{Snd} \rangle$. *The following holds*:

$$\mathrm{Fst}^m \circ P^n(A) =_{CCL\beta\eta SP} P^{n-m}(A) \circ \mathrm{Fst}^m \qquad \text{if } 1 \leqslant m \leqslant n$$

*Proof.* By induction on $m$,

$$\begin{aligned}
\mathrm{Fst}^{m+1} \circ P^n(A) &=_{\mathrm{def,Ass}} \mathrm{Fst} \circ (\mathrm{Fst}^m \circ P^n(A)) =_{\mathrm{ind}} \mathrm{Fst} \circ (P^{n-m}(A) \circ \mathrm{Fst}^m) \\
&=_{\mathrm{Ass}} (\mathrm{Fst} \circ P^{n-m}(A)) \circ \mathrm{Fst}^m \\
&=_{\mathrm{def}} (\mathrm{Fst} \circ (\langle P^{n-m-1}(A) \circ \mathrm{Fst}, \mathrm{Snd} \rangle) \circ \mathrm{Fst}^m \\
&=_{\mathrm{Fst}} (P^{n-m-1}(A) \circ \mathrm{Fst}) \circ \mathrm{Fst}^m =_{\mathrm{Ass}} P^{n-m-1}(A) \circ \mathrm{Fst}^{m+1}.
\end{aligned}$$

2.17. PROPOSITION. *Let* $M, N \in \lambda\mathbf{c}$ *and* $x_0,..., x_n$ *be* s.t. $FV(M) \cup FV(N) \subseteq \{x_0,..., x_n\}$. *The following holds*:

$$M =_{\beta\eta SP} N \Rightarrow M_{\mathrm{DB}(x_0,...,x_n)} =_{CCL\beta\eta SP} N_{\mathrm{DB}(x_0,...,x_n)}.$$

*Proof.* We only have to prove the properties for a single conversion step. For $\beta$ we only need to code properly the substitution and lifting. Let, for any $A, B$

$$A[m \leftarrow B] = A \circ P^m(\langle \mathrm{Id}, B\rangle) \qquad \text{and} \qquad U_i^m(B) = B \circ P^i(\mathrm{Fst}^m).$$

We only have to *prove* the equations in Definition 2.7. The application, abstraction cases are proved using Ass and DPair, and D$\Lambda$, respectively. We are left with the cases of variables:

— $n > m$: $n![m \leftarrow B] =_{\mathrm{def,Ass}} (\mathrm{Snd} \circ \mathrm{Fst}^{n-m}) \circ (\mathrm{Fst}^m \circ P^m(\langle \mathrm{Id}, B\rangle))$
$=_{2.16} (\mathrm{Snd} \circ \mathrm{Fst}^{n-m}) \circ (\langle \mathrm{Id}, B\rangle \circ \mathrm{Fst}^m)$
$=_{\mathrm{Ass,Fst}} \mathrm{Snd} \circ (\mathrm{Fst}^{n-m-1} \circ \mathrm{Fst}^m) =_{\mathrm{def}} n-1!$

— $n = m$: $n![m \leftarrow B] =_{2.16} \mathrm{Snd} \circ (\langle \mathrm{Id}, B\rangle \circ \mathrm{Fst}^m)$
$=_{\mathrm{Ass,Snd}} B \circ \mathrm{Fst}^m = U_0^m(B)$

— $n < m$: this case is the same as the last case below:
$n![m \leftarrow B] =_{CCL\beta\eta SP} n!$

And now the lifting:

— $j \geqslant i$: $U_i^m(j!) =_{\mathrm{Ass}} (\mathrm{Snd} \circ \mathrm{Fst}^{j-i}) \circ (\mathrm{Fst}^i \circ P^i(\mathrm{Fst}^m))$
$=_{2.16} (\mathrm{Snd} \circ \mathrm{Fst}^{j-i}) \circ \mathrm{Fst}^{m+i} =_{\mathrm{def}} m + j!$

— $j < i$: $U_i^m(j!) =_{2.16} \mathrm{Snd} \circ (P^{i-j}(\mathrm{Fst}^m) \circ \mathrm{Fst}^j)$
$=_{\mathrm{def}} \mathrm{Snd} \circ (\langle P^{i-j-1}(\mathrm{Fst}^m) \circ \mathrm{Fst}, \mathrm{Snd}\rangle \circ \mathrm{Fst}^j) =_{\mathrm{Ass,Snd}} j!$

Now we prove the simulation of the $\eta$-reduction. We suppose that $N/w = \lambda x.M_1 x$, where $x \notin FV(M_1)$. Let $M_{\mathrm{DB}(x_0,...,x_n)}/w = A$. Then

$$N_{\mathrm{DB}(x_0,...,x_n)} =_{CCL\beta\eta SP} M_{\mathrm{DB}(x_0,...,x_n)}[w \leftarrow B]$$

where

$$B = \Lambda(\mathrm{App} \circ \langle U_0^{+1}(A), 0!\rangle).$$

We just have to notice

$$A =_{\mathrm{IdR,AI,D}\Lambda} \Lambda(\mathrm{App} \circ \langle A \circ \mathrm{Fst}, \mathrm{Snd}\rangle).$$

Finally for the conversions fst, snd, and *SP*, we remark that the translation transforms them into conversions Fst, Snd, and SPair.

A more cautious treatment of the proof above (using DPair in particular) allows to render the proposition more precise. Let us consider $CCL\beta$ as a rewriting system by orientating the equations from left to right:

a $\beta$-reduction is simulated by a derivation (i.e., a sequence of elementary reduction steps) of $CCL\beta$, i.e.,

$$M \to_\beta N \Rightarrow M_{\mathrm{DB}(x_0,\ldots,x_n)} \to_{CCL\beta} N_{\mathrm{DB}(x_0,\ldots,x_n)}.$$

For $\eta$ the system obtained by adding the rule

$$x \to \Lambda(\mathrm{App} \circ \langle x \circ \mathrm{Fst}, \mathrm{Snd} \rangle)$$

(obtained by reversing S$\Lambda$) to $CCL\beta$ simulates the reversed $\eta$-reductions, i.e.,

$$M \to \lambda x.Mx \qquad (x \notin FV(M)).$$

The following exercises describe an equivalent way (not quite, see Sect. 4) of defining the De Bruijn's translation (actually it was first discovered that way by the author and Berry (1981)).

EXERCISE. Let $(\mathrm{Subst}_n)_{n \geqslant 0}$ be the sequence of terms defined by

$$\mathrm{Subst}_0 = \langle \mathrm{Fst} \circ \mathrm{Fst}, \mathrm{Snd} \rangle$$

$$\mathrm{Subst}_i = \langle \mathrm{Fst}^{i+2}, \mathrm{Snd}, \mathrm{Snd} \circ \mathrm{Fst}^i, \mathrm{Snd} \circ \mathrm{Fst}^{i-1}, \ldots, \mathrm{Snd} \circ \mathrm{Fst} \rangle.$$

Show that for all $j, k$,

$$\mathrm{Subst}_j(x, x_n, \ldots, x_0, z) =_{CCL} (x, x_n, \ldots, x_{j+1}, z, x_{j-1}, \ldots, x_0)$$

$$k! \circ \mathrm{Subst}_j =_{CCL\beta} \mathrm{Snd} \qquad \text{if } j = k,$$

$$k! \circ \mathrm{Subst}_j =_{CCL\beta} k+1! \qquad \text{if } j \neq k.$$

EXERCISE. Let $x_0, \ldots, x_n$ be a sequence of variables. For all terms $M \in \lambda$ s.t. $V(M) \subseteq \{x_0, \ldots, x_n\}$, define the *formal semantics*, denoted by $[\![M]\!]_{x_0,\ldots,x_n}$ of $M$ by

$$[\![x_j]\!]_{x_0,\ldots,x_n} = j!$$

$$[\![MN]\!]_{x_0,\ldots,x_n} = \mathrm{App} \circ \langle [\![M]\!]_{x_0,\ldots,x_n}, [\![N]\!]_{x_0,\ldots,x_n} \rangle$$

$$[\![\lambda x_j.M]\!]_{x_0,\ldots,x_n} = \Lambda([\![M]\!]_{x_0,\ldots,x_n} \circ \mathrm{Subst}_j)$$

Show that

$$[\![M]\!]_{x_0,\ldots,x_n} =_{CCL\beta} M_{\mathrm{DB}(x_0,\ldots,x_n)}.$$

We come to the main result of this section: the equivalence between the $\lambda c$-calculus and the categorical combinatory logic. We define the translations.

2.18. DEFINITION. With every term $M$ of $\lambda\mathbf{c}$ s.t. $FV(M) \subseteq \{x_0,..., x_n\}$ we associate a term $M_{\mathrm{CCL}}$ of **CCL** defined by

$$M_{\mathrm{CCL}} = M_{\mathrm{DB}(x_0,...,x_n)}(x, x_n,..., x_0)$$

($x$ is distinct from $x_0,..., x_n$). With every term $A$ of CCL we associate a term $A_{\lambda c}$ of $\lambda\mathbf{c}$ defined as follows:

$$\begin{aligned}
x_{\lambda c} &= x \\
\mathrm{Id}_{\lambda c} &= \lambda x.x \\
\mathrm{Fst}_{\lambda c} &= \lambda x.\mathrm{fst}(x) \\
\mathrm{Snd}_{\lambda c} &= \lambda x.\mathrm{snd}(x) \\
\mathrm{App}_{\lambda c} &= \lambda x.\mathrm{fst}(x)\,\mathrm{snd}(x) \\
(A \circ B)_{\lambda c} &= \lambda x.A_{\lambda c}(B_{\lambda c}x) \\
\langle A, B\rangle_{\lambda c} &= \lambda x.(A_{\lambda c}x, B_{\lambda c}x) \\
\Lambda(A)_{\lambda c} &= \lambda xy.A_{\lambda c}(x, y)
\end{aligned}$$

(with variables $x, y$ not belonging to $V(A) = FV(A_{\lambda c})$, $V(B) = FV(B_{\lambda c})$). We write $M_{\mathrm{CCL},\lambda c} = (M_{\mathrm{CCL}})_{\lambda c}$, and likewise for other compositions.

EXERCISE. Verify that $M_{\mathrm{CCL}}$ does not depend on the choice of the sequence $x_0,..., x_n$ modulo $CCL\beta$ (proceed as in 2.17). Notice that $M_{\mathrm{CCL},\lambda c} = M_{\mathrm{DB},\lambda c}(x, x_n,..., x_0)$ (we replace $\mathrm{DB}(x_0,..., x_n)$ by DB when no confusion may arise).

The following lemma shows that the translation $_{\lambda c}$ has the expected behaviour w.r.t. the applying and coupling operators.

2.19. LEMMA. *For all terms $A, B$ of* **CCL** *the following holds (with $u \notin FV(A_{\lambda c})$):*

$$\begin{aligned}
(A^{>})_{\lambda c} &= \lambda u.A_{\lambda c} \\
(A^{<})_{\lambda c} &= \lambda u.(A_{\lambda c}u)\,u \\
(AB)_{\lambda c} &= A_{\lambda c}B_{\lambda c} \\
(A, B)_{\lambda c} &= (A_{\lambda c}, B_{\lambda c}).
\end{aligned}$$

*Proof.*

$$(A^{>})_{\lambda c} =_{\text{def},\beta} \lambda uv.A_{\lambda c}(\text{Snd}(u, v)) =_{\text{snd},\eta} \lambda u.A_{\lambda c}$$

$$(A^{<})_{\lambda c} =_{\text{def},\beta} \lambda u.(\lambda v.\text{fst}(v)\ \text{snd}(v))(A_{\lambda c}u, u) =_{\beta,\text{fst,snd}} \lambda u.(A_{\lambda c}u)\ u$$

$$(AB)_{\lambda c} =_{\text{def}} ((A \circ B^{>})^{<})_{\lambda c} = \lambda u.((\lambda v.A_{\lambda c}((\lambda w.B_{\lambda c})\ v))\ u)\ u$$

$$=_{\beta} \lambda u.(A_{\lambda c}B_{\lambda c})\ u =_{\eta} A_{\lambda c}B_{\lambda c}$$

$$(A, B)_{\lambda c} = (\langle A^{>}, B^{>}\rangle^{<})_{\lambda c} = \lambda u.((\lambda v.((\lambda w.A_{\lambda c})\ v, (\lambda w.B_{\lambda c})\ v))\ u)\ u$$

$$=_{\beta} \lambda u.(A_{\lambda c}, B_{\lambda c})\ u =_{\eta} (A_{\lambda c}, B_{\lambda c}).$$

2.20. SYNTACTIC EQUIVALENCE THEOREM. *For all terms* $M, N \in \lambda\mathbf{c}$, $A, B \in \mathbf{CCL}$ *the following holds*:

(1) $M =_{\beta\eta SP} N \Rightarrow M_{\text{CCL}} =_{CCL\beta\eta SP} N_{\text{CCL}}$

(2) $A =_{CCL\beta\eta SP} B \Rightarrow A_{\lambda c} =_{\beta\eta SP} B_{\lambda c}$

(3) $M_{\text{CCL},\lambda c} =_{\beta\eta P} M$

(4) $A_{\lambda c,\text{CCL}} =_{CCL\beta\eta P} A.$

*Proof.* (1) is a consequence of 2.17. We prove (2) rule by rule.

$$((x \circ y) \circ z)_{\lambda c} =_{\text{def},\beta} \lambda u.x(y(zu)) =_{\text{def},\beta} (x \circ (y \circ z))_{\lambda c}$$

$$(\text{Id} \circ x)_{\lambda c} =_{\text{def}} \lambda u.(\lambda y.y)(xu) =_{\beta,\eta} x$$

$$(x \circ \text{Id})_{\lambda c} =_{\text{def}} \lambda u.x((\lambda y.y)\ u) =_{\beta,\eta} x$$

$$(\text{Fst} \circ \langle x, y\rangle)_{\lambda c} =_{\text{def},\beta} \lambda u.(\lambda v.\text{fst}(v))(xu, yu) =_{\beta,\text{fst},\eta} x$$

$$(\langle x, y\rangle \circ z)_{\lambda c} =_{\text{def},\beta} \lambda u.(\lambda v.(xv, yv))(zu)$$

$$=_{\beta} \lambda u.((\lambda v.x(zv))\ u, (\lambda v.y(zv))\ u) =_{\beta,\text{def}} \langle x \circ z, y \circ z\rangle_{\lambda c}$$

$$(\text{App} \circ \langle \Lambda(x), y\rangle)_{\lambda c} =_{\text{def},\beta} \lambda u.(\lambda v.\text{fst}(v)\ \text{snd}(v))(\lambda w.x(u, w), yu)$$

$$=_{\beta,\text{fst,snd}} \lambda u.(\lambda w.x(u, w))(yu)$$

$$=_{\beta} \lambda u.x(u, yu) =_{\beta,\text{def}} (x \circ \langle \text{Id}, y\rangle)_{\lambda c}$$

$$(\Lambda(x) \circ y)_{\lambda c} =_{\text{def},\beta} \lambda uw.x(yu, w) =_{\text{fst,snd}} \lambda uw.x(y(\text{fst}(u, w)), \text{snd}(u, w))$$

$$=_{\beta,\text{def}} \Lambda(x \circ \langle y \circ \text{Fst}, \text{Snd}\rangle)_{\lambda c}$$

$$\langle \text{Fst}, \text{Snd}\rangle_{\lambda c} =_{\text{def},\beta} \lambda u.(\text{fst}(u), \text{snd}(u)) =_{SP} \text{Id}_{\lambda c}$$

$$(\Lambda(\text{App}))_{\lambda c} =_{\text{def},\beta} \lambda uv.uv =_{\eta} \lambda u.u =_{\text{def}} \text{Id}_{\lambda c}.$$

Now we prove (3):

$$x_{\mathrm{CCL},\lambda c} =_{\mathrm{def}} \mathrm{Snd}(y, x)_{\lambda c} =_{(2),\mathrm{def}} x$$

$$(MN)_{\mathrm{CCL},\lambda c} =_{\mathrm{def}} ((\mathrm{App} \circ \langle M_{\mathrm{DB}}, N_{\mathrm{DB}} \rangle)\, u)_{\lambda c} =_{\mathrm{def},\beta} (M_{\mathrm{DB},\lambda c} u)(N_{\mathrm{DB},\lambda c} u)$$
$$=_{\mathrm{ind}} MN \text{ where } u = (x, x_n, \ldots, x_0) \text{ (also below)}$$

$$(\lambda x.M)_{\mathrm{CCL},\lambda c} = (\Lambda(A)(y, x_n, \ldots, x_0))_{\lambda c} =_{\mathrm{def}} (\lambda uv.A_{\lambda c}(u, v))(y, x_n, \ldots, x_0)$$
$$=_{\beta} \lambda v.A_{\lambda c}(y, x_n, \ldots, x_0, v) =_{\mathrm{def}} \lambda v.M[x \leftarrow v]_{CCL,\lambda c}$$
$$=_{\mathrm{ind}} \lambda v.M[x \leftarrow v] \text{ for } A = M_{\mathrm{DB}(x,x_0,\ldots,x_n)} = M[x \leftarrow v]_{\mathrm{DB}(v,x_0,\ldots,x_n)}$$

$$(M, N)_{\mathrm{CCL},\lambda c} =_{\mathrm{def}} (\langle M_{\mathrm{DB}}, N_{\mathrm{DB}} \rangle\, u)_{\lambda c} =_{\mathrm{def},\beta} (M_{\mathrm{DB},\lambda c} u, N_{\mathrm{DB},\lambda c} u) =_{\mathrm{ind}} (M, N)$$

$$(\mathrm{fst}(M))_{\mathrm{CCL},\lambda c} =_{\mathrm{def}} ((\mathrm{Fst} \circ M_{\mathrm{DB}})\, u)_{\lambda c} =_{\mathrm{def},\beta} \mathrm{fst}(M_{\mathrm{DB},\lambda c} u) =_{\mathrm{ind}} \mathrm{fst}(M).$$

Finally we prove (4):

$$x_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \mathrm{Snd}(y, x) =_{\mathrm{snd}} x.$$

For the composition (and also $\langle\ ,\ \rangle$ and $\Lambda$), we use the fact (easily resulting from the proof of 2.17) that if $x \notin FV(M)$, then $M_{\mathrm{DB}(x,x_0,\ldots,x_n)} =_{CCL\beta} M_{\mathrm{DB}(x_0,\ldots,x_n)} \circ \mathrm{Fst}$:

$$(A \circ B)_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \Lambda(\mathrm{App} \circ \langle A_{\lambda c,\mathrm{DB}} \circ \mathrm{Fst}, \mathrm{App} \circ \langle B_{\lambda c,\mathrm{DB}} \circ \mathrm{Fst}, \mathrm{Snd} \rangle \rangle)\, u$$
$$=_{\mathrm{Quote}_3} (\mathrm{App} \circ \langle A_{\lambda c,\mathrm{DB}} \circ \mathrm{Fst}, \mathrm{App} \circ \langle B_{\lambda c,\mathrm{DB}} \circ \mathrm{Fst}, \mathrm{Snd} \rangle \rangle)$$
$$\circ \langle \Lambda(\mathrm{Fst})\, u, \mathrm{Id} \rangle =_{\mathrm{Ass,DPair,Fst,Snd,Quote}_2,\mathrm{ind,IdR}} A \circ B$$

$$\mathrm{Id}_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \Lambda(\mathrm{Snd})\, u =_{\mathrm{Quote}_3} \mathrm{Snd} \circ \langle \ldots, \mathrm{Id} \rangle =_{\mathrm{Snd}} \mathrm{Id}$$

$$\langle A, B \rangle_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \Lambda(\langle \mathrm{App} \circ \langle A_{\lambda c,\mathrm{DB}} \circ \mathrm{Fst}, \mathrm{Snd} \rangle, \mathrm{App} \circ \langle B_{\lambda c,\mathrm{DB}} \circ \mathrm{Fst}, \mathrm{Snd} \rangle \rangle) u$$
$$=_{\mathrm{Quote}_3,\mathrm{DPair,Ass,Fst,Snd,Quote}_2,\mathrm{ind,IdR}} \langle A, B \rangle$$

$$\mathrm{Fst}_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \Lambda(\mathrm{Fst} \circ \mathrm{Snd})\, u =_{\mathrm{Quote}_3} (\mathrm{Fst} \circ \mathrm{Snd}) \circ \langle \ldots, \mathrm{Id} \rangle =_{\mathrm{Ass,Snd,IdR}} \mathrm{Fst}$$

$$\Lambda(A)_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \Lambda(\Lambda(\mathrm{App} \circ \langle A_{\lambda c,\mathrm{DB}} \circ (\mathrm{Fst} \circ \mathrm{Fst}), \langle \mathrm{Snd} \circ \mathrm{Fst}, \mathrm{Snd} \rangle \rangle))\, u$$
$$=_{\mathrm{Quote}_3} \Lambda(\mathrm{App} \circ \langle A_{\lambda c,\mathrm{DB}} \circ (\mathrm{Fst} \circ \mathrm{Fst}), \langle \mathrm{Snd} \circ \mathrm{Fst}, \mathrm{Snd} \rangle \rangle)$$
$$\circ \langle \Lambda(\mathrm{Fst})\, u, \mathrm{Id} \rangle$$
$$=_{DA} \Lambda((\mathrm{App} \circ \langle A_{\lambda c,\mathrm{DB}} \circ (\mathrm{Fst} \circ \mathrm{Fst}), \langle \mathrm{Snd} \circ \mathrm{Fst}, \mathrm{Snd} \rangle \rangle)$$
$$\circ \langle \langle \Lambda(\mathrm{Fst})\, u, \mathrm{Id} \rangle \circ \mathrm{Fst}, \mathrm{Snd} \rangle)$$
$$=_{\mathrm{DPair,Ass,Fst,Snd,IdL}} \Lambda(\mathrm{App} \circ \langle (A_{\lambda c,\mathrm{DB}} \circ \Lambda(\mathrm{Fst})\, u)$$
$$\circ \mathrm{Fst}, \langle \mathrm{Fst}, \mathrm{Snd} \rangle) =_{\mathrm{Ass,Quote}_1,\mathrm{Quote}_2,\mathrm{ind}} \Lambda(A \circ \langle \mathrm{Fst}, \mathrm{Snd} \rangle)$$
$$=_{\mathrm{IdL}} \Lambda(A \circ \langle \mathrm{Id} \circ \mathrm{Fst}, \mathrm{Snd} \rangle) =_{DA} \Lambda(A) \circ \mathrm{Id} =_{\mathrm{IdR}} \Lambda(A)$$

$$\mathrm{App}_{\lambda c,\mathrm{CCL}} =_{\mathrm{def}} \Lambda(\mathrm{App} \circ \langle \mathrm{Fst} \circ \mathrm{Snd}, \mathrm{Snd} \circ \mathrm{Snd} \rangle)\, u$$
$$=_{\mathrm{Quote}_3,\mathrm{Ass},\mathrm{DPair},\mathrm{Snd},\mathrm{IdR}} \mathrm{App} \circ \langle \mathrm{Fst}, \mathrm{Snd} \rangle =_{\mathrm{FSA}} \mathrm{App}.$$

Our equivalence theorem makes critical use of the $\eta$-rule, which is needed in the proof of IdL and IdR, and would not fit together with the coding of coupling in the $\lambda$-calculus (cf. 2.9). If one wants to get an equivalence involving $\beta$ only or the $\lambda$-calculus without explicitly pairing, or both together, one has to turn from $\lambda$-calculus to a calculus defined on top of it, where expressions are couples

$$\langle (x_0, \ldots, x_n), M \rangle$$

of a formal environment (like the ones manipulated in the definition of De Bruijn's translation) and a $\lambda$-expression. This point of view stresses that global variables should not be implicitly bound as in the approach developed here, and that instead another kind of binding structure, in addition to abstraction, should be considered explicitly. This approach is developed in (Poigné, in press), and is implicit in Mann (1975), where proofs in natural deduction systems are manipulated and have precisely the structure above (formal environments being just sequents). However this formalism is heavier than ours because additional conversion rules have to be added, which may be thought as top level $\lambda$-calculus rules.

In Section 6 we shall give more references of related, semantically phrased work on equivalences between pure $\lambda$-calculus and categorical structures. Now we show as a corollary of 2.20 that $CCL\beta\eta SP$ is equivalent to $CCL + FSI$ plus an extensionality axiom.

2.21. COROLLARY. *For all terms $A$, $B$ of* **CCL** *the following holds*:

$$A =_{CCL\beta\eta SP} B \qquad \textit{iff } CCL, \mathrm{FSI}, \mathrm{ext} \vdash A = B$$

*where* ext *is the following* (*first-order, but not equational*) *axiom*:

$$(\mathrm{ext})\ Ax = Bx \Rightarrow A = B \ (x \textit{ occurs neither in } A \textit{ nor in } B).$$

*Proof.* First we check the equations of $CCL\beta\eta SP$ (except *FSI*) using *CCL* and *ext*. For Beta and D$\Lambda$:

$$(\mathrm{App} \circ \langle \Lambda(x), y \rangle)\, u = \Lambda(x)\, u(yu) = x(u, yu) = (x \circ \langle \mathrm{Id}, y \rangle)\, u$$
$$(\Lambda(x) \circ y)\, uv = \Lambda(x)(yu)\, v = x(yu, v) = (x \circ \langle y \circ \mathrm{Fst}, \mathrm{Snd} \rangle)(u, v)$$
$$= \Lambda(x \circ \langle y \circ \mathrm{Fst}, \mathrm{Snd} \rangle)\, uv.$$

Reciprocally we prove

$$Ax =_{CCL\beta\eta SP} Bx \Rightarrow A =_{CCL\beta\eta SP} B.$$

Indeed we have $(Ax)_{\lambda c} =_{\text{def}} A_{\lambda c}x$. Since $Ax =_{CCL\beta\eta SP} Bx$, we get

$$A_{\lambda c}x =_{\beta\eta SP} B_{\lambda c}x \qquad \text{and} \qquad A_{\lambda c} =_{\eta} \lambda x.A_{\lambda c}x =_{\beta\eta} \lambda x.B_{\lambda c}x =_{\eta} B_{\lambda c}$$

whence we conclude $A =_{CCL\beta\eta SP} A_{\lambda c,\mathrm{CCL}} =_{CCL\beta\eta SP} B_{\lambda c,\mathrm{CCL}} =_{CCL\beta\eta SP} B$.

Another corollary of Theorem 2.20 is a functional completeness result.

2.22. COROLLARY (Functional completeness). *For every term* $A \in \mathbf{CCL}$ s.t. $V(A) \subseteq \{x_0,..., x_n\}$ *there exists a unique closed term* $A^*$ *modulo* $CCL\beta\eta SP$ *s.t.*

$$A =_{CCL\beta\eta SP} A^*(x_n,..., x_0).$$

*Proof.* For the existence, we know by 2.20 that

$$A =_{CCL\beta\eta SP} A_{\lambda c,\mathrm{DB}}(x, x_n,..., x_0).$$

As $x$ does not occur in $A$ we verify by reading over the proof of 2.20 that we may also write, for instance,

$$A =_{CCL\beta\eta SP} A_{\lambda c,\mathrm{DB}}(\mathrm{Id}, x_n,..., x_0).$$

One defines easily a projection $\pi$ s.t.

$$(x, x_n,..., x_0) =_{CCL\beta\eta SP} \pi(x, (x_n,..., x_0)).$$

Then $A^* = \Lambda(A \circ \pi)\,\mathrm{Id}$ fits. Uniqueness is by *ext*.

Intuitively this result means that categorical combinatory logic is its own meta language: in the above statement $A$ may be seen as the specification of a function, and $A^*$ as the code for that function. One may also say that $A^*$ "internalizes" the "function" $(x_n,..., x_0) \mapsto A$.

The same result, expressed in a semantic setting, was shown using a different method by Lambek and Scott (1985). 2.20–2.22 remain true if one replaces

$$\left.\begin{matrix} \beta\eta SP \\ CCL\beta\eta SP \\ \mathrm{FSI} \end{matrix}\right\} \quad \text{by} \quad \left\{\begin{matrix} \beta\eta P \\ CCL\beta\eta P. \\ \mathrm{FSA} \end{matrix}\right.$$

One easily incorporates constants in these results by defining for any constant $C$ of **λc, CCL**:

$$M_{\mathrm{DB}}(C) = \Lambda(\mathrm{Fst})\,C, \qquad C_{\lambda c} = C.$$

The following proposition lists some properties of categorical rewriting systems.

2.23. PROPOSITION. *The following rewriting systems are locally confluent (equations are orientated from left to right)*:

$$\mathrm{Ass} + \mathrm{IdL}$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR}\ (\text{monoid})$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{AssFst} + \mathrm{AssSnd}$$

*where*

$$(\mathrm{AssFst})\ \mathrm{Fst} \circ (\langle x, y \rangle \circ z) = x \circ z$$

$$(\mathrm{AssSnd})\ \mathrm{Snd} \circ (\langle x, y \rangle \circ z) = y \circ z$$

(*one has* Ass, Fst⊢—AssFst *and* Ass, Snd⊢—AssSnd)

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair}$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair} + \mathrm{FSI} + \mathrm{SPair}$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair} + \mathrm{FSI} + \mathrm{SPair} + DA$$

(*one has* Ass, Fst, Snd, SPair⊢—DPair *and* SPair, IdR⊢—FSI).

*Up to here these systems are also noetherian*

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair} + \mathrm{Beta} + \mathrm{Beta}'$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair} + \mathrm{FSI} + \mathrm{SPair} + \mathrm{Beta} + \mathrm{Beta}'$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{rIdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair} + \mathrm{FSI} + \mathrm{SPair} + \mathrm{Beta} + DA.$$

*The following systems are not locally confluent*:

$$\mathrm{Ass} + \mathrm{IdR}$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd}$$

$$\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Fst} + \mathrm{Snd} + \mathrm{DPair} + \mathrm{Beta} + DA.$$

*Proof.* For the local influence, check the critical pairs or rely on the software written to complete rewriting systems (Formel (INRIA, Huet and Cousineau), Reve (CRIN-MIT, P. Lescanne)). For the negative results, we remark that the three systems suffer from the absence of IdL, DPair, and FSI, respectively (details omitted).

We do not know how to get a locally confluent version of $CCL\beta\eta SP$. In an unpublished note, Cartmell has exhibited canonical (i.e., confluent and noetherian) rewrite rules for the adjunctions, as summarized in the exercise below.

EXERCISE. Adjunctions may be characterized equationally (forgetting about objects) by the following set of equations, over the signature of monoids augmented with unary operators $F, \zeta, \eta$:

$$\begin{aligned}
&\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Hom}_1 + \mathrm{Hom}_2, \text{ where} \\
&\quad (\mathrm{Hom}_1) \qquad F(x) \circ F(y) = F(x \circ y) \\
&\quad (\mathrm{Hom}_2) \qquad F(\mathrm{Id}) = \mathrm{Id} \\
&\zeta(\eta(x)) = \eta(\zeta(x)) = x \\
&\zeta(y \circ F(x)) = \zeta(y) \circ x.
\end{aligned}$$

Show that an equivalent formalization is provided on the signature of monoids augmented by unary $F, \zeta$, and 0-ary $\varepsilon$ by

$$\begin{aligned}
&\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Hom}_1 + \mathrm{Hom}_2 \\
&\zeta(\varepsilon \circ F(x)) = x \\
&\varepsilon \circ F(\zeta(x)) = x.
\end{aligned}$$

Prove that the following set of rules is equivalent to the previous ones and canonical:

$$\begin{aligned}
&\mathrm{Ass} + \mathrm{IdL} + \mathrm{IdR} + \mathrm{Hom}_1 + \mathrm{Hom}_2 \\
&F(x) \circ (F(y) \circ z) \to F(x \circ y) \circ z \\
&\zeta(\varepsilon \circ F(x)) \to x \\
&\zeta(\varepsilon) \to \mathrm{Id} \\
&\varepsilon \circ F(\zeta(x)) \to x \\
&\varepsilon \circ (F(\zeta(x)) \circ y) \to x \circ y \\
&\zeta(y) \circ x \to \zeta(y \circ F(x)).
\end{aligned}$$

But this does not help for $CCL\beta\eta SP$, because two adjunctions are involved, one of which is built on the top of the other, provoking interferences.

## 3. Classical Combinatory Logic

The following is a brief account of the similar results known in combinatory logic. They go back to Curry and Feys (1958). We add our personal touch by rendering syntactic recent contributions of Scott (1980) and Meyer (1982), unified by Koymans (1984): these authors have discussed semantic combinatorial characterizations of $\lambda$-models, but their constructions may be rephrased syntactically and then become characterizations of $\lambda$-algebras ($\lambda$-algebras and $\lambda$-models are discussed in Sect. 5, but are not needed here).

We shall first introduce combinatory logic, then discuss the equivalence with $\beta$-conversion, and finally we shall discuss the efficiency of known translations into combinatory logic, insisting on the combinator strings of Kennaway and Sleep (1982), which have some similarities with categorical combinators.

Combinatory algebras are defined as follows. A **combinatory algebra** is a structure $(D, \cdot)$, i.e. a set $D$ endowed with an **application** operation, denoted by simple juxtaposition, s.t. any term $A$ in $T(\{x_1,..., x_n\})$ (i.e., built with application and variables $x_1,..., x_n$) has associated with it an element $A^*$ in $D$ s.t.

$$\forall d_1,..., d_n \in D, \qquad \{x_1 = d_1,..., x_n = d_n\}^{\#}_D (A) = A^* d_1,..., d_n.$$

This property is called combinatory completeness. This is like functional completeness above, but uniqueness is not required.

The striking property of combinatory algebras is that combinatory completeness follows from only two of its instances, the terms $(xz)(yz)$, and $x$ (considered as a term in $T\{x, y\}$). This justifies the introduction of the following language of **combinatory logic CL**. The signature of combinatory logic is made of $S$ and $K$ of arity 0, and application of arity 2. $CL$ is the following set of equations

$$(S) \qquad Sxyz = (xz)(yz)$$

$$(K) \qquad Kxy = x$$

known as the **weak rules** of combinatory logic. We set $I$ as an abbreviation of $SKK$, and check easily

$$Ix =_{\mathrm{CL}} x.$$

To show that combinatory algebras are exactly the models of **CL**, $CL$, we build the **abstraction algorithm**, associating with any terms $s$ a term $[x].s$ with no occurrence of $x$, as follows:

(i) $[x].x = I$

(ii) $[x].A = KA$ if $x \notin V(A)$

(iii) $[x].AB = S([x].A)([x].B)$ if (ii) does not apply.

This definition is justified by the following property

$$([x].M)\,x =_{CL} M. \qquad (*)$$

Then combinatory completeness follows by taking above $A^* = [x_1,..., x_n].A$ (using the same notation as for multiple $\lambda$-abstraction). Actually what we get is a *syntactic* form of combinatory completeness, namely

$$A =_{CL} A^* x_1,..., x_n.$$

Another property of abstraction is

$$([x].A)[y \leftarrow B] =_{CL} [x].(A[y \leftarrow B]) \qquad \text{if } x \notin V(B). \qquad (**)$$

(Here the clause (ii) in the definition of abstraction is needed.) We are now in a position to define translations between $\lambda$ and **CL**. Define

$$x_{\mathrm{CL}} = x$$

$$(MN)_{\mathrm{CL}} = M_{\mathrm{CL}} N_{\mathrm{CL}}$$

$$(\lambda x.M)_{\mathrm{CL}} = [x].M_{\mathrm{CL}}$$

and

$$K_\lambda = \lambda xy.x$$

$$S_\lambda = \lambda xyz.(xz)(yz)$$

$$(MN)_\lambda = M_\lambda N_\lambda.$$

The following exercise suggests that the weak rules of categorical combinatory logic are weaker than those of classical combinatory logic.

EXERCISE. When is it true that

$$(\lambda x....(\lambda y.M)\,N...)_{\mathrm{CL}} =_{CL} (\lambda x....M[y \leftarrow N]...)_{\mathrm{CL}}.$$

We want to establish a syntactic equivalence in the style of the equivalence theorem of this section. First we want

$$M =_\beta N \Rightarrow M_{\mathrm{CL}} = N_{\mathrm{CL}}. \qquad (1)$$

From (*) and (**), it is easily seen that (1) holds for head redexes, i.e., $M = (\lambda x.P)\,Q$, $N = P[x \leftarrow Q]$. So to get (1) we are left with

$$(\xi)\; A = B \Rightarrow [x].A = [x].B.$$

We leave property (1) for a while, and turn to

$$A = B \Rightarrow A_\lambda = B_\lambda. \tag{2}$$

This will depend on the new axioms we shall need to ensure $\xi$. But it will be only routine verification, just as

$$M_{\mathrm{CL},\lambda} = M. \tag{3}$$

So, besides $\xi$ the only other difficulty will reside in

$$A_{\lambda,\mathrm{CL}} = A. \tag{4}$$

This reduces to the instances $S$ and $K$. Using $\xi$ and (*), we get

$$K_{\lambda,\mathrm{CL}} = [xy].x = [xy].Kxy = 1_2 K$$
$$S_{\lambda,\mathrm{CL}} = [xyz].Sxyz = 1_3 S$$

where for any $n$

$$1_n = [xx_1 \ldots x_n].xx_1 \ldots x_n \qquad \text{(we set } 1_1 = 1\text{)}.$$

So (4) is ensured by the two axioms

$$(K\lambda) \qquad 1_2 K = K$$
$$(S\lambda) \qquad 1_3 S = S.$$

Moreover these two axioms allow us to make some progress on $\xi$. We remark that we have then

$$1([x].M) = [x].M$$

for any $M$, since $[x].M$ has always the form $SPQ$ or $KP$ (remember $I = SKK$), and indeed

$$SPQ = 1_3 SPQ = [x].SPQx = 1(SPQ)$$

and likewise for $K$. Now $\xi$ has something to do with extensionality. Indeed the hypothesis $M = N$ may be rephrased as

$$([x].M)\,x = ([x].N)\,x.$$

Hence $\xi$ is surely implied by *ext* (cf. 2.21). The following exercise proves however that *ext* is too strong for our purposes.

EXERCISE. Prove that CL + *ext* yields a syntactic equivalence with $\beta\eta$.
We need a weak extensionality axiom (not to be confused with the semantic concept of weak extensionality (see 5.1)), and the discussion above suggests

$$\text{(wext)} \qquad Ax = Bx \Rightarrow 1A = 1B.$$

In the presence of $K\lambda$ and $S\lambda$, this ensures $\xi$. So we have got a syntactic equivalence between $\lambda.\beta$, and **CL** with the system

$$\begin{aligned} &CL + \\ &(K\lambda) \qquad K = 1_2 K \\ &(S\lambda) \qquad S = 1_3 S \\ &\text{(wext)} \qquad Ax = Bx \Rightarrow 1A = 1B. \end{aligned}$$

Actually the axioms of Meyer and Scott use sequences other than $1, 1_2, 1_3$, and make use of an extra operator $\varepsilon$ in the syntax. But they are essentially the same (for details we refer to Curien (1985)).

So far, we have not yet produced an equational theory of classical combinatory logic since wext is first order, but not equational. Indeed wext can be replaced in our first axiom system by three quite ugly axioms, which we introduce now.

Here clause (ii) of the abstraction algorithm is unfortunate, since if we could rely on (iii) without restriction, the verification of $\xi$ would reduce to

$$[x].KPQ = [x].P \qquad \text{and} \qquad [x].SPQR = [x].(PR)(QR).$$

So we need, for any $P, Q$ s.t. $x \notin V(PQ)$,

$$K(PQ) = S(KP)(KQ).$$

Now, introducing new equations, we need to take care that they will be involved in the verification of $\xi$; we have to close them, because then $\xi$ holds trivially because of clause (i). Hence the necessary equation is

$$\text{(abs)} \qquad [xy].K(xy) = [xy].S(Kx)(Ky).$$

Now set $A = [x].P$ and $B = [x].Q$. Then the required property of $K$ becomes

$$S(S(KK)\,A)\,B = A$$

so that we finally get a syntactic equivalence between $\lambda$, $\beta$ and **CL**, $CL\beta$, where $CL\beta$ is

$$CL + K\lambda + S\lambda +$$

(abs) $\quad [xy].K(xy) = [xy].S(Kx)(Ky)$

$(K\xi)$ $\quad [xy].S(S(KK)\,x)\,y = [xy].x$

$(S\xi)$ $\quad [xyz].S(S(S(KS)\,x)\,y)\,z = [xyz].S(Sxz)(Syz)$

Finally if we want the equivalence with $\beta\eta$, we need

$$[x].Ux = U \qquad \text{if } x \notin U.$$

We get an equivalence between $\lambda$, $\beta\eta$ and **CL**, $CL\beta\eta$, where $CL\beta\eta$ is

$$CL\beta +$$

(Eta) $\quad [x].S(Kx)\,I = [x].x.$

We hope to have convinced the reader that the Curry axioms are quite "natural" if we mean that they arise in a simple way when trying to get the equivalence. But this is a rather ad hoc naturalness, whereas categorical axioms are natural and intuitive by themselves.

We end the section by a short account of combinator strings, a recent proposal of Kennaway and Sleep (1982) for an efficient translation of $\lambda$-expressions into an interesting rephrasing of the classical combinators, which is in a way symmetric to the De Bruijn's translation. The abstraction algorithm presented above is dramatically inefficient (the explosion of the size is exponential). The point is that, when making successive abstractions, the structure of the initial body of the innermost abstraction is lost, as is easily seen when compiling $\lambda xy.yx$. Indeed

$$[y].yx = S([y].y)([y].x) = SI(Kx)$$

$$[x].(SI(Kx)) = S([x].(SI))([x].(Kx)) = S(K(SI))(S([x].K)([x].x))$$

$$= S(K(SI))(S(KK)\,I).$$

The structure of the body $yx$, a simple application of two variables, has been lost when performing the first abstraction: what has been obtained is the application of a constant expression $(SI)$ to an application of a constant to a variable. This is the beginning of an explosion.

Turner (1979a, b) formulated successive refinements to obtain more efficient algorithms, which we propose as exercises.

EXERCISE. Show that, using the abstraction algorithm defined above, the worst case space complexity of the translation is exponential. Notice that the typical worst case is

$$\lambda x_1 \ldots x_n . M_1 M_2 .$$

EXERCISE. Design a new abstraction algorithm which divides the case (iii) into three subcases, according to whether $x$ occurs in $A$ only, $B$ only, or both. For the first two cases use the combinators $C$, $B$, which have the following definitions:

$$(C) \qquad Cxyz = (xz)\ y$$

$$(B) \qquad Bxyz = x(yz).$$

Show that the obtained algorithm has a worst case cubic complexity (hint: take as worst case the same expression as in the preceding exercise, supposing that $x_1, \ldots, x_n$ appear all in $M_1$ and all in $M_2$).

EXERCISE. The reader will have noticed that the trick in the previous exercise does not prevent the explosion quoted above. We introduce three new combinators $S'$, $C'$, and $B'$ with the following defining equations:

$$(S') \qquad S'txyz = t(xz)(yz)$$

$$(C') \qquad C'txyz = t(xz)\ y$$

$$(B') \qquad B'txyz = tx(yz).$$

Find a modified abstraction algorithm using these combinators, such that the space complexity becomes quadratic.

We shall introduce the combinator strings (which are actually an elegant rephrasing of the optimization in the last exercise, see exercise below) in a more detailed way, since they preserve the structure of the compiled expressions just as the categorical translation does. Our notation is slightly different from the one in Kennaway and Sleep (1982). The idea is to introduce infinitely many combinators, which are strings decorating application nodes, and which indicate how the variables are distributed in an expression (the exercises above have suggested how useful it is to detect where abstracted variables appear in an expression). There are four characters, or **directors**, corresponding to the four possible cases:

$\#$: there is no occurrence of the variable

$\hat{}$ : the variable appears, on both sides if the expression is an application

/: the expression is an application, and the variable appears only on the left

\: the expression is an application, and the variable appears only on the right.

Now, when compiling multiple abstractions such as $\lambda xy.\, yx$ above, these characters are catenated to form strings. Here is the syntax of the modified combinatory logic (the only constant is now $I$: $S$, $K$ are incorporated in director strings), where $u$ is a word built on the alphabet $\{\#, \hat{}, /, \backslash\}$:

$[x, u]$ is an expression

$[I, u]$ is an expression

$[AB, u]$ is an expression, if $A$, $B$ are expressions.

One writes simply $[AB, \varepsilon] = AB$. The computation rules are as follows:

$$[x, \#u]\, C = [x, u]$$
$$[I, \#u]\, C = [I, u]$$
$$[AB, \#u]\, C = [AB, u]$$
$$[I, \hat{}u]\, C = [C, u]$$
$$[AB, \hat{}u]\, C = [(AC)(BC), u]$$
$$[AB, /u]\, C = [(AC)\, B, u]$$
$$[AB, \backslash u]\, C = [A(BC), u].$$

Now the compilation rules are as follows:

$$[x].[x, u] = [I, \hat{}u]$$
$$[x].[y, u] = [y, \#u]$$
$$[x].[I, u] = [I, \#u]$$
$$[x].[AB, u] = [AB, \#u] \text{ if } x \text{ does appear neither in } A, \text{ neither in } B$$
$$[x].[AB, u] = [([x].A)([x].B), \hat{}u] \text{ if } x \text{ appears in both } A \text{ and } B$$
$$[x].[AB, u] = [([x].A)\, B, /u] \text{ if } x \text{ appears in } A, \text{ but not in } B$$
$$[x].[AB, u] = [A([x].B), \backslash u] \text{ if } x \text{ appears in } B, \text{ but not in } A.$$

Hence the structure of the initial expression is preserved. The coding of variables is very similar to the De Bruijn's notation. The difference is that

abstractions have disappeared and are replaced by strings attached to application nodes, which indicate where to transmit the arguments.

As an example we compile the expression

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

We have

$$\begin{aligned}
[z].zx &= [([z].z)\ x, /] = [[I, \hat{}]\ x, /] \\
[x].(([[I, \hat{}]\ x, /])\ y) &= [([x].[[I, \hat{}]\ x, /])\ y, /] \\
&= [[[I, \hat{}]([x].x), \backslash]\ y, /] \\
&= [[[I, \hat{}][I, \hat{}], \backslash]\ y, /],
\end{aligned}$$

so that the compilation of the whole expression is

$$[[[I, \hat{}][I, \hat{}], \backslash]\ y, /]([I, \hat{}]\ z).$$

The reader should draw a tree for this expression and compare it to the tree of the corresponding De Bruijn's expression. Now we execute the code (in a leftmost–outermost way):

$$\begin{aligned}
&[[[I, \hat{}][I, \hat{}], \backslash]\ y, /]([I, \hat{}]\ z) \\
&\quad \to ([[I, \hat{}][I, \hat{}], \backslash]([I, \hat{}]\ z))\ y \\
&\quad \to ([I, \hat{}]([I, \hat{}]([I, \hat{}]\ z)))\ y \\
&\quad \to ([I, \hat{}]([I, \hat{}]\ z))\ y \\
&\quad \to ([I, \hat{}]\ z)\ y \\
&\quad \to zy.
\end{aligned}$$

The categorical translation which we propose, as well as the translation into combinator strings, may be considered as linear if one does not take care of the place needed to store arbitrary integers and arbitrary strings, respectively. If one assumes that storing $n$ takes $\log n$ places in memory, then the worst case complexity of both translations is $n(\log n)$.

Summarizing our short digression into classical combinatory logic, the categorical translation is as efficient as the best known translation into classical combinators. Moreover the strong rules of categorical combinatory logic are simple whereas Curry axioms are untractable. Strong rules are important, because they often give rise to compile time optimizations, as shown in Cousineau, Curien, and Mauny (1985).

## 4. Types and Cartesian Closed Categories

In this section we introduce type constraints into the $\lambda c$-calculus and the categorical combinatory logic. We show that the equivalence theorem of Section 2 goes through these constraints. Then we establish a correspondence between the typed categorical combinatory logic and the free Cartesian closed category, the models of which are the Cartesian closed categories with objects freely constructed by product and exponential from a set of basic objects. Roughly the applying and coupling operators, which need to be considered as primitive in the typed categorical combinatory logic, are coded with the terminal object in the free Cartesian closed category, and vice versa. The point is that categories know only about arrows, not about elements of objects. To handle those elements the trick is to consider arrows from the terminal object to the concerned object. Then applying and coupling are nothing but "degenerate" cases of composition, pairing, where one, two arrows start from the terminal object.

The following example may help to understand what is going on. Suppose we want to code integers in the setting of Cartesian closed categories. We suppose that there exists an object $\iota$ representing the integers, and integers are then arrows from the terminal object 1 to $\iota$. Now a first question arises. Is the function *succ* to be considered as an arrow from $\iota$ to $\iota$ or as an arrow from 1 to the exponential $\iota \Rightarrow \iota$? The second choice is consistent with the coding of integers, while the first choice is more suited for coding the application of *succ* to, say, 2: one just needs to compose the arrows:

$$f^{\iota \Rightarrow \iota} \circ 2^{1 \Rightarrow \iota}.$$

The two choices are equivalent by a well-known isomorphism associating with any arrow $f$ from, say, $A$ to $B$ an "element" of $A \Rightarrow B$, i.e., an arrow from 1 to $A \Rightarrow B$, called the name of $f$ (this isomorphism will be recalled formally below).

So let us take the first choice and see if it resists to a more involved example. Suppose that *plus* is given as a curried function, so that it is coded by an arrow from $\iota$ to $\iota \Rightarrow \iota$. How shall we code *plus* 2 3? Composing *plus* with 2 yields an arrow from 1 to $\iota \Rightarrow \iota$, thus forcing the second choice for the coding of the function *plus* 2. In order to be able to apply *plus* 2 to 3 we need to transform *plus* 2 by the isomorphism mentioned above, which we denote by $^-$, so that the following somewhat unnatural coding is obtained finally:

$$(plus \circ 2)^- \circ 3.$$

The point is that mathematicians do not care much about canonical isomorphisms, while computers have to cope with them explicitly. This short discussion should justify our introduction of typed categorical combinatory logic, which arises naturally from Section 2, and where such painful codings are avoided. The price to pay is that the unique equation for the terminal object (*Ter*, see below) has to be replaced by the whole set of weak rules (and *Quote*). But we have seen how interesting these rules are for implementation purposes.

The results of the section are applied to the decidability problem for the equational equality in the free Cartesian closed category.

*Introducing Type Constraints*

We define the typed $\lambda c$-calculus and the typed categorical combinatory logic, and give a typed version of last section's equivalence theorem.

4.1. DEFINITION. The **$K$-typed $\lambda$-calculus $\lambda\mathbf{c}_K$** and the **$K$-typed categorical combinatory logic $\mathbf{CCL}_K$** 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 $\sigma$, we write $M^\sigma$ or $M:\sigma$. We agree that $\times$ has precedence over $\Rightarrow$, and we write

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

The structure of terms is as follows: For $\lambda\mathbf{c}_K$:

if $x$ is a variable and $\sigma$ is a type, then $x:\sigma$

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 $\mathrm{fst}(M):\sigma$

if $M:\sigma \times \tau$, then $\mathrm{snd}(M):\tau$.

For $\mathbf{CCL}_K$:

if $x$ is a variable and $\sigma$ is a type then $x:\sigma$

if $A:\sigma_2 \Rightarrow \sigma_3$ and $B:\sigma_1 \Rightarrow \sigma_2$ then $A \circ B:\sigma_1 \Rightarrow \sigma_3$

$\mathrm{Id}:\sigma \Rightarrow \sigma$

if $A:\sigma \Rightarrow \tau_1$ and $B:\sigma \Rightarrow \tau_2$ then $\langle A, B\rangle:\sigma \Rightarrow \tau_1 \times \tau_2$

$\mathrm{Fst}:\sigma \times \tau \Rightarrow \sigma$ (we shall often write $\mathrm{Fst}^{\sigma,\tau}$)

$\mathrm{Snd}:\sigma \times \tau \Rightarrow \tau$ (we shall often write $\mathrm{Snd}^{\sigma,\tau}$)

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 $\mathrm{App}^{\sigma,\tau}$)

if $A: \sigma \Rightarrow \tau$ and $B: \sigma$ then $AB: \tau$

if $A: \sigma$ and $B: \tau$ then $(A, B): \sigma \times \tau$.

Hence $\mathbf{CCL_K}$ is an algebra of first order terms.

Let $\beta\eta SP_K$ and $AA_K$ be the typed versions of the theories $\beta\eta SP$ and $CCL\beta\eta SP + CCL - \mathrm{id} - \mathrm{d}\Lambda + \mathrm{Quote} - \mathrm{Quote}_3$ of Section 2, now written as (we keep the same names for the equations, and types are only specified at the first occurrence of a variable or a subterm)

$\beta\eta SP_K$:

| | |
|---|---|
| $(\beta)$ | $(\lambda x^{\sigma}.M^{\tau})\, N^{\sigma} = M[x \leftarrow N]$ |
| $(\eta)$ | $\lambda x^{\sigma}.M^{\sigma \Rightarrow \tau} x = M$ if $x \notin FV(M)$ |
| (fst) | $\mathrm{fst}(M^{\sigma}, N^{\tau}) = M$ |
| (snd) | $\mathrm{snd}(M^{\sigma}, N^{\tau}) = N$ |
| $(SP)$ | $(\mathrm{fst}(M^{\sigma \times \tau}), \mathrm{snd}(M)) = M.$ |

$AA_K$:

| | |
|---|---|
| (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) | $\mathrm{Id}^{\tau \Rightarrow \tau} \circ x^{\sigma \Rightarrow \tau} = x^{\sigma \Rightarrow \tau}$ |
| (IdR) | $x^{\sigma \Rightarrow \tau} \circ \mathrm{Id}^{\sigma \Rightarrow \sigma} = x$ |
| (Fst) | $\mathrm{Fst}^{\tau_1,\tau_2} \circ \langle x^{\sigma \Rightarrow \tau_1}, y^{\sigma \Rightarrow \tau_2} \rangle = x$ |
| (Snd) | $\mathrm{Snd}^{\tau_1,\tau_2} \circ \langle x^{\sigma \Rightarrow \tau_1}, y^{\sigma \Rightarrow \tau_2} \rangle = y$ |
| (DPair) | $\langle x^{\sigma_1 \Rightarrow \tau_1}, y^{\sigma_1 \Rightarrow \tau_2} \rangle \circ z^{\sigma \Rightarrow \sigma_1} = \langle x \circ z, y \circ z \rangle$ |
| (Beta) | $\mathrm{App}^{\sigma_2,\sigma_3} \circ \langle \Lambda(x^{\sigma_1 \times \sigma_2 \Rightarrow \sigma_3}), y^{\sigma_1 \Rightarrow \sigma_2} \rangle = x \circ \langle \mathrm{Id}^{\sigma_1 \Rightarrow \sigma_1}, y \rangle$ |
| $(D\Lambda)$ | $\Lambda(x^{\sigma_1 \times \sigma_2 \Rightarrow \sigma_3}) \circ y^{\sigma \Rightarrow \sigma_1} = \Lambda(x \circ \langle y \circ \mathrm{Fst}^{\sigma,\sigma_2}, \mathrm{Snd}^{\sigma,\sigma_2} \rangle)$ |
| (AI) | $\Lambda(\mathrm{App}^{\sigma,\tau}) = \mathrm{Id}^{(\sigma \Rightarrow \tau) \Rightarrow (\sigma \Rightarrow \tau)}$ |
| (FSI) | $\langle \mathrm{Fst}^{\sigma,\tau}, \mathrm{Snd}^{\sigma,\tau} \rangle = \mathrm{Id}^{\sigma \times \tau \Rightarrow \sigma \times \tau}$ |
| (ass) | $(x^{\sigma_1 \Rightarrow \sigma_2} \circ y^{\sigma \Rightarrow \sigma_1})\, z^{\sigma} = x(yz)$ |
| (fst) | $\mathrm{Fst}^{\sigma_1,\sigma_2}(x^{\sigma_1}, y^{\sigma_2}) = x$ |
| (snd) | $\mathrm{Snd}^{\sigma_1,\sigma_2}(x^{\sigma_1}, y^{\sigma_2}) = y$ |
| (dpair) | $\langle x^{\sigma \Rightarrow \tau_1}, y^{\sigma \Rightarrow \tau_2} \rangle\, z^{\sigma} = (xz, yz)$ |
| (app) | $\mathrm{App}^{\sigma,\tau}(x^{\sigma \Rightarrow \tau}, y^{\sigma}) = xy$ |
| ($\mathrm{Quote}_1$) | $\Lambda(\mathrm{Fst}^{\sigma,\sigma_2})\, x^{\sigma} \circ y^{\sigma_1 \Rightarrow \sigma_2} = \Lambda(\mathrm{Fst}^{\sigma,\sigma_1})\, x$ |
| ($\mathrm{Quote}_2$) | $\mathrm{App}^{\sigma_2,\sigma_3} \circ \langle x^{\sigma \Rightarrow (\sigma_2 \Rightarrow \sigma_3)} \circ \Lambda(\mathrm{Fst}^{\sigma,\sigma_1})\, y^{\sigma}, z^{\sigma_1 \Rightarrow \sigma_2} \rangle = xy \circ z.$ |

Some of these equations must be applied with caution. For instance, we can only replace Id by $\Lambda(\mathrm{App})$, $\langle \mathrm{Fst}, \mathrm{Snd} \rangle$ if Id is of type $(\sigma \Rightarrow \tau) \Rightarrow$

$(\sigma \Rightarrow \tau)$, $\sigma \times \tau \Rightarrow \sigma \times \tau$ respectively; to replace a subterm $A$ by $\mathrm{Fst} \circ \langle A, B \rangle$ we need to check $A: \sigma \Rightarrow \tau_1$ and $B: \sigma \Rightarrow \tau_2$ with the same $\sigma$. In contrast one can safely replace $\Lambda(\mathrm{App})$ or $\langle \mathrm{Fst}, \mathrm{Snd} \rangle$, since the typing required in the equations AI and FSI is the most general type for the left members of the untyped concerned equations.

The point is that two types may be associated with the left or right members of the untyped equations:

— its most general type in the typed calculus

— the most general type it may have such that the equation may be applied, which is the less general among the most general types of both members.

For the right member of AI, the two types are $\sigma \Rightarrow \sigma$, $(\sigma_1 \Rightarrow \sigma_2) \Rightarrow (\sigma_1 \Rightarrow \sigma_2)$ respectively (for any $\sigma, \sigma_1, \sigma_2$). The equations concerned are $\eta$, *SP*, IdL, IdR, Fst, Snd, AI, and FSI.

We observe another important difference with Section 2: now applying and coupling are *primitive*. Indeed, the equations of Definition 2.13 cannot be typed satisfactorily: we want to define $xy$ for any $y: \sigma$ whereas the definition of $xy$ involves $y^>$ which makes sense only for $y: \sigma_1 \Rightarrow \sigma_2$.

We have chosen to allow a variable to have any type, whereas usually one considers a distinct set of variable names for each type. This is to allow a more natural translation between typed categorical combinatory logic and the free Cartesian closed category (see 4.8 below). We consider types as being explicitly given with the terms, rather than being inferred by suitable rules. Hence to be perfectly clear the formation rule for, say, the abstraction, should be read

$$(\lambda x^{\sigma}.M^{\tau})^{\sigma \Rightarrow \tau}.$$

Finally we note that the structural rules for building couples and abstractions do not imply type constraints (in other words, they define total functions).

One can verify easily that the typed versions of $\mathrm{Quote}_3$, id, and $\mathrm{d}\Lambda$ are consequences of $AA_K$ (cf. Proof of 2.14).

The following breakdown of $\mathrm{Quote}_2$ will be useful.

4.2. LEMMA. *The system $AA_K$ is equivalent to the system obtained by replacing* $\mathrm{Quote}_2$ *by the two following equations*:

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

$$(\mathrm{Quote}_{2b}) \qquad \Lambda(\mathrm{Fst}^{\sigma_2, \sigma})(x^{\sigma_1 \Rightarrow \sigma_2} y^{\sigma_1}) = x \circ \Lambda(\mathrm{Fst}^{\sigma_1, \sigma})\, y.$$

*Proof.* First we show $\text{Quote}_2$:

$$\begin{aligned}\text{App} \circ \langle x^{\sigma \Rightarrow (\sigma_2 \Rightarrow \sigma_3)} \circ \Lambda(\text{Fst})\ y, z \rangle &=_{\text{Quote}_{2a}} \text{App} \circ \langle \Lambda(\text{Fst})((xy)^{\sigma_2 \Rightarrow \sigma_3}), z \rangle \\ &=_{\text{Quote}_{2a}} \text{App} \circ \langle \Lambda(xy \circ \text{Snd}), z \rangle \\ &=_{\text{Beta,Snd}} xy \circ z.\end{aligned}$$

Then $\text{Quote}_{2a}$:

$$\begin{aligned}(\Lambda(\text{Fst})\ x)^{\sigma \Rightarrow (\sigma_1 \Rightarrow \sigma_2)} &=_{\text{AI}} \Lambda(\text{App}^{\sigma_1,\sigma_2}) \circ \Lambda(\text{Fst})\ x \\ &=_{\text{D}\Lambda,\text{Quote}_1} \Lambda(\text{App} \circ \langle \Lambda(\text{Fst})\ x, \text{Snd} \rangle) \\ &=_{\text{IdL,Quote}_2} \Lambda(\text{Id}\ x \circ \text{Snd}) =_{\text{id}} \Lambda(x \circ \text{Snd}).\end{aligned}$$

Finally $\text{Quote}_{2b}$:

$$\begin{aligned}\Lambda(\text{Fst})(xy) &=_{\text{ass}} (\Lambda(\text{Fst}) \circ x)\ y =_{\text{D}\Lambda,\text{Fst}} \Lambda(x \circ \text{Fst})\ y \\ &=_{\text{Quote}_3} x \circ \text{Fst} \circ \langle \Lambda(\text{Fst})\ y, \text{Id} \rangle =_{\text{Fst}} x \circ \Lambda(\text{Fst})\ y.\end{aligned}$$

Now we define the typed versions of the translations $_{\text{DB}}$, $_{\text{CCL}}$ and $_{\lambda c}$.

4.3. DEFINITION. Let $M: \sigma \in \lambda \mathbf{c}_{\mathbf{K}}$, and $x_0: \sigma_0, \ldots, x_n: \sigma_n$ be s.t. $FV(M) \subseteq \{x_0, \ldots, x_n\}$. We define $M_{\text{DB}_K(x_0,\ldots,x_n)}$ as in 2.15, with types as follows ($\sigma$ is any type):

$$x_{\text{DB}_K(x_0,\ldots,x_n)} = \text{Snd}^{\sigma \times \sigma_n \cdots \times \sigma_{i+1}, \sigma_i} \circ \text{Fst}^{\sigma \times \sigma_n \cdots \times \sigma_i, \sigma_{i-1}} \circ \cdots \circ \text{Fst}^{\sigma \times \sigma_n \cdots \times \sigma_1, \sigma_0}.$$

The other cases are as in 2.15. One has

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

($\sigma_0, \ldots, \sigma_n, \tau$ are determined by $M, x_0, \ldots, x_n$ while $\sigma$ is any type). As in 2.18 we define

$$M_{\text{CCL}_K} = M^{\tau}_{\text{DB}_K(x_0^{\sigma_0},\ldots,x_n^{\sigma_n})}(y^{\sigma}, x_n^{\sigma_n}, \ldots, x_0^{\sigma_0})$$

where $y$ is different from all $x_i$ and has the type $\sigma$ in $M_{\text{DB}_K}$.

Here is the typed version of $_{\lambda c}$:

$$\begin{aligned}x^{\sigma}_{\lambda c_K} &= x^{\sigma} \\ \text{Id}^{\sigma \Rightarrow \sigma}_{\lambda c_K} &= \lambda x^{\sigma}.x \\ \text{Fst}^{\sigma,\tau}_{\lambda c_K} &= \lambda x^{\sigma \times \tau}.\text{fst}(x) \\ \text{Snd}^{\sigma,\tau}_{\lambda c_K} &= \lambda x^{\sigma \times \tau}.\text{snd}(x) \\ \text{App}^{\sigma,\tau}_{\lambda c_K} &= \lambda x^{(\sigma \Rightarrow \tau) \times \sigma}.\text{fst}(x)\ \text{snd}(x)\end{aligned}$$

$$(A^{\sigma_2 \Rightarrow \sigma_3} \circ B^{\sigma_1 \Rightarrow \sigma_2})_{\lambda c_K} = \lambda x^{\sigma_1}.\lambda x^{\sigma_1}.A_{\lambda c_K}(B_{\lambda c_K} x)$$

$$(A^{\sigma \Rightarrow \tau} B^{\sigma})_{\lambda c_K} = A_{\lambda c_K} B_{\lambda c_K}$$

$$\langle A^{\sigma \Rightarrow \tau_1}, B^{\sigma \Rightarrow \tau_2} \rangle_{\lambda c_K} = \lambda x^{\sigma}.(A_{\lambda c_K} x, B_{\lambda c_K} x)$$

$$(A^{\sigma}, B^{\tau})_{\lambda c_K} = (A_{\lambda c_K}, B_{\lambda c_K})$$

$$\Lambda(A^{\sigma_1 \times \sigma_2 \Rightarrow \sigma_3})_{\lambda c_K} = \lambda x^{\sigma_1} y^{\sigma_2}.A_{\lambda c_K}(x, y).$$

Clearly $M^{\tau}_{\mathrm{CCL}_K}: \tau$ and $A^{\tau}_{\lambda c_K}: \tau$. We suppose that $x$, $y$ do not appear in $A$, $B$.

One may also define a typed version of $[\![M^{\tau}]\!]_{x_0,\ldots,x_n}$ (cf. exercises after 2.17) by taking the most general types of the terms $\mathrm{Subst}_i$:

$$\sigma \times \sigma_i \ldots \times \sigma_0 \times \sigma_i \Rightarrow \sigma \times \sigma_i \ldots \times \sigma_0.$$

This provides $[\![M]\!]_{x_0,\ldots,x_n}$ with the type $\sigma \times \sigma_n \cdots \times \sigma_0 \Rightarrow \tau$. However it must be noted that the two translations are not equivalent w.r.t. typing: for $[\![M]\!]$ we suppose that the set of variables of $M$ are among the $x_i$, while this condition relaxes to the set of free variables of $M$ in the case of De Bruijn's translation. Hence by choosing appropriate $x_0,\ldots, x_n$, $M_{\mathrm{DB}}$ may have a more general type than $[\![M]\!]$.

A careful reading of the proofs of 2.17 and 2.20 allows us to state the typed version of the syntactic equivalence theorem (for (3) the checking of the equations of *CCL* and *Quote*, for (4) the cases of applying and coupling have to be added (details omitted)).

4.4. THEOREM. *For any terms* $M, N \in \lambda\mathbf{c}_{\mathbf{K}}$, $A, B \in \mathbf{CCL}_{\mathbf{K}}$, *the following holds*:

(1) $M_{\mathrm{CCL}_K, \lambda c_K} =_{\beta P_K} M$

(2) $A_{\lambda c_K, \mathrm{CCL}_K} =_{AA_K} A$

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

(4) $M =_{\beta\eta SP_K} N \Rightarrow M_{\mathrm{CCL}_K} =_{AA_K} N_{\mathrm{CCL}_K}$.

We point out that $\eta$ is no longer needed in (1), where $\beta P_K$ is the typed version of $\beta P$ ($\eta$ was used only through 2.19). In fact the statement in 4.4 should be more precise and should include explicit sets of types, as pointed out in (Goguen and Meseguer, 1982), or even using more elaborate deduction systems allowing to extend first order equational reasoning to dependent types (Cartmell's (1978) generalized algebraic theories) or partial algebras (Obtutowicz, in press). For example, (1) should at least be restated as

$$M_{\mathrm{CCL}_K, \lambda c_K} \underset{FV(M)}{=}_{\beta P_K} M.$$

But the important point is that the same variables are involved in corresponding proofs of equalities in both calculi (see the exercise after 5.2 and the remark after 5.5). Corollaries 2.21 and 2.22 of the syntactic equivalence theorem also have their typed versions (details omitted).

*The Free Cartesian Closed Category*

In the rest of the section we establish the precise link between $\mathbf{CCL}_\mathbf{K}$ and the Cartesian closed categories. We introduce a purely equational setting for the definition of categorical constructions.

4.5. DEFINITION. Let $K$ be a set of **basic objects**. The types are now couples written $\sigma \to \tau$ of terms $\sigma, \tau$ of $T_{\times,\Rightarrow}(K \cup \{\varepsilon\})$, where $\varepsilon$, 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** $\mathbf{CCC}_\mathbf{K}$ is defined as follows:

if $x$ is a variable and $\sigma, \tau$ are objects then $x: \sigma \to \tau$ is a term

if $f: \sigma_2 \to \sigma_3$ and $g: \sigma_1 \to \sigma_2$ then $f \circ g: \sigma_1 \to \sigma_3$

$\mathrm{Id}: \sigma \to \sigma$

if $f: \sigma \to \tau_1$ and $g: \sigma \to \tau_2$ then $\langle f, g \rangle: \sigma \to \tau_1 \times \tau_2$

$\mathrm{Fst}: \sigma \times \tau \to \sigma$

$\mathrm{Snd}: \sigma \times \tau \to \tau$

$1: \sigma \to \varepsilon$

if $f: \sigma_1 \times \sigma_2 \to \sigma_3$ then $\Lambda(f): \sigma_1 \to (\sigma_2 \Rightarrow \sigma_3)$

$\mathrm{App}: (\sigma \Rightarrow \tau) \times \sigma \to \tau$.

We use as above the notation $\mathrm{Fst}^{\sigma,\tau}$, $\mathrm{Snd}^{\sigma,\tau}$, and $\mathrm{App}^{\sigma,\tau}$, and we also write $\mathrm{Id}^\sigma$ for $\mathrm{Id}: \sigma \to \sigma$ and $1^\sigma$ for $1: \sigma \to \varepsilon$. $CCC_K$ is the set of equations $CCL\beta\eta SP$ + Ter, where

$$\text{(Ter)} \qquad 1^\sigma = x^{\sigma \to \varepsilon},$$

and a typed version of $CCL\beta\eta SP$ is meant (as in 4.2, but with some $\Rightarrow$ replaced by $\to$ (details omitted)).

Here typing is critical since Ter without types would reduce to: "everything equals 0." The difference to the Definition 4.1 is the absence of applying and coupling operators, and the presence of a family of constants 1, the unique arrows to the terminal object.

Now we establish the equivalence of $\mathbf{CCL}_\mathbf{K}$, $AA_K$ and $\mathbf{CCC}_\mathbf{K}$, $CCC_K$. First we have to connect the types of both theories. We shall use the well-known isomorphism between $A \to B$ and $1 \to (A \Rightarrow B)$ in a Cartesian closed

category ($A$, $B$ are any objects, 1 is the terminal object), which is as follows (we represent 1 by $\varepsilon$ to stress the syntactic nature of objects in our setting):

$$(x^{\sigma\to\tau})^+ = \Lambda(x\circ \mathrm{Snd}^{\varepsilon,\sigma})$$
$$(x^{\varepsilon\to\sigma\Rightarrow\tau})^- = \mathrm{App}^{\sigma,\tau}\circ\langle x\circ 1^{\sigma\to\varepsilon}, \mathrm{Id}^\sigma\rangle.$$

One proves easily the following equations:

$$((x^{\sigma\to\tau})^+)^- =_{CCC_K} x \qquad \text{and} \qquad ((x^{\varepsilon\to\sigma\Rightarrow\tau})^-)^+ =_{CCC_K} x.$$

4.6. DEFINITION. With every object $\sigma$ we associate

$$\sigma^* \in T_{\times,\Rightarrow}(K)\cup\{\varepsilon\}$$
$$\sigma^-: \sigma\to\sigma^* \in \mathbf{CCC_K}$$
$$\sigma^+: \sigma^*\to\sigma \in \mathbf{CCC_K}$$

defined by ($^{+,-}$ concern objects, not terms as above):

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

For the product we proceed by cases:

— $\sigma_1^*, \sigma_2^* \neq \varepsilon$:
$$(\sigma_1\times\sigma_2)^* = \sigma_1^*\times\sigma_2^*$$
$$(\sigma_1\times\sigma_2)^+ = \langle\sigma_1^+\circ\mathrm{Fst}, \sigma_2^+\circ\mathrm{Snd}\rangle$$
$$(\sigma_1\times\sigma_2)^- = \langle\sigma_1^-\circ\mathrm{Fst}, \sigma_2^-\circ\mathrm{Snd}\rangle$$

— $\sigma_1^*\neq\varepsilon$, $\sigma_2^* = \varepsilon$:
$$(\sigma_1\times\sigma_2)^* = \sigma_1^*$$
$$(\sigma_1\times\sigma_2)^+ = \langle\mathrm{Id}, \sigma_2^+\circ 1^{\sigma_1}\rangle\circ\sigma_1^+$$
$$(\sigma_1\times\sigma_2)^- = \sigma_1^-\circ\mathrm{Fst}$$

— $\sigma_1^* = \varepsilon$, $\sigma_2^*\neq\varepsilon$: symmetric:
$$\sigma_1^*, \sigma_2^* = \varepsilon$$
$$(\sigma_1\times\sigma_2)^* = \varepsilon$$
$$(\sigma_1\times\sigma_2)^+ = \langle\sigma_1^+, \sigma_2^+\rangle$$
$$(\sigma_1\times\sigma_2)^- = 1.$$

Now the exponential:

— $\sigma_1^*, \sigma_2^*\neq\varepsilon$:
$$(\sigma_1\Rightarrow\sigma_2)^* = \sigma_1^*\Rightarrow\sigma_2^*$$
$$(\sigma_1\Rightarrow\sigma_2)^+ = \Lambda(\sigma_2^+\circ\mathrm{App}\circ\langle\mathrm{Fst}, \sigma_1^-\circ\mathrm{Snd}\rangle)$$
$$(\sigma_1\Rightarrow\sigma_2)^- = \Lambda(\sigma_2^-\circ\mathrm{App}\circ\langle\mathrm{Fst}, \sigma_1^+\circ\mathrm{Snd}\rangle)$$

— $\sigma_1^* = \varepsilon$, $\sigma_2^* \neq \varepsilon$:

$$(\sigma_1 \Rightarrow \sigma_2)^* = \sigma_2^*$$
$$(\sigma_1 \Rightarrow \sigma_2)^+ = \Lambda(\mathrm{Fst}) \circ \sigma_2^+$$
$$(\sigma_1 \Rightarrow \sigma_2)^- = \sigma_2^- \circ \mathrm{App} \circ \langle \mathrm{Id}, \sigma_1^+ \circ 1^{\sigma_1 \Rightarrow \sigma_2} \rangle$$

— $\sigma_2^* = \varepsilon$:

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

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

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

$$\sigma^+ \circ \sigma^- =_{CCC_K} \mathrm{Id}^\sigma \quad \text{and} \quad \sigma^- \circ \sigma^+ =_{CCC_K} \mathrm{Id}^{\sigma^*}.$$

*Proof.* By induction on $\sigma$, and by cases as in the definition,

$$\langle \sigma_1^+ \circ \mathrm{Fst}, \sigma_2^+ \circ \mathrm{Snd} \rangle \circ \langle \sigma_1^- \circ \mathrm{Fst}, \sigma_2^- \circ \mathrm{Snd} \rangle$$
$$= \langle \sigma_1^+ \circ \sigma_1^- \circ \mathrm{Fst}, \sigma_2^+ \circ \sigma_2^- \circ \mathrm{Snd} \rangle =_{\mathrm{ind}} \langle \mathrm{Fst}, \mathrm{Snd} \rangle = \mathrm{Id}$$

$$\langle \mathrm{Id}, \sigma_2^+ \circ 1^{\sigma_1} \rangle \circ \sigma_1^+ \circ \sigma_1^- \circ \mathrm{Fst}^{\sigma_1 \times \sigma_2} =_{\mathrm{ind,Ter}} \langle \mathrm{Fst}, \sigma_2^+ \circ 1^{\sigma_1 \times \sigma_2} \rangle$$
$$=_{\mathrm{Ter}} \langle \mathrm{Fst}, \sigma_2^+ \circ 1^{\sigma_2} \circ \mathrm{Snd}^{\sigma_1 \times \sigma_2} \rangle$$
$$=_{\mathrm{ind}} \langle \mathrm{Fst}, \mathrm{Snd} \rangle = \mathrm{Id}$$

$$\sigma_1^- \circ \mathrm{Fst} \circ \langle \mathrm{Id}, \sigma_2^+ \circ 1 \rangle \circ \sigma_1^+ = \sigma_1^- \circ \sigma_1^+ =_{\mathrm{ind}} \mathrm{Id}$$

$$\langle \sigma_1^+, \sigma_2^+ \rangle \circ 1^{\sigma_1 \times \sigma_2} = \langle \sigma_1^+ \circ 1^{\sigma_1} \circ \mathrm{Fst}, \sigma_2^+ \circ 1^{\sigma_2} \circ \mathrm{Snd} \rangle$$
$$=_{\mathrm{ind}} \langle \mathrm{Fst}, \mathrm{Snd} \rangle = \mathrm{Id}$$

$$1 \circ \langle (\sigma_1^+)^{\varepsilon \to \sigma_1}, \sigma_2^+ \rangle = 1^\varepsilon = \mathrm{Id}^\varepsilon.$$

Let $A = \sigma_2^- \circ \mathrm{App} \circ \langle \mathrm{Fst}, \sigma_1^+ \circ \mathrm{Snd} \rangle$

$$\Lambda(\sigma_2^+ \circ \mathrm{App} \circ \langle \mathrm{Fst}, \sigma_1^- \circ \mathrm{Snd} \rangle) \circ \Lambda(A)$$
$$= \Lambda(\sigma_2^+ \circ \mathrm{App} \circ \langle \Lambda(A) \circ \mathrm{Fst}, \sigma_1^- \circ \mathrm{Snd} \rangle)$$
$$= \Lambda(\sigma_2^+ \circ \sigma_2^- \circ \mathrm{App} \circ \langle \mathrm{Fst}, \sigma_1^+ \circ \mathrm{Snd} \rangle \circ \langle \mathrm{Fst}, \sigma_1^- \circ \mathrm{Snd} \rangle)$$
$$=_{\mathrm{ind}} \Lambda(\mathrm{App} \circ \langle \mathrm{Fst}, \sigma_1^+ \circ \sigma_1^- \circ \mathrm{Snd} \rangle) =_{\mathrm{ind}} \mathrm{Id}$$

$$\Lambda(\mathrm{Fst}) \circ \sigma_2^+ \circ \sigma_2^- \circ \mathrm{App} \circ \langle \mathrm{Id}, \sigma_1^+ \circ 1 \rangle$$
$$=_{\mathrm{ind}} \Lambda(\mathrm{App} \circ \langle \mathrm{Id}, \sigma_1^+ \circ 1 \rangle \circ \mathrm{Fst})$$
$$= \Lambda(\mathrm{App} \circ \mathrm{Id}) = \mathrm{Id} \qquad \text{(as in the second case)}$$

$$\sigma_2^- \circ \mathrm{App} \circ \langle \mathrm{Id}, \sigma_1^+ \circ 1 \rangle \circ \Lambda(\mathrm{Fst}) \circ \sigma_2^+ = \sigma_2^- \circ \mathrm{App} \circ \langle \Lambda(\mathrm{Fst}), \sigma_1^+ \circ 1 \rangle \circ \sigma_2^+$$
$$= \sigma_2^- \circ \mathrm{Fst} \circ \langle \mathrm{Id}, \ldots \rangle \circ \sigma_2^+ =_{\mathrm{ind}} \mathrm{Id}$$
$$\Lambda(\sigma_2^+ \circ 1^{\varepsilon \times \sigma_1}) \circ 1^{\sigma_1 \Rightarrow \sigma_2} = \Lambda(\sigma_2^+ \circ 1^{(\sigma_1 \Rightarrow \sigma_2) \times \sigma_1}) = \Lambda(\sigma_2^+ \circ 1^{\sigma_2} \circ \mathrm{App}) =_{\mathrm{ind}} \mathrm{Id}$$
$$1 \circ \Lambda(\sigma_2^+ \circ 1) = 1^{\varepsilon}.$$

Now we define the translations between $\mathbf{CCL_K}$ and $\mathbf{CCC_K}$. A good key to understand them is to think, in a category with terminal object 1, of arrows in $1 \to A$ as elements of the object $A$, and arrows $f$ in $A \to B$ as mapping an arrow $x$ in $1 \to A$ to the arrow $f \circ x$ in $1 \to B$.

4.8. DEFINITION. With any term $A: \sigma$ of $\mathbf{CCL_K}$ we associate a term $A_{\mathrm{CCC}_K}: \varepsilon \to \sigma$ of $\mathbf{CCC_K}$ defined as follows:

$$x^{\sigma}_{\mathrm{CCC}_K} = x^{\varepsilon \to \sigma}$$
$$A_{\mathrm{CCC}_K} = A^+, \qquad \text{if } A = \mathrm{Id}, \mathrm{Fst}, \mathrm{Snd}, \mathrm{App}$$
$$(A \circ B)_{\mathrm{CCC}_K} = (A^-_{\mathrm{CCC}_K} \circ B^-_{\mathrm{CCC}_K})^+$$
$$\langle A, B \rangle_{\mathrm{CCC}_K} = \langle A^-_{\mathrm{CCC}_K}, B^-_{\mathrm{CCC}_K} \rangle^+$$
$$\Lambda(A)_{\mathrm{CCC}_K} = \Lambda(A^-_{\mathrm{CCC}_K})^+$$
$$(AB)_{\mathrm{CCC}_K} = A^-_{\mathrm{CCC}_K} \circ B_{\mathrm{CCC}_K}$$
$$(A, B)_{\mathrm{CCC}_K} = \langle A_{\mathrm{CCC}_K}, B_{\mathrm{CCC}_K} \rangle.$$

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

$$x^{\sigma \to \tau}_{\mathrm{CCL}_K} = x^{(\sigma \Rightarrow \tau)^*}$$
$$\mathrm{Id}^{\sigma}_{\mathrm{CCL}_K} = \mathrm{Id}^{\sigma^* \Rightarrow \sigma^*}$$
$$\mathrm{Fst}^{\sigma,\tau}_{\mathrm{CCL}_K} = \mathrm{Fst}^{\sigma^*,\tau^*} \text{ if } \tau^* \neq \varepsilon, \qquad \mathrm{Id}^{\sigma^* \Rightarrow \sigma^*} \text{ if } \tau^* = \varepsilon.$$

Symmetrically for Snd,

$$\mathrm{App}^{\sigma,\tau}_{\mathrm{CCL}_K} = \mathrm{App}^{\sigma^*,\tau^*} \text{ if } \sigma^* \neq \varepsilon, \qquad \mathrm{Id}^{\tau^* \Rightarrow \tau^*} \text{ if } \sigma^* = \varepsilon$$
$$(f^{\sigma_2 \to \sigma_3} \circ g^{\sigma_1 \to \sigma_2})_{\mathrm{CCL}_K} = f_{\mathrm{CCL}_K} \circ g_{\mathrm{CCL}_K} \qquad \text{if } \sigma_1^*, \sigma_2^* \neq \varepsilon$$
$$= f_{\mathrm{CCL}_K} g_{\mathrm{CCL}_K} \qquad \text{if } \sigma_1^* = \varepsilon, \sigma_2^* \neq \varepsilon$$
$$= \Lambda(\mathrm{Fst}^{\sigma_3^*,\sigma_1^*}) f_{\mathrm{CCL}_K} \qquad \text{if } \sigma_1^* \neq \varepsilon, \sigma_2^* = \varepsilon$$
$$= f_{\mathrm{CCL}_K} \qquad \text{if } \sigma_1^*, \sigma_2^* = \varepsilon$$

$$\begin{aligned}
\langle f^{\sigma\to\tau_1}, g^{\sigma\to\tau_2}\rangle_{\mathrm{CCL}_K} &= \langle f_{\mathrm{CCL}_K}, g_{\mathrm{CCL}_K}\rangle && \text{if } \sigma^*, \tau_1^*, \tau_2^* \neq \varepsilon\\
&= (f_{\mathrm{CCL}_K}, g_{\mathrm{CCL}_K}) && \text{if } \sigma^* = \varepsilon, \tau_1^*, \tau_2^* \neq \varepsilon\\
&= f_{\mathrm{CCL}_K} && \text{if } \tau_1^* \neq \varepsilon, \tau_2^* = \varepsilon\\
&= g_{\mathrm{CCL}_K} && \text{if } \tau_1^* = \varepsilon, \tau_2^* \neq \varepsilon\\
\Lambda(f^{\sigma_1\times\sigma_2\to\sigma_3})_{\mathrm{CCL}_K} &= \Lambda(f_{\mathrm{CCL}_K}) && \text{if } \sigma_1^*, \sigma_2^* \neq \varepsilon\\
&= f_{\mathrm{CCL}_K} && \text{if } \sigma_1^* = \varepsilon \text{ or } \sigma_2^* = \varepsilon.
\end{aligned}$$

Now we may state the equivalence theorem.

4.9. THEOREM. *For all terms $A, B$ of* $\mathbf{CCL_K}$ *and $f, g$ of* $\mathbf{CCC_K}$ *of appropriate types, the following holds*:

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

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

(3) $A_{\mathrm{CCC}_K,\mathrm{CCL}_K} =_{\mathrm{CCC}_K} A$

(4) $f^{\sigma\to\tau}_{\mathrm{CCL}_K,\mathrm{CCC}_K} =_{AA_K} \overline{\sigma\to\tau}(f[x_0 \leftarrow \underline{\sigma\to\tau}(x_0),\ldots, x_n \leftarrow \underline{\sigma\to\tau}(x_n)])$

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

$$\begin{aligned}
\overline{\sigma\to\tau}(f^{\sigma\to\tau}) &= (\tau^- \circ f \circ \sigma^+)^+ && \text{if } \sigma^* \neq \varepsilon, \tau^- \circ f \circ \sigma^+ \text{ if } \sigma^* = \varepsilon\\
\underline{\sigma\to\tau}(g^{\varepsilon\to(\sigma\Rightarrow\tau)^*}) &= \tau^+ \circ f^- \circ \sigma^- && \text{if } \sigma^* \neq \varepsilon, \tau^+ \circ f \circ \sigma^- \text{ if } \sigma^* = \varepsilon.
\end{aligned}$$

*Proof.* First (1): $((x \circ y) \circ z)_{\mathrm{CCC}_K} = (x^- \circ y^- \circ z^-)^+ = (x \circ (y \circ z))_{\mathrm{CCC}_K}$

$$(\mathrm{Id} \circ x)_{\mathrm{CCC}_K} = (\mathrm{Id} \circ x^-)^+ = (x^-)^+ = x.$$

All the other equations are proved likewise, except app, $\mathrm{Quote}_1$, and $\mathrm{Quote}_2$:

$$\begin{aligned}
(xy)_{\mathrm{CCC}_K} &= (x^{\varepsilon\to(\sigma\Rightarrow\tau)})^- \circ y^{\varepsilon\to\sigma} =_{\mathrm{def}} \mathrm{App} \circ \langle x \circ 1, \mathrm{Id}\rangle \circ y\\
&= \mathrm{App} \circ \langle x \circ 1 \circ y^{\varepsilon\to\sigma}, y\rangle\\
&= \mathrm{App} \circ \langle x \circ 1^{\varepsilon}, y\rangle = \mathrm{App} \circ \langle x \circ \mathrm{Id}^{\varepsilon}, y\rangle\\
&= \mathrm{App} \circ \langle x, y\rangle = (\mathrm{App}(x, y))_{CCC_K}.
\end{aligned}$$

For Quote we first establish $\Lambda(\mathrm{Fst}^{\sigma,\tau}) \circ x^{\varepsilon,\sigma} = (x \circ 1^{\tau})^+$:

$$\begin{aligned}
\Lambda(\mathrm{Fst}^{\sigma,\tau}) \circ x^{\varepsilon,\sigma} &= \Lambda(x \circ \mathrm{Fst}^{\varepsilon,\tau}) = \Lambda(x \circ 1^{\varepsilon,\tau}) = \Lambda(x \circ 1^{\tau} \circ \mathrm{Snd}^{\varepsilon,\tau}) = (x \circ 1^{\tau})^+\\
(\Lambda(\mathrm{Fst}^{\sigma,\tau})\, x^{\sigma} \circ y^{\tau_1\Rightarrow\tau})_{\mathrm{CCC}_K} &= ((\Lambda(\mathrm{Fst}^{\sigma,\tau}) \circ x^{\varepsilon\to\sigma})^- \circ (y^-)^{\tau_1\to\tau})^+\\
&= (x \circ 1^{\tau} \circ (y^-)^{\tau_1\to\tau})^+\\
&= (x \circ 1^{\tau_1})^+ =_{\mathrm{def}} (\Lambda(\mathrm{Fst}^{\sigma,\tau_1})\, x)_{\mathrm{CCC}_K}
\end{aligned}$$

$$(\mathrm{App} \circ \langle x \circ \Lambda(\mathrm{Fst})\ y, z^{\sigma_1 \Rightarrow \sigma_2} \rangle)_{CCC_K} = (\mathrm{App} \circ \langle x^- \circ y \circ 1^{\sigma_1}, (z^-)^{\sigma_1 \to \sigma_2} \rangle)^+$$
$$= (\mathrm{App} \circ \langle x^- \circ y \circ 1^{\sigma_2}, \mathrm{Id} \rangle \circ z^-)^+$$
$$=_{\mathrm{def}} ((x^- \circ y)^- \circ z^-)^+ = (xy \circ z)_{CCC_K}.$$

We prove (2): The equations of $CCC_K$ are checked by cases on the definition of $_{CCL_K}$. We omit the cases where the members of the equations have a type $\sigma \to \tau$ with $\tau^* = \varepsilon$, since one can easily observe that the translation of $f: \sigma_1 \to \tau_1$ does not depend on its possible subterms of type $\sigma \to \tau$ where $\tau^* = \varepsilon$. For instance if

$$f = f_1 \circ f_2 \qquad \text{with } f_2 = \mathrm{Id}^{\varepsilon} \circ x^{\sigma \to \varepsilon},$$
$$f' = f_1 \circ f'_2 \qquad \text{with } f'_2 = x^{\sigma \to \varepsilon},$$

then by definition

$$f_{\mathrm{CCL}_K} = (\Lambda(Fst)(f_1)_{\mathrm{CCL}_K} \text{ or } (f_1)_{\mathrm{CCL}_K}) = f'_{\mathrm{CCL}_K}$$

(and likewise for $f = \langle f_1, f_2 \rangle$). For the same reason we need not check Ter. We detail only Ass: Set $((x^{\sigma_3 \to \sigma_4}) \circ (z^{\sigma_1 \to \sigma_2}))_{CCL_K} = A$:

$(\sigma_1^*, \sigma_2^* \neq \varepsilon)\ A =_{\mathrm{def}} (x \circ y)_{\mathrm{CCL}_K} \circ z = B$

$(\sigma_3^* \neq \varepsilon)\ B =_{\mathrm{def}} (x \circ y) \circ z =_{\mathrm{Ass,def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_3^* = \varepsilon)\ B =_{\mathrm{def}} (\Lambda(Fst)\ x) \circ z =_{\mathrm{Quote_1,def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_1^* = \varepsilon, \sigma_2^* \neq \varepsilon)\ A =_{\mathrm{def}} (x \circ y)_{\mathrm{CCL}_K} z = B$

$(\sigma_3^* \neq \varepsilon)\ B =_{\mathrm{def}} (x \circ y)\ z =_{\mathrm{ass,def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_3^* = \varepsilon)\ B =_{\mathrm{def}} (\Lambda(\mathrm{Fst})\ x)\ z =_{d\Lambda,\mathrm{fst}} x =_{\mathrm{def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_1^* \neq \varepsilon, \sigma_2^* = \varepsilon)\ A =_{\mathrm{def}} \Lambda(\mathrm{Fst})((x \circ y)_{\mathrm{CCL}_K}) = B$

$(\sigma_3^* \neq \varepsilon)\ B =_{\mathrm{def}} \Lambda(\mathrm{Fst})(xy) =_{\mathrm{Quote_2\ ,def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_3^* = \varepsilon)\ B =_{\mathrm{def}} \Lambda(\mathrm{Fst})\ x =_{\mathrm{def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_1^*, \sigma_2^* = \varepsilon)\ A =_{\mathrm{def}} (x \circ y)_{\mathrm{CCL}_K} = B$

$(\sigma_3^* \neq \varepsilon)\ B =_{\mathrm{def}} xy =_{\mathrm{def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}$

$(\sigma_3^* = \varepsilon)\ B =_{\mathrm{def}} x =_{\mathrm{def}} (x \circ (y \circ z))_{\mathrm{CCL}_K}.$

For (3) we first establish for all $\sigma^*, \tau^* \neq \varepsilon$:

$$(x^{\sigma \to \tau})^+_{\mathrm{CCL}_K} =_{CCL_K} x_{\mathrm{CCL}_K}$$

whence we derive by (2)

$$(x^{\varepsilon \to \sigma \Rightarrow \tau})^{-}_{\mathrm{CCL}_K} = CCL_K\, x_{\mathrm{CCL}_K}$$

(one has $(x^{-})^{+}_{\mathrm{CCL}_K} = CCL_K\, x_{\mathrm{CCL}_K}$)

$$\begin{aligned} x^{\sigma \Rightarrow \tau}_{\mathrm{CCL}_K} &= \Lambda(x \circ \mathrm{Snd}^{\varepsilon,\sigma})_{\mathrm{CCL}_K} = (x \circ \mathrm{Snd}^{\varepsilon,\sigma})_{\mathrm{CCL}_K} \\ &= x \circ \mathrm{Snd}^{\varepsilon,\sigma}_{\mathrm{CCL}_K} = x \circ \mathrm{Id} = x. \end{aligned}$$

Now (3) is easy to check:

$$\begin{aligned} x^{\sigma}_{\mathrm{CCC}_K,\mathrm{CCL}_K} &= x^{\varepsilon \Rightarrow \sigma}_{\mathrm{CCL}_K} = x^{\sigma} \\ \mathrm{Id}^{\sigma \Rightarrow \sigma}_{\mathrm{CCC}_K,\mathrm{CCL}_K} &= (\mathrm{Id}^{\sigma})^{+}_{\mathrm{CCL}_K} = \mathrm{Id}^{\sigma}_{\mathrm{CCL}_K} = \mathrm{Id}. \end{aligned}$$

Likewise for Fst, Snd, and App

$$\begin{aligned} (x^{\sigma_2 \Rightarrow \sigma_3} \circ y^{\sigma_1 \Rightarrow \sigma_2})_{\mathrm{CCC}_K,\mathrm{CCL}_K} &= ((x^{-})^{\sigma_2 \to \sigma_3} \circ (y^{-})^{\sigma_1 \to \sigma_2})^{+}_{\mathrm{CCL}_K} \\ &= (x^{\varepsilon \to \sigma_2 \Rightarrow \sigma_3})^{-}_{\mathrm{CCL}_K} \circ y^{-}_{\mathrm{CCL}_K} = x^{\varepsilon \to \sigma_2 \Rightarrow \sigma_3}_{\mathrm{CCL}_K} \circ y_{\mathrm{CCL}_K} \\ &= x^{\sigma_2 \Rightarrow \sigma_3} \circ y. \end{aligned}$$

Likewise for $\langle\ \rangle$ and $\Lambda$

$$\begin{aligned} (x^{\sigma \Rightarrow \tau} y)_{\mathrm{CCC}_K,\mathrm{CCL}_K} &= ((x^{-})^{\sigma \to \tau} \circ y^{\varepsilon \to \sigma})_{\mathrm{CCL}_K} = (x^{\varepsilon \to \sigma \Rightarrow \tau})^{-}_{\mathrm{CCL}_K} y^{\varepsilon \to \sigma}_{\mathrm{CCL}_K} \\ &= x^{\varepsilon \to \sigma \Rightarrow \tau}_{\mathrm{CCL}_K} y^{\sigma} = x^{\sigma \Rightarrow \tau} y. \end{aligned}$$

Likewise for ( ). Finally we prove (4) by cases as (2). We detail only currying:

$$\Lambda(f^{\sigma_1 \times \sigma_2 \to \sigma_3})_{\mathrm{CCL}_K,\mathrm{CCC}_K} = A$$

— $(\sigma_1^{*}, \sigma_2^{*} \neq \varepsilon)\ A = \Lambda(f^{-}_{\mathrm{CCL}_K,\mathrm{CCC}_K})^{+} = \Lambda(\sigma_3^{-} \circ f' \circ (\sigma_1 \times \sigma_2)^{+})^{+}$.

We have to check $(\sigma_2 \Rightarrow \sigma_3)^{-} \circ \Lambda(f') \circ \sigma_1^{+} = \Lambda(\sigma_3^{-} \circ f' \circ (\sigma_1 \times \sigma_2)^{+})$:

$$\begin{aligned} &(\sigma_2 \Rightarrow \sigma_3)^{-} \circ \Lambda(f') \circ \sigma_1^{+} \\ &\quad =_{\mathrm{def}} \Lambda(\sigma_3^{-} \circ \mathrm{App} \circ \langle \mathrm{Fst}, \sigma_2^{+} \circ \mathrm{Snd} \rangle) \circ \Lambda(f') \circ \sigma_1^{+} \\ &\quad = \Lambda(\sigma_3^{-} \circ \mathrm{App} \circ \langle \mathrm{Fst}, \sigma_2^{+} \circ \mathrm{Snd} \rangle \circ \langle \Lambda(f') \circ \sigma_1^{+} \circ \mathrm{Fst}, \mathrm{Snd} \rangle) \\ &\quad = \Lambda(\sigma_3^{-} \circ \mathrm{App} \circ \langle \Lambda(f') \circ \sigma_1^{+} \circ \mathrm{Fst}, \sigma_2^{+} \circ \mathrm{Snd} \rangle) \\ &\quad =_{\mathrm{def}} \Lambda(\sigma_3^{-} \circ f' \circ (\sigma_1 \times \sigma_2)^{+}) \end{aligned}$$

$$
\begin{aligned}
-\ (\sigma_1^* = \varepsilon,\ \sigma_2^* \neq \varepsilon)\ A &= f_{\mathrm{CCL}_K,\mathrm{CCC}_K} =_{\mathrm{ind}} (\sigma_3^- \circ f' \circ (\sigma_1 \times \sigma_2)^+)^+ \\
&=_{\mathrm{def}} \Lambda(\sigma_3^- \circ f' \circ \langle \sigma_1^+ \circ 1^{\sigma_2^*} \circ \mathrm{Snd}^{\varepsilon,\sigma_2^*},\ \sigma_2^+ \circ \mathrm{Snd} \rangle \\
&= \Lambda(\sigma_3^- \circ f' \circ \langle \sigma_1^+ \circ \mathrm{Fst},\ \sigma_2^+ \circ \mathrm{Snd} \rangle) \\
&= (\sigma_2 \Rightarrow \sigma_3)^- \circ \Lambda(f') \circ \sigma_1^+ \qquad \text{(cf. case before).}
\end{aligned}
$$

$$
-\ (\sigma_1^* \neq \varepsilon,\ \sigma_2^* = \varepsilon)\ A = f_{\mathrm{CCL}_K,\mathrm{CCC}_K} = (\sigma_3^- \circ f' \circ (\sigma_1 \times \sigma_2)^+)^+.
$$

We have to check $(\sigma_2 \Rightarrow \sigma_3)^- \circ \Lambda(f') \circ \sigma_1^+ = \sigma_3^- \circ f' \circ (\sigma_1 \times \sigma_2)^+$:

$$
\begin{aligned}
(\sigma_2 \Rightarrow \sigma_3)^- \circ \Lambda(f') \circ \sigma_1^+ &= \sigma_3^- \circ \mathrm{App} \circ \langle \mathrm{Id},\ \sigma_2^+ \circ 1 \rangle \circ \Lambda(f') \circ \sigma_1^+ \\
&= \sigma_3^- \circ \mathrm{App} \circ \langle \Lambda(f') \circ \sigma_1^+,\ \sigma_2^+ \circ 1 \rangle \\
&= \sigma_3^- \circ f' \circ (\sigma_1 \times \sigma_2)^+
\end{aligned}
$$

$$
\begin{aligned}
-\ (\sigma_1^*, \sigma_2^* = \varepsilon)\ A &= f_{\mathrm{CCL}_K,\mathrm{CCC}_K} = \sigma_3^- \circ f' \circ (\sigma_1 \times \sigma_2)^+ \\
&= \sigma_3^- \circ f' \circ \langle \sigma_1^+, \sigma_2^+ \rangle = \sigma_3^- \circ f' \circ \langle \sigma_1^+, \sigma_2^+ \circ 1^{\varepsilon} \rangle \\
&= (\sigma_2 \Rightarrow \sigma_3)^- \circ \Lambda(f') \circ \sigma_1^+ ;
\end{aligned}
$$

$f'$, $g'$ are abbreviations for the result of suitable substitutions on $f$, $g$.

*Application to a Known Decision Problem*

We end the section by pointing out that the two equivalence theorems of the section may be used to decide the equality in $\mathbf{CCC_K}$ (and also in $\mathbf{CCL_K}$). Indeed the rewriting system obtained by orientating the equations of $\beta\eta SP_K$ from left to right is confluent (cf. Pottinger, 1979) and noetherian. We refer to Lambek and Scott (1985) for a proof of that property, based on Tait's computability method. We simply note

$$
f^{\sigma \to \tau} =_{\mathrm{CCC}_K} g^{\sigma \to \tau} \qquad \text{iff } f_{\mathrm{CCL}_K} =_{AA_K} g_{\mathrm{CCL}_K} \text{ iff } f_{\mathrm{CCL}_K,\lambda c_K} =_{\beta\eta SP_K} g_{\mathrm{CCL}_K,\lambda c_K}
$$

using

$$
\overline{\sigma \to \tau}(\underline{\sigma \to \tau}(x)) =_{\mathrm{CCC}_K} x \qquad \text{and} \qquad \underline{\sigma \to \tau}(\overline{\sigma \to \tau}(x)) =_{\mathrm{CCC}_K} x.
$$

The very same kind of problem was solved in (Szabo, 1978), using cut elimination techniques.

## 5. Models of the $\lambda$-Calculus

Our syntactic equivalence theorems have semantic counterparts, which guided the definition of $\lambda$-calculus models given in this section. Our definition corresponds to $\lambda$-algebras (Barendregt, 1984; Koymans, 1982; Meyer, 1982), but we are primarily interested in $\beta\eta SP$ rather than only $\beta$.

Formal definitions of $\lambda$-calculus models were first dealt with extensively and compared in Hindley and Longo (1980).

We shortly review equivalent definitions of models which have been given for $\beta$ only: functional and combinatory models (Meyer, 1982), and diverse categorical characterizations, similar to ours (Koymans, 1982, 1984; Obtułowicz, 1982, in press; Longo and Moggi, 1984).

An important observation is that the structures defined in the section are meaningful only when the underlying sets contain at least two elements.

*The Model Definition*

We formulate our model definition in a way which makes it very similar to the definition of models of equational presentations. We derive easily a semantic equivalence theorem from the syntactic equivalence theorem of Section 2. We briefly discuss an interesting congruence on terms induced by a model.

5.1. DEFINITION. A $\beta\eta SP$-**model** M of the $\lambda c$-calculus is a set $|\mathrm{M}|$ together with a **semantic function** associating with any term $M$ (modulo $\alpha$-equivalence) and any environment $\rho$ defined on $FV(M)$ (which will be always assumed implicitly) an element of $|\mathrm{M}|$ denoted by

$$[\![M, \rho]\!]_{\mathrm{M}} \qquad \text{or} \qquad [\![M]\!]_{\mathrm{M}}\,\rho$$

(the second notation is more common, but the first one is clearer for foundations). The semantic function is assumed to verify the four following axioms:

(1) $[\![x, \rho]\!]_{\mathrm{M}} = \rho(x)$

(2) $M =_{\beta\eta SP} N \Rightarrow [\![M, \rho]\!]_{\mathrm{M}} = [\![N, \rho]\!]_{\mathrm{M}}$

(3) $(\forall x \in FV(M), \rho(x) = \nu(x)) \Rightarrow [\![M, \rho]\!]_{\mathrm{M}} = [\![M, \nu]\!]_{\mathrm{M}}$

(4) $[\![M[x_1 \leftarrow N_1, \ldots, x_n \leftarrow N_n], \rho]\!]_{\mathrm{M}}$
$= [\![M, \rho[x_1 \leftarrow [\![N_1, \rho]\!]_{\mathrm{M}}, \ldots, x_n \leftarrow [\![N_n, \rho]\!]_{\mathrm{M}}]]\!]_{\mathrm{M}}$.

Sometimes, for a closed term $M$, we shall write $[\![M]\!]$, since the semantics does not depend on $\rho$ by (2).

One defines the $\beta\eta P$-models, the $\beta\eta$-models, the $\beta$-models of $\lambda\mathbf{c}$ (or of $\lambda$ in the two last cases) by replacing $\beta\eta SP$ by $\beta\eta P$, $\beta\eta$ or $\beta$ in the condition (2). The $\beta$-models of $\lambda$ will also be called $\lambda$-**algebras**, according to Barendregt (1984). When we shall state a property which is true of all these kinds of models, we shall simply talk of models without more precision. Here are two such properties.

A model M is called **weakly extensional** if for all terms $M, N$, any environment $\rho$:

$$(\forall d \in |\mathrm{M}|, \quad [\![M, \rho[x \leftarrow d]]\!]_{\mathrm{M}} = [\![N, \rho[x \leftarrow d]]\!]_{\mathrm{M}} \Rightarrow [\![\lambda x.M, \rho]\!]_{\mathrm{M}} = [\![\lambda x.N, \rho]\!]_{\mathrm{M}}.$$

One defines an **application** operation $(d_1, d_2) \mapsto (d_1 d_2)_{\mathrm{M}}$ by

$$(d_1 d_2)_{\mathrm{M}} = [\![xy, \{x = d_1, y = d_2\}]\!]_{\mathrm{M}}.$$

A model M is called **extensional** if for all $d_1, d_2 \in |\mathrm{M}|$

$$(\forall d \in |\mathrm{M}|, \quad (d_1 d)_{\mathrm{M}} = (d_2 d)_{\mathrm{M}}) \Rightarrow d_1 = d_2.$$

The weakly extensional $\lambda$-algebras are called $\lambda$-**models**. A **homomorphism** from a model M to a model N is a function $f$ from $|\mathrm{M}|$ to $|\mathrm{N}|$ s.t.

$$\forall M, \rho, f([\![M, \rho]\!]_{\mathrm{M}}) = [\![M, f \circ \rho]\!]_{\mathrm{N}}.$$

On the categorical side a model of **CCL**, *CCLβSP* will be called a *βηSP*-**monoid**.

Notice that our axioms are "copies" of properties holding in an equational theory. *βηSP*-monoids are also considered in Lambek and Scott (1985), where they are called *C*-monoids. The definition above applies also to calculi with constants. If $C$ is a constant, it has a value $c = [\![C, \rho]\!]_{\mathrm{M}}$ in the model, independent of $\rho$ by (3). The composition and identity determine all the other operators in a *βηSP*-monoid:

EXERCISE. Let M, M′ be *βηSP*-monoids: show that a bijection $\iota$: $|\mathrm{M}| \mapsto |\mathrm{M}'|$ is an isomorphism iff $\iota$ is a morphism for the monoid structure of M, M′ w.r.t. composition. (Hint: show that once the interpretation of the composition is given, there is only one possible interpretation for the other operators in a *βηSP*-monoid (for Fst consider Fst $\circ \langle$Fst, Snd$\rangle$,...).)

Examples of models which are not weakly extensional are the model of sequential algorithms (Berry and Curien, 1982; Curien, 1985a), and the interior (i.e., the set of elements which are the meaning of some closed term) of, say, Scott's model *Pω* (Hindley and Longo, 1980). *Pω* is an example of a weakly extensional, but not extensional model (Scott, 1976).

We first show how the translations from $\lambda$**c** to **CCL** and from **CCL** to $\lambda$**c** induce naturally *βηSP*-monoids from *βηSP*-models of $\lambda$**c**, and *βηSP*-models of $\lambda$**c** from *βηSP*-monoids. # denotes the extension of an environment by initiality.

5.2. DEFINITION. Let M be a $\beta\eta SP$-monoid. One defines $\mathrm{M}_{\lambda c}$ by ($M$ is any term of $\lambda\mathbf{c}$):

$$|\mathrm{M}_{\lambda c}| = |\mathrm{M}|$$
$$[\![M, \rho]\!]_{\mathrm{M}_{\lambda c}} = \rho^{\#}_{\mathrm{M}}(M_{\mathrm{CCL}}).$$

Let N be a $\beta\eta SP$-model of $\lambda\mathbf{c}$. One defines $\mathrm{N}_{\mathrm{CCL}}$ by ($f$ is any operator of **CCL**, with arity $n$):

$$|\mathrm{N}_{\mathrm{CCL}}| = |\mathrm{N}|$$
$$f_{\mathrm{N}_{\mathrm{CCL}}}(d_1,..., d_n) = [\![f(x_1,..., x_n)_{\lambda c}, \{x_1 = d_1,..., x_n = d_n\}]\!]_{\mathrm{N}}.$$

To justify the definition, we have to show the following property, which we leave as an

EXERCISE. Show that for any term $M \in \lambda\mathbf{c}$ s.t. $FV(M) \subseteq \{x_0,..., x_n\}$, there exists a term $A \in \mathbf{CCL}$ s.t. $V(A) \subseteq \{x_0,..., x_n\}$ and

$$M_{\mathrm{CCL}} =_{\beta\eta SP} A$$

(Hint: show by ext that the property is equivalent to the one asserting the existence of a closed term $B$ s.t.

$$A^{n+1}(M_{\mathrm{DB}(x_0,..., x_n)}) = A(Fst)\, B;$$

then use the first form for application, the second for abstraction in the induction steps).

The next lemma states that the translation $_{\mathrm{CCL}}$ behaves well w.r.t. substitutions.

5.3. LEMMA. *For all terms $M, N_1,..., N_n$ and variables $x_1,..., x_n$, the following holds*:

$$M[x_1 \leftarrow N_1,..., x_n \leftarrow N_n]_{\mathrm{CCL}}$$
$$=_{CCL\beta\eta SP} M_{\mathrm{CCL}}[x_1 \leftarrow (N_1)_{\mathrm{CCL}},..., x_n \leftarrow (N_n)_{\mathrm{CCL}}].$$

*Proof.* By induction on the structure of $N$. The case of the application results from the following property, easy to check:

$$(M_1 M_2)_{\mathrm{CCL}} =_{CCL\beta\eta SP} (M_1)_{\mathrm{CCL}}\, (M_2)_{\mathrm{CCL}}.$$

Likewise for the couple and the projections. We are left with the abstraction $\lambda y.M$. We state the induction hypothesis in an appropriate way

(assuming $FV(M[x_1 \leftarrow N_1,..., x_n \leftarrow N_n]) \subseteq \{v_1,..., v_q\}$ and $FV(M) \subseteq \{x_1,..., x_n, u_1,..., u_p\}$):

$$\begin{aligned} M[x_1 \leftarrow N_1,..., x_n \leftarrow N_n]_{\mathrm{DB}(y,v_1,...,v_q)}(z, v_q,..., v_1, y) \\ =_{\mathrm{def}} M[x_1 \leftarrow N_1,..., x_n \leftarrow N_n]_{\mathrm{CCL}} \\ =_{\mathrm{ind}} M_{\mathrm{CCL}}[x_1 \leftarrow (N_1)_{\mathrm{CCL}},..., x_n \leftarrow (N_n)_{\mathrm{CCL}}] \\ =_{\mathrm{def}} M_{\mathrm{DB}(y,x_1,...,x_n,u_1,...,u_p)}(z, u_p,..., u_1, (N_n)_{\mathrm{CCL}},..., (N_1)_{\mathrm{CCL}}, y). \end{aligned}$$

We write

$$\begin{aligned} M_{\mathrm{DB}(y,x_1,...,x_n,u_1,...,u_p)} &= A \\ M[x_1 \leftarrow N_1,..., x_n \leftarrow N_n]_{\mathrm{DB}(y,v_1,...,v_q)} &= B \\ (z, u_p,..., u_1, (N_n)_{\mathrm{CCL}},..., (N_1)_{\mathrm{CCL}}) &= C \\ (z, v_q,..., v_1) &= D. \end{aligned}$$

The induction hypothesis becomes

$$A(C, y) =_{CCL\beta\eta SP} B(D, y).$$

Now we compute the two members of the desired equality (we omit the case where $y$ is one of the $x_i$'s: the argument is the same with the remaining $x_i$'s):

$$\begin{aligned} (\lambda y.M[x_1 \leftarrow N_1,..., x_n \leftarrow N_n])_{CCL} &=_{\mathrm{def}} \Lambda(B)\, D \\ (\lambda y.M)_{CCL}[x_1 \leftarrow (N_1)_{CCL},..., x_n \leftarrow (N_n)_{CCL}] &=_{\mathrm{def}} \Lambda(A)\, C. \end{aligned}$$

We are left to prove

$$\Lambda(A)\, C =_{CCL\beta\eta SP} \Lambda(B)\, D$$

which by ext (cf. 2.25) results from

$$\Lambda(A)\, Cy = A(C, y) =_{\mathrm{ind}} B(D, y) = \Lambda(B)\, Dy.$$

Now we can state the semantic equivalence theorem.

5.4. SEMANTIC EQUIVALENCE THEOREM. — *For any* $\beta\eta SP$*-monoid* $\mathbf{M}$, $\mathbf{M}_{\lambda c}$ *is a* $\beta\eta SP$*-model, and* $\mathbf{M}_{\lambda c,\mathrm{CCL}} = \mathbf{M}$

— *For any* $\beta\eta SP$*-model* $\mathbf{N}$ *of* $\lambda\mathbf{c}$, $\mathbf{N}_{\mathrm{CCL}}$ *is a* $\beta\eta SP$*-monoid* s.t.

$$\forall A \in \mathbf{CCL},\ \rho^{\#}_{\mathbf{N}_{\mathrm{CCL}}}(A) = [\![A_{\lambda c}, \rho]\!]_{\mathbf{N}} \quad \textit{and} \quad \mathbf{N}_{\mathrm{CCL},\lambda c} = \mathbf{N}.$$

*Proof.* For $M_{\lambda c}$ the conditions (1), (2), and (3) of 5.1 are trivial consequences of the definition of $M_{\lambda c}$, while condition (4) results from the previous lemma and from the universal algebra property which has inspired (4).

For $N_{CCL}$, the equality of the theorem is proved easily using (4), and as a consequence the equations of $CCL\beta\eta SP$ are valid, using (2). Then we check $N_{CCL,\lambda c} = N$,

$$[\![M, \rho]\!]_{N_{CCL,\lambda c}} = \rho^{\#}_{N_{CCL}}(M_{CCL}) = [\![M_{CCL,\lambda c}, \rho]\!]_N = [\![M, \rho]\!]_N.$$

Finally we check likewise $M_{\lambda c,CCL} = M$,

$$f_{M_{\lambda c,CCL}}(d_1,..., d_n) = [\![f(x_1,..., x_n)_{\lambda c}, \{x_1 = d_1,..., x_n = d_n\}]\!]_{M_{\lambda c}}$$
$$= \{x_1 = d_1,..., x_n = d_n\}^{\#}_M (f(x_1,..., x_n)_{\lambda c,CCL}) = f_M(d_1,..., d_n).$$

The theorem easily extends to an equivalence between the categories with models, homomorphisms as objects, arrows. The situation is exactly the same in the typed case, where the models are defined as follows:

5.5. DEFINITION. A model M of the typed $\lambda$-calculus $\lambda_K$ is given by a family of sets $M_\sigma$ for all types $\sigma$, and by a semantic function $(M^\sigma, \rho) \mapsto [\![M, \rho]\!]_M \in M^\sigma$ verifying the axioms of the Definition 5.1. The models of $CCL_K$, $AA_K$ are called **applicative algebras**.

The same proofs yield a typed version of the semantic equivalence theorem. We stress that this rather nice transfer is simplified by the property that *essentially for all terms A, M, $A_{\lambda c}$, and $M_{CCL}$ have the same variables as A, M, respectively* (cf. the exercise after 5.2 and the remark after 4.4.).

Now we come back to the model definition. The next lemma allows us to connect Definition 5.1 to more usual ones (Hindley and Longo, 1980; Barendregt, 1984; Koymans, 1984).

5.6. LEMMA. *The following axiom*

(4′) *There are operations* $(d_1, d_2) \mapsto (d_1 d_2)_M$, $(d_1, d_2) \mapsto (d_1, d_2)_M$, $d \mapsto fst_M(d)$, *and* $d \mapsto snd_M(d)$ s.t.

$$[\![MN, \rho]\!]_M = ([\![M, \rho]\!]_M\ [\![N, \rho]\!]_M)_M$$
$$([\![\lambda x.M, \rho]\!]_M\ d)_M = [\![M, \rho[x \leftarrow d]]\!]_M$$
$$[\![(M, N), \rho]\!]_M = ([\![M, \rho]\!]_M, [\![N, \rho]\!]_M)_M$$
$$[\![\mathrm{fst}(M), \rho]\!]_M = \mathrm{fst}_M([\![M, \rho]\!]_M)$$
$$[\![\mathrm{snd}(M), \rho]\!]_M = \mathrm{snd}_M([\![M, \rho]\!]_M)$$

*may replace the axiom* (4) *of* 5.1. *It may also replace the axiom* (2) *of* 5.1 *in the case of* $\lambda$*-algebras* (*throwing the equalities for the products and the projections in* (4′)).

*Proof.* First we start from Definition 5.1. One defines the operations by

$$(d_1 d_2)_M = [\![xy, \{x = d_1, y = d_2\}]\!]_M$$
$$(d_1, d_2)_M = [\![(x, y), \{x = d_1, y = d_2\}]\!]_M$$
$$\mathrm{fst}_M(d) = [\![\mathrm{fst}(x), \{x = d\}]\!]_M$$
$$\mathrm{snd}_M(d) = [\![\mathrm{snd}(x), \{x = d\}]\!]_M.$$

We prove the equations in (4′):

$$\begin{aligned}[\![MN, \rho]\!]_M &= [\![(xy)[x \leftarrow M, y \leftarrow N], \rho]\!]_M \\ &= [\![xy, \{x = [\![M, \rho]\!]_M, y = [\![N, \rho]\!]_M\}]\!]_M \\ &= ([\![M, \rho]\!]_M\ [\![N, \rho]\!]_M)_M.\end{aligned}$$

Likewise for the couple and the two projections.

$$\begin{aligned}([\![\lambda x.M, \rho]\!]_M\ d)_M &= ([\![\lambda x.M, \rho]\!]_M\ [\![x, \{x = d\}]\!]_M)_M \\ &= ([\![\lambda x.M, \rho[x \leftarrow d]]\!]_M\ [\![x, \rho[x \leftarrow d]]\!]_M)_M \\ &= [\![(\lambda x.M)\, x, \rho[x \leftarrow d]]\!]_M = [\![M, \rho[x \leftarrow d]]\!]_M.\end{aligned}$$

Reciprocally we prove the substitution property, first with $n = 1$,

$$\begin{aligned}[\![M[x \leftarrow N], \rho]\!]_M &= [\![(\lambda x.M)\, N, \rho]\!]_M = ([\![\lambda x.M, \rho]\!]_M\ [\![N, \rho]\!]_M)_M \\ &= [\![M, \rho[x \leftarrow [\![N, \rho]\!]_M]]\!]_M.\end{aligned}$$

A rephrasing of this argument shows the last assertion of the statement. For the extension to any $n$, we use the fact that simultaneous substitutions may be simulated by successive substitutions; we only check the case $n = 2$,

$$M[x_1 \leftarrow N_1, x_2 \leftarrow N_2] = A[z \leftarrow x_2] \qquad \text{where } A = B[x_2 \leftarrow N_2]$$

where

$$B = M[x_1 \leftarrow N_1[x_2 \leftarrow z]] \quad (\text{supposing } z \notin FV(M) \cup FV(N_1) \cup FV(N_2)).$$

We reduce

$$[\![M[x_1 \leftarrow N_1, x_2 \leftarrow N_2], \rho]\!] \text{ to } [\![A, \rho_1]\!] \qquad \text{where } \rho_1 = \rho[z \leftarrow \rho(x_2)]$$

then to $[\![B, \rho_2]\!]$ where

$$\rho_2 = \rho_1[x_1 \leftarrow [\![N_2, \rho_1]\!]] = \rho[z \leftarrow \rho(x_2), x_2 \leftarrow [\![N_2, \rho]\!]].$$

Finally we obtain $[\![M, \rho_3]\!]$, where $\rho_3 = \rho_2[x_1 \leftarrow [\![N_1[x_2 \leftarrow z], \rho_2]\!]]$. We compute $[\![N_1[x_2 \leftarrow z], \rho_2]\!] = [\![N_1, \rho_4]\!]$, where

$$\rho_4 = \rho_2[x_2 \leftarrow \rho_2(z)] = \rho_2[x_2 \leftarrow \rho(x_2)] = \rho[z \leftarrow \rho(x_2)].$$

Finally using $z \notin FV(N_1)$ $[\![N_1[x_2 \leftarrow z], \rho_2]\!] = [\![N_1, \rho]\!]$ and using $z \notin FV(M)$ $[\![M[x_1 \leftarrow N_1, x_2 \leftarrow N_2], \rho]\!] = [\![M, \rho[x_1 \leftarrow [\![N_1, \rho]\!], ..., x_2 \leftarrow [\![N_2, \rho]\!]]]\!]$.

In Definition 5.1 the semantics is a value relative to a term *and* an environment, whereas Section 2 implicitly suggests that we could associate in a value with a term only, setting

$$[\![M]\!]_{\mathrm{M}} = (M_{\mathrm{DB}})_{\mathrm{M_{CCL}}}.$$

But this can also be done directly from our model definition.

5.7. LEMMA. *Let* M *be a model, M, N two terms,* $x_0, ..., x_m$, $y_0, ..., y_n$ *two sequences of variables s.t.* $FV(M) \cup FV(N) \subseteq \{x_0, ..., x_m\} \cap \{y_0, ..., y_n\}$.

*For all environments* $v$, $\rho$ *the following holds*:

$$[\![\lambda x_m ... x_0.M, v]\!]_{\mathrm{M}} = [\![\lambda x_m ... x_0.N, v]\!]_{\mathrm{M}}$$

iff

$$[\![\lambda y_n ... y_0.M, \rho]\!]_{\mathrm{M}} = [\![\lambda y_n ... y_0.N, \rho]\!]_{\mathrm{M}}.$$

*Proof.* We only have to check the property in two situations:

— $n \geqslant m$ and $\forall i \leqslant m$, $x_i = y_i$: Let $P = \lambda z y_n ... y_{m+1}.z$. One has

$$\lambda y_n .. y_0.M = P(\lambda x_m ... x_0.M) \qquad \text{and} \qquad \lambda y_n .. y_0.N = P(\lambda x_m ... x_0.N)$$

(we use that $FV(M) \cup FV(N) \subseteq \{x_0, ..., x_m\}$), and

$$\lambda x_m .. x_0.M = (\lambda y_n ... y_0.M)\ y_n ... y_{m+1}$$

and

$$\lambda x_m ... x_0.N = (\lambda y_n ... y_0.N)\ y_n ... y_{m+1}.$$

We conclude by 5.6

— $n = m$ and $\forall i \leqslant n$, $x_i = y_{\sigma(i)}$, where $\sigma$ is a permutation: we proceed as above, using $P = \lambda z x_n ... x_0.z x_{\sigma(n)} ... x_{\sigma(0)}$.

Hence we may safely write

$$[\![M]\!]_{\mathrm{M}} = [\![N]\!]_{\mathrm{M}} \quad \text{iff } (\forall \rho, [\![\lambda x_m \ldots x_0.M, \rho]\!]_{\mathrm{M}} = [\![\lambda x_m \ldots x_0.N, \rho]\!]_{\mathrm{M}})$$

if $FV(M) \cup FV(N) \subseteq \{x_0, \ldots, x_m\}$ (for any $\rho$).

EXERCISE. If M is a $\beta\eta SP$-model of $\lambda\mathbf{c}$, then the following holds:

$$[\![M]\!]_{\mathrm{M}} = [\![N]\!]_{\mathrm{M}} \Leftrightarrow (M_{\mathrm{DB}(x_0,\ldots,x_n)})_{\mathrm{M_{CCL}}} = (N_{\mathrm{DB}(x_0,\ldots,x_n)})_{\mathrm{M_{CCL}}}.$$

The next proposition states that the equivalence just defined above is a congruence. First we introduce the useful context notation.

5.8. DEFINITION. The **contexts** are functions from $\lambda$ to $\lambda$, denoted by $C[\ ] = M \mapsto C[M]$ and defined as follows:

— the identity $M \mapsto M$, and for all term $N$, the constants $M \mapsto N$ are contexts

— if $C_1[\ ]$ and $C_2[\ ]$ are contexts, then the function $M \mapsto C_1[M]\ C_2[M]$, denoted by $C_1[\ ]\ C_2[\ ]$, is a context

— if $C[\ ]$ is a context and $x$ a variable, then the function $M \mapsto \lambda x.C[M]$, denoted by $\lambda x.C[\ ]$, is a context.

The interest of the context notation w.r.t. substitution is that no care is taken for free variable captures; compare

$$(\lambda y.x)[x \leftarrow y] \text{ and } C[y] \qquad \text{where } C[\ ] = M \mapsto \lambda y.M.$$

5.9. PROPOSITION. *Let* M *be a model. The following implication is true for all terms* $M, N$ *and any context* $C[\ ]$:

$$[\![M]\!]_{\mathrm{M}} = [\![N]\!]_{\mathrm{M}} \Rightarrow [\![C[M]]\!]_{\mathrm{M}} = [\![C[N]]\!]_{\mathrm{M}}.$$

*Proof.* By induction on the context. The basic cases are evident. Let $x_0, \ldots, x_n$ be s.t. $FV(C[M]) \cup FV(C[N]) \subseteq \{x_0, \ldots, x_n\}$:

— $C[\ ] = C_1[\ ]\ C_2[\ ]$. Let $P = \lambda yzx_n \ldots x_0.(yx_n \ldots x_0)(zx_n \ldots x_0)$. We conclude as in the previous lemma by remarking

$$\lambda x_n \ldots x_0.C[\ ] = P(\lambda x_n \ldots x_0.C_1[\ ])(\lambda x_n \ldots x_0.C_2[\ ])$$

— $C[\ ] = \lambda x.C_1[\ ]$. Evident since

$$\lambda x_n \ldots x_0.C[\ ] = \lambda x_n \ldots x_0 x.C_1[\ ]$$

and

$$FV(C_1[M]) \cup FV(C_1[N]) \subseteq \{x, x_0, ..., x_n\}.$$

*Remark.* The model definition induces another equivalence between terms, namely: $M$ and $N$ are equivalent iff for all $\rho[\![M, \rho]\!]_M = [\![N, \rho]\!]_M$. As Meyer (1982) points out, this is not a congruence in general. The weak extensionality condition ensures that both equivalences coincide, and hence that the last one is a congruence (see Barendregt, 1984; Koymans, 1984, for an example of a $\lambda$-algebra which is not a $\lambda$-model). It is also easily checked that the axioms (2) and (4) may be replaced by the only axiom (4′) of 5.6, in the presence of weak extensionality. Finally a $\lambda$-algebra is extensional iff it is a weakly extensional $\beta\eta$-model.

We end the section with a short account of known characterizations of $\lambda$-algebras and $\lambda$-models. First we discuss combinatory models and functional models which are simple and elegant ways of defining $\lambda$-models. Then we turn our attention to categorical characterizations of $\lambda$-algebras and $\lambda$-models. Koymans has essentially the same constructions as in Sections 2 and 4, except that they cannot be made purely syntactic: $\lambda$-algebras are characterized either as models of categorical combinatory logic or as Cartesian closed categories with a universal object which is retract of its function space. Obtułowicz uses the idea, clearly from Section 2, that less than Cartesian closure is needed to characterize $\lambda$-algebras. Finally Longo and Moggi relax Obtułowicz's structures to get the equivalence with combinatory algebras, nicely connecting their axioms with the recursion theoretic notion of Goedel numbering. We begin with functional and combinatory models.

*Functional and Combinatory Models*

The syntactic equivalence with classical combinatory logic discussed in Section 3 induces that $\lambda$-algebras are models of $CL\beta$, or models of the axiom systems we have listed including wext or its variant with $\varepsilon$. But what do we mean by saying that M satisfies wext? If we impose

$$\forall d \in \mathrm{M}, \qquad (d_1 d)_M = (d_2 d)_M \Rightarrow \varepsilon d_1 = \varepsilon d_2$$

this is too strong a requirement, since the conclusion is inferred from a semantic equality, which is looser than an equality which has been proved by the axioms. Hence to keep at the $\lambda$-algebra level, we have to mean as a model of the axiom systems involving wext or its variant the models satisfying all the equations derivable from the axiom systems. On the other hand, the requirement above lifts us at the level of $\lambda$- models, and leads us to the notion of combinatory model. A third, equivalent, definition is

provided by functional models and stands somehow in between. We begin with functional models.

5.10. DEFINITION. A **functional model** is given by a set $D$, a set $[D \to D]$ of functions from $D$ to $D$ and two functions

$$\varphi: [D \to D] \mapsto D \qquad \text{and} \qquad \psi: D \mapsto [D \to D]$$

and s.t. the construction of the following function $(M, \rho) \mapsto [\![M, \rho]\!]\gamma D$ is possible:

$$[\![x, \rho]\!] = \rho(x)$$
$$[\![MN, \rho]\!] = \psi([\![M, \rho]\!])([\![N, \rho]\!])$$
$$[\![\lambda x.M, \rho]\!] = \varphi(d \mapsto [\![M, \rho[x \leftarrow d]]\!])$$

i.e., s.t.

$$\forall M, \rho, \qquad (d \mapsto [\![M, \rho[x \leftarrow d]]\!]) \in [D \to D].$$

One can show easily that this yields a $\lambda$-model (intuitively the weak extensionality is due to the fact that semantic values are functions). Reciprocally one associates a functional model with a $\lambda$-model M in the following way. Let

$$D = |\mathbf{M}|$$
$$[D \to D] = \{d_0 \mapsto (dd_0)_{\mathbf{M}} \mid d \in |\mathbf{M}|\}$$
$$\varphi(f) = [\![\lambda x.yx, \{y = d\}]\!] \qquad \textit{if } d \text{ is s.t. } f = \psi(d)$$
$$\psi(d) = d_0 \mapsto (dd_0)_{\mathbf{M}}.$$

Elementary computations show that $\varphi$ is well defined, that the construction yields a functional model, and that it has the previous construction as inverse. The correspondence is extended to the extensional case by adding the condition $\varphi \circ \psi = \mathrm{Id}$.

Here is the second definition.

5.11. DEFINITION. A **combinatory model** is a model M of the following one-sorted equational presentation, satisfying additionnally a first-order implication:

— the operators are $S$, $K$, $\varepsilon$ of arity 0, and the application of arity 2

— the equations are

$$Sxyz = (xz)(yz)$$
$$Kxy = x$$
$$\varepsilon xy = xy$$

— the implication is, for all $d_1, d_2$ in $|\mathrm{M}|$,

$$(\forall d \in |\mathrm{M}|, \quad (d_1 d)_{\mathrm{M}} = (d_2 d)_{\mathrm{M}}) \Rightarrow (\varepsilon d_1)_{\mathrm{M}} = (\varepsilon d_2)_{\mathrm{M}}.$$

$\varepsilon d$ may be viewed as a canonical representative for $d$ w.r.t. the equivalence defined by $d_1 \sim d_2$ iff $(\forall d, (d_1 d)_{\mathrm{M}} = (d_2 d)_{\mathrm{M}})$. With a combinatory model we associate a functional model as follows:

$$D = |\mathrm{M}|$$
$$[D \to D] = \{d_0 \mapsto (dd_0)_{\mathrm{M}} \mid d \in |\mathrm{M}|\}$$
$$\varphi(f) = \varepsilon d \qquad \text{if } f = \psi(d)$$
$$\psi(d) = d_0 \mapsto (dd_0)_{\mathrm{M}}.$$

Reciprocally we associate a combinatory model $\mathrm{M}_C$ with a $\lambda$-model M:

$$|\mathrm{M}_C| = |\mathrm{M}|$$
$$S_{\mathrm{M}_c} = [\![\lambda xyz.(xz)(yz), \rho]\!]_{\mathrm{M}}$$
$$K_{\mathrm{M}_c} = [\![\lambda xy.x, \rho]\!]_{\mathrm{M}}$$
$$\varepsilon_{\mathrm{M}_c} = [\![\lambda xy.xy, \rho]\!]_{\mathrm{M}}.$$

The two constructions are not totally inverse:

— If M is a $\lambda$-model, if $\mathrm{M}_C$ is the associated combinatory model, if $\mathrm{M}_{C,F}$ is the functional model associated with $\mathrm{M}_C$, then the $\lambda$-model associated with $\mathrm{M}_{C,F}$ coincides with M.

— If N is a combinatory model, if M is the $\lambda$-model obtained from the functional model associated with N, then $\mathrm{M}_C$ is the combinatory model $\mathrm{N}_S$ built from N as follows:

$|\mathrm{N}_S| = |\mathrm{N}|$ and the application in $\mathrm{N}_S$ is that of N
$$\varepsilon_{\mathrm{N}_S} = (\varepsilon\varepsilon)_{\mathrm{N}}$$
$$K_{\mathrm{N}_S} = (\varepsilon_2 K)_{\mathrm{N}}$$
$$S_{\mathrm{N}_S} = (\varepsilon_3 S)_{\mathrm{N}}$$

where $\varepsilon_2, \varepsilon_3$ are as follows:

$$\varepsilon_2 = (B\varepsilon)(B\varepsilon), \qquad \varepsilon_3 = (B\varepsilon)(B\varepsilon_2) \qquad \text{where } B = S(KS)\,K.$$

The properties above arise from the proof naturally (cf. Sect. 3). They lead to the following definition (Meyer, 1982).

5.12. DEFINITION. A combinatory model is called **stable** if it satisfies the three following equations, where $\varepsilon_2, \varepsilon_3$ are as above:

$$\varepsilon\varepsilon = \varepsilon$$

$$K = \varepsilon_2 K$$

$$S = \varepsilon_3 S.$$

It is easily checked that for any $\lambda$-model M, $\mathrm{M}_C$ is a stable combinatory model; moreover the chain of constructions starting from a stable combinatory model yields the same combinatory model back, whence we may deduce that in a stable combinatory model $\varepsilon$ determines $S$, $K$; i.e., once the interpretation of the application and of $\varepsilon$ is given, there is only one possible choice for the meanings of $S$, $K$ (notice that the construction of a functional model from a combinatory model does not depend on the interpretation of $S$, $K$). But also $S$, $K$ determine $\varepsilon$, as follows easily from the following exercise, which completes the connection with Section 3.

EXERCISE. Let M be a combinatory model. Show for all $M$, $\rho$

$$[\![M, \rho]\!] = \rho^{\#}_{\mathrm{M}}(M_{CL})$$

where the semantic function $[\![\ ]\!]$ *is taken in the* $\lambda$-model built as above from M.

*Semantic Categorical Constructions*

Following suggestions of Scott (1980), Koymans (1982, 1984) has characterized the $\lambda$-algebras by categorical constructions which are quite similar to those presented here, except that they are (and indeed need to be) expressed in a semantic rather than syntactic setting. Koymans establishes triangular relations between three calculi:

— The $\lambda$-calculus with $\beta$-reduction only.

— The pure categorical combinatory endowed with various weakenings of $CCL\beta\eta SP$. Models of these theories are called *Cartesian closed monoids.*

— A free Cartesian closed category whose objects are generated from a unique object $U$, retract of $U \Rightarrow U$, which is axiomatized by introducing two operators $\varphi: (U \Rightarrow U) \to U$ and $\psi: U \to (U \Rightarrow U)$ s.t. $\psi \circ \varphi = \mathrm{Id}$. This induces that every object is retract of $U$, hence $U \times U$ in particular. Variants of the calculus arise from considering this retraction explicitly, and even canonically. Models of these calculi are called categorical $\lambda$-algebras.

Koymans grafts a semantic chain on this triangle:

— With a $\lambda$-algebra he associates a Cartesian closed monoid in a way which looks very much like the translation in 2.22 from **CCL** to $\lambda\mathbf{c}$, composed with the translation from $\lambda\mathbf{c}$ to $\lambda$ defined in 2.7. Remember that $\eta$ was needed to prove by equational deductions that the identity is a neutral element. The solution of Koymans is to restrict the models to the points "verifying" $\eta$.

— With a Cartesian closed monoid we associate a categorical $\lambda$-algebra, using a construction due in its full generality to Karoubi (1978), adapted to $\lambda$-calculus by Scott (1980). Objects are idempotents for composition, and arrows $d$ from, say, $a$ to $b$ are s.t. $d = b \circ d \circ a$.

— With a categorical $\lambda$-algebra we associate a $\lambda$-algebra, the definition of which involves a De Bruijn-like translation in which constants $\varphi, \psi$ are incorporated at appropriate places. That the $\beta$-rule is simulated may be proved as in 2.17.

These constructions yield equivalences of categories of models. Moreover the categorical counterpart of weak extensionality is the property of having **enough points**, i.e., for any objects $A$, $B$ and arrows $f, g: A \to B$, then

$$f = g \text{ iff for any } x: 1 \to A, \qquad f \circ x = g \circ x$$

(the property assumes the existence of a terminal object 1).

Another categorical characterization is provided by Church algebraic theories. Obtułowicz has pointed out a connection between $\lambda$-algebras and categories, involving the algebraic theories of Lawvere. The following account is directly inspired by Obtułowicz (1977, 1982). First we make two remarks about the Scott–Koymans approach

— The heavy and technical aspects of the construction of Koymans are primarily due to the coding of couples and projections in the $\lambda$-calculus. It would be nice not to have to cope with it.

— The equations in $CCL\beta$ lack $S\Lambda$: intuitively this means that $\beta$ corresponds to less than Cartesian closure. Nevertheless a Cartesian closed category is built, and then "loosened" by considering a retraction, and not an isomorphism, between $U$ and $U \Rightarrow U$.

In contrast Obtułowicz proposes the following kind of Cartesian category where these complications disappear, built in the framework of algebraic theories (Manes, 1976) which we recall first. An algebraic theory may be viewed as a model for a calculus of categorical combinators without App and $\Lambda$. But it is convenient to handle products of any arity, so that the objects built on a single basic type $[1]$ and a terminal object $[0]$ may be written $[0],..., [n],...$, and the operators and equations will express that $[m+n]$ is a product of $[m]$ and $[n]$. Now, formally, we define the following equational presentation:

— types are couples of integers, written $[m] \to [n]$

— operators are

— Id, $\circ$, 1 (with appropriate types)

— for all $1 \leqslant i \leqslant n$ $p_i^n$: $[n] \to [1]$

— for any $m, n \geqslant 1$ an $n$-ary pairing operator associating $\langle A_1,..., A_n \rangle$: $[m] \to [n]$ with $A_1,..., A_n$: $[m] \to [1]$

— the equations are those expressing that we have a Cartesian category.

Models of this presentation are called **algebraic theories**. The reader may check that $[1]$ is a product of $[0]$ and $[1]$, writing $\mathrm{Fst} = 1$, $\mathrm{Snd} = \mathrm{Id}$, and $\langle A, B \rangle = B$. Also, to specify an algebraic theory, we only need to define $[n] \to [1]$ for all $n$, and then take as $[n] \to [m]$ the set product of $m$ copies of $[n] \to [1]$.

Now a **Church algebraic theory**, as defined by Obtułowicz, is a model of the above presentation augmented by two operators, $\varepsilon$: $[2] \to [1]$ (hence not polymorphic) and $*$ associating $A^*$: $[n] \to [1]$ with any $A$: $[n+1] \to [1]$ (for arbitrary $n$). The equations are the suitable rephrasing of App (or Beta) and D$\Lambda$. Hence, roughly speaking, a Church algebraic theory is a Cartesian category where a distinguished object $U$ is almost an exponential of $U$ and $U$.

We are now in position to connect $\lambda$-algebras and Church algebraic theories. With a $\lambda$-algebra M we associate a Church algebraic theory $\mathrm{M}_I$ defined as follows:

$$[n] \to [1] = \{ [\![ \lambda x_1 ... x_n . d x_1 ... x_n ]\!] \mid d \in |\mathrm{M}| \}$$

(we write freely $[\![ \lambda x_1 ... x_n . y x_1 ... x_n,\ \{ y = d \} ]\!]_{\mathrm{M}} = [\![ \lambda x_1 ... x_n . d x_1 ... x_n ]\!]$) (in particular $[0] \to [1] = |\mathrm{M}|$),

$$a \circ (b_1^{[m] \to [1]},..., b_n) = [\![ \lambda x_1 ... x_m . a(b_1 x_1 ... x_m) ... (b_n x_1 ... x_m) ]\!]$$

(in particular $a \circ 1^{[m]} = [\![\lambda x_1 \ldots x_m . a]\!]$),

$$p_i^n = [\![\lambda x_1 \ldots x_n . x_i]\!]$$
$$\varepsilon = [\![\lambda xy . xy]\!]$$
$$a^* = a.$$

Checking that we have obtained a Church algebraic theory is quite a routine. Notice the very simple definition of currying: the sets $[n] \to [1]$ form a decreasing chain for inclusion. The association of a $\lambda$-algebra $\mathrm{M}_{II}$ with a Church algebraic theory M is as in our setting via a De Bruijn-like translation which is as follows ("OB" for Obtułowicz):

$$x_{\mathrm{OB}(x_1,\ldots,x_n)} = p_i^n \qquad (i \text{ minimum s.t. } x = x_i)$$
$$(\lambda x . M)_{\mathrm{OB}(x_1,\ldots,x_n)} = M^*_{\mathrm{OB}(x,x_1,\ldots,x_n)}$$
$$(MN)_{\mathrm{OB}} = \varepsilon \circ \langle M_{\mathrm{OB}}, N_{\mathrm{OB}} \rangle.$$

Hence $M_{\mathrm{OB}(x_1,\ldots,x_n)} : [n] \to [1]$; $\mathrm{M}_{II}$ is defined by

$$|\mathrm{M}_{II}| = \mathrm{M}^{[0]\to[1]}$$
$$[\![M, \{x_1 = d_1, \ldots, x_n = d_n\}]\!] = M_{\mathrm{OB}(x_1,\ldots,x_n)} \circ \langle d_n, \ldots, d_1 \rangle$$

(in particular for $M$ closed $[\![M, \{\ \}]\!] = M_{\mathrm{OB}(\ )}$).

The constructions are connected nicely:

— For any $\lambda$-algebra M, $\mathrm{M}_{I,II} = \mathrm{M}$ (check $(M_{\mathrm{OB}(x_1,\ldots,x_n)})_{\mathrm{M}_I} = [\![\lambda x_n \ldots x_1 . M]\!]_{\mathrm{M}}$).

— For any Church algebraic theory M, $\mathrm{M}_{II,I}$ is isomorphic to M. The key property here is that $a = [\![\lambda x_1 \ldots x_n . a x_1 \ldots x_n]\!]_{\mathrm{M}_{II}}$ iff $a = b^{*n}$ for some $b$ (setting $b^{*n} = (b^{*n-1})^*$): take for instance

$$[\![\lambda x . ax]\!]_{\mathrm{M}_{II}} = (yx)^*_{\mathrm{OB}(x,y)} \circ a = b^* \qquad \text{where } b = \varepsilon \circ \langle a \circ p_1^2, p_2^2 \rangle$$

so if $[\![\lambda x . ax]\!]_{\mathrm{M}_{II}} = a$, then $a = b^*$, and if $a = c^*$, then by App $b = c$. This suggests to define the isomorphisms $i_n$ from $[n] \to [1]$ onto $[n] \to [1]$ in $\mathrm{M}_{II,I}$ by $i_n(a) = a^{*n}$. We are left to show that this defines a homomorphism. For example, we check

$$\varepsilon_{\mathrm{M}_{II,I}} = (\varepsilon^{*2})_{\mathrm{M}}$$
$$(i_{n+1}(a))^*_{\mathrm{M}_{II,I}} = i_{n+1}(a) = i_n(a^*).$$

Checking that compositions are preserved is more tedious and is left as an exercise.

The equivalence we have described is very simple, and can be extended easily to $\lambda$-models by requiring the Church algebraic theory to have enough points, and to encompass $\eta$ by adding a suitable rephrasing of $S\Lambda$.

EXERCISE. Prove these assertions; show that with $\eta$ the equivalence can be made syntactic.

*Combinatory Algebras*

The most recent contribution to the categorical characterization of languages concerns the combinatory algebras, and was made by Longo and Moggi (1984). We propose here a slight rephrasing of their characterization, which is nicely connected with Church algebraic theories, being simply a weakening of those structures, and being nicely connected with our observations on $\beta$-reduction simulations. We shall call **combinatory algebraic theory** a model of the presentation of Church algebraic theories, where the rephrasing of $D\Lambda$ has been removed. Hence we are left with the axiom App. The bridges between combinatory algebras (cf. Sect. 3) and combinatory algebraic theories are as follows (omitting $_M$).

Let M be a combinatory algebra. We define a combinatory algebraic theory as follows:

$$[n] \to [1] = \{(d_1, \ldots, d_n) \mapsto dd_1 \ldots d_n \mid d \in |M|\}$$

(these functions are called the *representable* functions) (in particular $[0] \to [1] = |M|$),

$$p_i^n = (x_1, \ldots, x_n) \mapsto x_i$$

$$\varepsilon = (x, y) \mapsto xy$$

$$((d_1, \ldots, d_n) \mapsto dd_1 \ldots d_n)^* = (d_1, \ldots, d_{n-1}) \mapsto dd_1 \ldots d_{n-1}.$$

Of course composition, pairing, identity, and projections are the set theoretical ones. Notice that currying is not defined in a canonical way, but we do not care because we require only an existence axiom: indeed App is very different from $D\Lambda$: in the first rule only one instance of $\Lambda$ is involved; in the second, two different instances of currying are related, needing canonicity. Notice also that in contrast with the constructions from $\lambda$-algebras, we have built a category of sets and functions, and hence a category with enough points. But it does not restrict the generality of the correspondence, as we shall see below.

With any combinatory algebraic theory we associate a combinatory algebra by a simple observation on the construction of a $\lambda$-algebra from a Church algebraic theory. We can build a semantic function for all

$\lambda$-expressions exactly as if we had a Church algebraic theory. Now we shall only use a special kind of $\lambda$-expressions: those of the form $\lambda x_1 \ldots x_n . A$, where $A$ is a term of **CL**. We know from 2.17 that we may simulate

$$(\lambda x_1 \ldots x_n . A)\, x_1 \ldots x_n \xrightarrow{*} A$$

without using $DA$, because $A$ contains no abstractions. This suggests we should define

$$|\mathrm{M}_{\Pi'}| = [0] \to [1]$$
$$(d_1 d_2)_{\mathrm{M}_{\Pi'}} = \varepsilon \circ \langle d_1, d_2 \rangle$$

which allows us to prove combinatory completeness very easily. The correspondence between the two constructions is left as an

EXERCISE. Show that for any M, one has $\mathrm{M}_{\Gamma,\Pi'} = \mathrm{M}$; show that for any M, $\mathrm{M}_{\Pi',\Gamma}$ is the algebraic theory constructed from M by taking as arrows the functions defined by the arrows $f\colon [n] \to [1]$ of M, i.e.,

$$(d_1, \ldots, d_n) \mapsto f \circ \langle d_1, \ldots, d_n \rangle \qquad \text{where } d_1, \ldots, d_n\colon [0] \to [1].$$

This is the natural way of constructing a category with enough points from a given category **C** with a terminal object (by the way if **C** is Cartesian closed, is the corresponding category with enough points Cartesian closed (hint: take care of types))?

So what this exercise says is that the correspondence is not between combinatory algebras and combinatory algebraic theories, but between combinatory algebras and classes of combinatory algebraic theories defining the same combinatory algebraic theory with enough points.

Longo and Moggi also exhibit a connection with Cartesian closed categories. With a combinatory algebra one may associate (following Scott, 1976) the Cartesian closed category of *partial equivalence relations* (see Longo and Moggi, 1984) which is a combinatory algebraic theory. They decompose the validation of the rephrasing

$$x = \varepsilon \circ \langle x^* \circ \mathrm{Fst}, \mathrm{Snd} \rangle$$

of App into the case where $x$ has type $[2] \to [1]$, and the hypothesis that $[2]$ is a retract of $[1]$ (which allows us to propagate the rephrasing of App to $x^{[n]\to[1]}$ for any $n$). Now when a combinatory algebraic theory is obtained within a Cartesian closed category (as in the category of partial equivalence relations), then the decomposition becomes: $[2]$ is a retract of $[1]$, and $A(\varepsilon)$ is **principal**, i.e., for any $g$ of the same type as $A(\varepsilon)$ there

exists $h$ s.t. $g = \Lambda(\varepsilon) \circ h$. This simple notion is nicely connected with recursion-theoretic notions (Goedel numbering, *s-m-n* iteration theorem).

Finally in the case of $\lambda$-models the constructions of Koymans, Obtułowicz on one hand, of Longo and Moggi on the other hand yield the same objects modulo equivalences (we refer to Longo and Moggi, 1984, for details).

## 6. Discussion

The connection between $\lambda$-calculus and Cartesian closed categories became well known after the beautiful papers of Lambek (1980b) and Scott (1980). Lambek pioneered in treating categories as deductive systems (Lambek, 1980a). A large catalogue of proof-theoretic versions of categorical constructions may be found in Szabo (1978). This is an illustration of the so called Curry–Howard isomorphism, which identifies propositions with types (or objects), proofs with terms (or arrows). The deductive systems corresponding to typed $\lambda$-calculus and Cartesian closed categories by this isomorphism are the natural deduction and the sequent calculus, which are related directly in (Mann, 1975). Another early reference is Obtułowicz (1977) (cf. Sect. 5).

Quite independently (Berry, 1979; Curien, 1985c; Berry and Curien, 1982) contained an explicit set of semantic equations allowing us to associate values in categories with $\lambda$-expressions. As far as I know, Berry (1979, 1981) wrote the first formal proof of the fact that these equations validate the $\beta$-rule.

In addition to the pioneering works mentioned above other people contributed significantly to the comparative study of Cartesian closure versus $\lambda$-calculus: Parsaye-Ghomi (1981), Poigné (in press, 1984), Dybjer (1983, 1984), Koymans (1982, 1984), and Lambek and Scott (1985). A special mention should be made about Lambek and Scott (1985), where most of our correspondences are also pointed out; the setting is more proof-theoretic and semantic, and the underlying translations are quite different. Parsaye-Ghomi, Poigné, and Dybjer are interested in extending to higher order types the formalism of algebraic specifications. Poigné (in press) contains a proof of the equivalence between $\lambda\mathbf{c}_{\mathbf{K}}(C)$ and $\mathbf{CCC}_{\mathbf{K}}(C)$ for a given set of constants. He also discusses models in a categorical, functorial setting, investigating initiality and completeness and correcting some statements of Parsaye–Ghomi (1981), who remains pioneer in the field of higher order specifications. Dybjer proposes axiomatizations for the partial orders and fixed points involved in what I call here least fixed point applicative algebras.

All these works seem to have developed fairly independently. My own results were also obtained independently, with two exceptions however:

— The functional completeness theorem of Lambek (1980a) (which is in a typed setting) was a motivation for the constructions of Section 4, and then of Section 2; however I stress that the untyped version of functional completeness given here (2.22) was obtained independently of the similar result in Lambek and Scott (1985).

— Reading Koymans (1982) gave me some intuition on how to derive the system *CCL* from $CCL\beta\eta SP$ (cf. 1.2.14).

In my opinion the main originalities of my approach compared with the other ones are

— The equivalences are radically syntactical, and semantic equivalences are induced very naturally.

— The categorical notions are worked out in a very operational way. Strong and weak rules allow us to simulate two classically quite different ways of computing $\lambda$-expressions: $\beta$-reductions and environment machines. Moreover strong rules are nicely connected with De Bruijn's notation, while weak rules naturally lead to a very simple abstract machine.

— The typed and untyped cases are handled in exactly the same way, owing to the introduction of $\mathbf{CCL}_{\mathbf{K}}$, where the terminal object is replaced by operators which are more familiar to machines: coupling and applying.

As stressed in the introduction, Turner (1979) had a similar operational approach to classical combinatory logic, considered as executable code. The translation into classical combinators can be made as efficient as the translation into categorical combinators, following Kennaway and Sleep (1982) (cf. end of Sect. 3). But the categorical setting has the advantage of being able to freely intermix strong rules and weak rules. As stressed at the end of Section 3, this is very important in the categorical abstract machine, where compile time optimizations are directly inferred from strong rules, while execution simulates the weak rules. With classical combinators strong rules are untractable.

Finally, as far as Cartesian categories only are concerned, the categorical combinators are very like the FP systems of Backus (1978). $A$, App, and recursion seem to be hidden somehow in the metacomposition rule of the FFP systems. Here are some open problems and perspectives:

— The result of Klop (1980) on the non-confluence of the $\lambda c$-calculus relies on a fixed point argument; on the other hand, the typed $\lambda c$-calculus is canonical (cf. end of Sect. 4). Hence it seems reasonable to conjecture that $CCL\beta$ is not confluent in the untyped case. For the typed case the

following confluence result is proved in Obtułowicz (in press). Reductions are decomposed in two levels: after each reduction Beta or S$A$, compute the normal form w.r.t. to the system

$$CCL\beta - \text{Beta} + \text{FSI} + \text{SPair}$$

which is locally confluent, and which we conjecture to be noetherian even in the untyped case. We conjecture that the whole $CCL\beta\eta SP$ is noetherian in the typed case.

— The relationships between weak rules and strong rules should be studied further. We can implement the weak rules through the categorical abstract machine. What about strong rules? There should exist abstract machines for them too. The following works can be relevant, and could possibly be rephrased in categorical terms: the CUCH machine of Boehm and Dezani (1972), which is a multi-stack machine performing strong normal $\beta$ and $\eta$-reductions with an explicit treatment of $\alpha$-conversions, and some proposals in (De Bruijn, 1978; O'Donnell and Strandh, 1984) for developing a calculus of delayed De Bruijn's substitutions.

— The ability of our various strategies and implementations w.r.t. sharing and optimality should be examined carefully. Formal bridges with the framework of J.-J. Lévy (1978) should be established, but are not easy to conceive, because categorical reductions do not seem to have such a nice theory of residuals and families of redexes as in the $\lambda$-calculus. Some comparison should also be done with other sharing techniques, for example in (Raoult, 1984; Staples, 1982).

RECEIVED: September 23, 1985; ACCEPTED: November 26, 1985

## REFERENCES

BACKUS, J. (1978), Can programming be liberated from the Neumann style? A functional style and its algebra of programs, *Comm. ACM* **21**, No. 8, 613–641.

BARENDREGT, H. (1984), "The Lambda Calculus: Its Syntax and Semantics," rev. ed., North-Holland, Amsterdam.

BARENDREGT, H. (1974), Pairing without conventional constraints, *Z. Math. Logik Grundlag. Math.* **20**, 289–306.

BERRY, G. (1981), Some syntactic and categorical constructions of lambda-calculus models, "Rapport INRIA 80."

BERRY, G. (1979), "Modèles Complètement Adéquats et Stables des Lambda-calculs typés," Thèse de Doctorat d'Etat, Université Paris VII, Mars.

BERRY, G., AND CURIEN, P. L. (1982), Sequential algorithms on concrete data structures, *Theoret. Comput. Sci.* **20**, 265–321.

BOEHM, C., AND DEZANI, M. (1972), A CUCH machine, the automatic treatment of bound variables, *Internat. J. Comput. Inform. Sci.* **1**, No. 2.

DE BRUIJN, N. (1972), Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, *Indag. Math.* **34**, 381–392.

DE BRUIJN, N. (1978), Lambda-calculus with namefree formulas involving symbols that represent reference transforming mappings, *Indag. Math.* **40**, 348–356.

CARTMELL, J. (1978), "Generalized Algebraic Theories and Contextual Categories," Ph. D. thesis, University of Oxford.

CARTMELL, J. (1983), Notes on rewrite rules for adjunctions, Edinburgh University.

COUSINEAU, G., CURIEN, P.-L., AND MAUNY, M. (1985), The categorical abstract machine, *in* "Proceedings, ACM Conf. on Functional Programming Languages and Computer Architecture," Nancy; "Lect. Notes in Comput. Sci., Vol. 201"; *Sci. Comput. Programming*, to appear.

CURIEN, P.-L. (1985a), "Categorical Combinators, Sequential Algorithms, and Functional Programming," Research Notes in Theoret. Comput. Sci., Pitman, Boston/London.

CURIEN, P.-L. (1985b), Categorical combinatory logic, *in* "Internat. Colloq. Automata, Lang., Programming 85," Nafplion, Lect. Notes in Comput. Sci. Vol. 194, Springer-Verlag, New York/Berlin.

CURIEN, P.-L. (1985c), Typed categorical combinatory logic, *in* "CAAP 85 (TAPSOFT)," Berlin, Lect. Notes in Comput. Sci. Vol. 185, Springer-Verlag, New York/Berlin.

CURRY, H. B., AND FEYS, R. (1958), "Combinatory Logic," Vol. 1, North-Holland, Amsterdam.

O'DONNELL, M., AND STRANDH, P. (1984), Toward a fully parallel implementation of the lambda calculus, Johns Hopkins University, Baltimore, preprint.

DYBJER, P. (1983), "Category-Theoretic Logics and Algebras of Programs," Ph. D. thesis, Chalmers University of Technology, Goteborg.

DYBJER, P. (1984), Domain algebras, *in* "Proceedings, 11th Internat. Colloq. Automata, Lang., Programming," Antwerp, Lect. Notes in Comput. Sci. Vol. 172, Springer-Verlag, Berlin/New York.

GOGUEN, J., AND MESEGUER, J. (1982), An initiality primer, to appear.

HINDLEY, R., AND LONGO, G. (1980), Lambda-calculus models and extensionality, *Z. Math. Logik Grundlag. Math.* **26**, 289–310.

HOWARD, W. (1980), The formulae-as-types notion of construction, *in* "To H. B. Curry: Essays on Combinatoric Logic, Lambda-calculus, and Formalism" (J. P. Seldin and J. R. Hindley, Eds.), Academic Press, London/New York.

HUET, G., AND OPPEN, D. (1980), Equations and rewrite rules: A survey, *in* "Formal Language Theory: Perspectives and Open Problems" (R. Book, Ed.), pp. 349–405, Academic Press, New York.

KAROUBI, M. (1978), *K*-theory, An Introduction, Springer-Verlag, New York/Berlin.

KENNAWAY, J., AND SLEEP, M. (1982), Combinator strings, internal report, University of East Anglia.

KLOP, J. (1980), "Combinatory Reduction Systems," Ph. D. thesis, Utrecht.

KOYMANS, C. (1982), Models of the lambda calculus, *Inform. Control* **52**, No. 3, 306–332.

KOYMANS, C. (1984), "Models of the Lambda Calculus," Ph. D. thesis, Utrecht.

LAMBEK, J. (1972), Deductive systems and categories, III, *in* "Proceedings, Dalhousie Conf. on Toposes, Algebraic Geometry and Logic," Lect. Notes in Math. Vol. 274, pp. 57–82, Springer-Verlag, New York/Berlin.

LAMBEK, J. (1980a), From lambda-calculus to cartesian closed categories, *in* "To H. B. Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism" (J. P. Seldin and J. R. Hindley, Eds.), Academic Press, New York/London.

LAMBEK, J. (1980b), From types to sets, *Adv. in Math.* **36**, 113–164.

LAMBEK, J., AND SCOTT, P. (1985), "Introduction to Higher Order Categorical Logic," Cambridge Univ. Press, London/New York.

LANDIN, P. (1964), The mechanical evaluation of expressions, *Comput. J.* **6**, 308–320.

LÉVY, J. J. (1978), "Réductions Correctes et Optimales dans le Lambda-calcul," Thèse de Doctorat d'Etat, Université Paris VII.

LÉVY, J. J. (1977), Le Problème du Partage dans l'Evaluation des Lambda-expressions, *in* "Proceedings, Colloq. AFCET-SMF," Ecole Polytechnique.

LONGO, G., AND MOGGI, E. (1984), Goedel numberings, principal morphisms, combinatory algebras, *in* "Math. Found. of Comput. Sci. 1984," Praha, Lect. Notes in Comput. Sci. Vol. 176, Springer-Verlag, New York/Berlin.

MAC LANE, S. (1972), "Categories for the Working Mathematician," Graduate Texts in Mathematics Vol. 5, Springer-Verlag, Berlin/New York.

MANES, E. (1976), "Algebraic Theories," Graduate Texts in Mathematics Vol. 26, Springer-Verlag, New York/Berlin.

MANN, C. (1975), Connection between equivalence of proofs and cartesian closed categories, *Proc. London Math. Soc. (3)* **31**, 289–310.

MEYER, A. (1982), What is a model of the lambda calculus, *Inform. Control.* **52**, No. 1, 87–121.

OBTUŁOWICZ, A. (1977), Functorial semantics of the type free $\lambda$-$\beta\eta$ calculus, *in* "Proceedings, Fund. of Comput. Theory," Lect. Notes in Comput. Sci. Vol. 56, Springer-Verlag, Berlin/New York.

OBTUŁOWICZ, A., AND WIWEGER, A. (1982), Categorical, functorial and algebraic aspects of the type free lambda-calculus, *in* "Universal Algebra and Applications," Banach Center Publ. Vol. 9, Warsaw.

OBTUŁOWICZ, A. (in press), Algebra of constructions I, *Inform. Control.*

PARSAYE-GHOMI, K. (1981), "Higher Order Abstract Algebras," Ph. D. thesis, UCLA.

POIGNÉ, A. (in press), On specifications, theories, and models with higher types, *Inform. Control.*

POIGNÉ, A. (1984), Higher order data structures, Cartesian closure versus $\lambda$-calculus, *in* "Sympos. on Theoret. Aspects of Comput. Sci. 84," Lect. Notes in Comput. Sci. Vol. 166, Springer-Verlag, New York/Berlin.

POTTINGER, G. (1979), The Church–Rosser theorem for the typed $\lambda$-calculus with extensional pairing, preprint, Carnegie–Mellon University, Pittsburgh.

RAOULT, J. C. (1984), On graph rewritings, *in* "CAAP 1984," Bordeaux.

SCOTT, D. (1976), Data types as lattices, *SIAM J. Comput.* **5**, No. 3, 522–586.

SCOTT, D. (1980), Relating theories of the lambda-calculus, *in* "To H. B. Curry: Essays on Combinatory Logic, Lambda-calculus, and Formalism" (J. P. Seldin and J. R. Hindley, Eds.), Academic Press, New York/London.

STAPLES, J. (1982), A new strategy for efficient lazy evaluation of lambda expressions, *Austral. Comput. Sci. Commun.* **4**, 277–285.

SZABO, M. (1978), The algebra of proofs, *in* "Studies in Logic 83," North-Holland, Amsterdam.

TURNER, D. (1979a), A new implementation technique for applicative languages, *Software Practice Experience* **9**, 31–49.

TURNER, D. (1979b), Another algorithm for bracket abstraction, *J. Symbolic Logic* **44**, No. 3, 267–270.