Partial Categorical Multi-Combinators and Church-Rosser Theorems

Rafael D.Lins
Dept.de Informática - Universidade Federal de Pernambuco - Recife - Brazil
Computing Laboratory - the University of Kent - Canterbury - England

Abstract

Categorical Multi-Combinators form a rewriting system developed with the aim of providing efficient implementations of lazy functional languages. The core of the system of Categorical Multi-Combinators consists of only four rewriting laws with a very low pattern-matching complexity. This system allows the equivalent of several β-reductions to be performed at once, as functions form frames with all their arguments. Although this feature is convenient for most cases of function application it does not allow partially parameterised functions to fetch arguments. In this paper we present Partial Categorical Multi-Combinators, a new rewriting system, which removes this drawback.

Keywords: Functional Languages, Language Implementation, Combinators

Introduction

The method of compilation of functional languages into combinators, first explored by Turner in[Tur79], provides a way of removing the variables from a program, transforming it into an applicative combination of constant functions or **combinators**. Turner used a set of combinators based on Curry's Combinatory Logic. To each combinator there is associated a rewriting law. In rewriting a combinator expression, Turner rewrites the leftmost-outermost reducible subexpression (or *redex*) at each stage. When no further rewriting can take place the expression is said to be in **normal form**.

Another theory of functions is provided by Category Theory [Lam80], and we can see the notation used herein as providing an alternative set of combinators. The original system of Categorical Combinators was developed by Curien [Cur86]. This work was inspired by the equivalence of the theories of typed λ-calculus and Cartesian Closed Categories as shown by Lambek [Lam80] and Scott [Sco80].

Aiming to implement lazy functional languages in an efficient way using rewritings of Categorical Combinators we developed a number of optimisations [Lin86, Lin87a] of the naïve system, the most refined of which was the system of Linear Categorical Combinators [Lin87a]. The modifications introduced reduce the number of rewriting laws and increase the efficiency of the system by reducing the number of rewriting steps involved in taking an expression to normal form, whilst leaving the complexity of the pattern matching algorithm unchanged. Categorical Multi-Combinators are a generalisation of Linear Categorical Combinators. Each rewriting step of the Multi-Combinator code is equivalent to several rewritings of Linear Categorical Combinators, since an application of a function to several arguments can be reduced in a single step.

Categorical Multi-Combinators served as basis for two compiled machines: GMC [Mus91] and CM-CM [Lin90, Tho92]. GMC is inspired by the G-Machine [Joh87], in the sense that it generates graph lazily. The implementation of GMC has shown performance close, but slower, than the G-machine. CM-CM is a stack machine with a frame update mechanism similar to the one presented in TIM [Fai87]. CM-CM served as a basis for ΓCMC [Lin92], a lower level abstract machine suitable for efficient implementation of functional languages on RISC architectures. The implementation of ΓCMC, still in progress has shown performance figures which in the best case is several times faster and in the worst case it is 10% slower than Chalmers LML compiler based on the G-machine.

The core of the system of Categorical Multi-Combinators consists of only four rewriting laws with a very low pattern-matching complexity. This system allows the equivalent of several β-reductions to be performed at once, as functions form frames with all their arguments. Although this feature is convenient for most cases of function application it does not allow partially parameterised functions. In this paper we present a new set of Categorical Multi-Combinators, called Partial Categorical Multi-Combinators, which removes this drawback. We prove that Partial Categorical Multi-Combinators have the Church-Rosser properties of uniqueness of normal forms and that they are normalising, i.e. rewriting of the leftmost-outermost pattern at each point of the reduction sequence leads to normal form, if it exists. In this paper we concentrate on the formal aspects of Partial Categorical Multi-Combinators. We hope to use them as a way to fast implementation of full-laziness in ΓCMC.

Categorical Multi-Combinators

In this section we present the compilation algorithm and rewriting laws for Categorical Multi-Combinators. To make presentation easier we adopted a slightly different notation from the one presented in [Lin87b]. The multi-pair combinator is represented by a tuple $(x_0, \ldots, x_n)$, we use the empty tuple $()$ to denote identity, and angle brackets stand for closures $\langle a, b\rangle$.

Compilation Algorithm

In Categorical Multi-Combinators, function application is denoted by juxtaposition, taken to be left-associative. The compilation algorithm for translating λ-expressions into Categorical Multi-Combinators is:

(T .1) $[\underbrace{\lambda x_i \ldots \lambda x_j}_{n}.a] = \langle L^{n-1}(R^{x_i \ldots x_j} a), ()\rangle$

(T .2) $[a \ldots b] = [a] \ldots [b]$

(T .3) $[c] = c$, where c is a constant

(T .4) $R^{x_i \ldots x_j} \underbrace{\lambda x_k \ldots \lambda x_l}_{m}.a = L^{m-1}(R^{x_i \ldots x_j x_k \ldots x_l} a)$

(T .5) $R^{x_i \ldots x_j}(a \ldots b) = (R^{x_i \ldots x_j} a \ldots R^{x_i \ldots x_j} b)$

(T .6) $R^{x_i \ldots x_j} b = \begin{cases} b \text{ , } if\ b\ is\ a\ constant \\ n_k \text{ , } if\ b = x_k \end{cases}$

Again, if whenever applying rule T.6 above a variable b can be associated with more than one x_k then one must choose the minimum correspondent n_k.

Example of Compilation

There follows an example of the translation of a λ-expression into Categorical Multi-Combinators using the algorithm above:

$$
\begin{aligned}
[(\lambda a.a\ a)\ ((\lambda c \lambda d.d)B)\ C] &\overset{\text{T.2}}{=} [(\lambda a.a\ a)]\ [((\lambda c \lambda d.d)B)]\ [C] \\
&\overset{\text{T.1}}{=} \langle \mathrm{L}^0(R^a(a\ a)), ()\rangle\ [((\lambda c \lambda d.d)B)]\ [C] \\
&\overset{\text{T.5}}{=} \langle \mathrm{L}^0(R^a a\ R^a a), ()\rangle\ [((\lambda c \lambda d.d)B)]\ [C] \\
&\overset{\text{T.6}}{=} \langle \mathrm{L}^0(0\ R^a a), ()\rangle\ [((\lambda c \lambda d.d)B)]\ [C] \\
&\overset{\text{T.6}}{=} \langle \mathrm{L}^0(0\ 0), ()\rangle\ [((\lambda c \lambda d.d)B)]\ [C] \\
&\overset{\text{T.2}}{=} \langle \mathrm{L}^0(0\ 0), ()\rangle\ ([\lambda c \lambda d.d]\ [B])\ [C] \\
&\overset{\text{T.1}}{=} \langle \mathrm{L}^0(0\ 0), ()\rangle\ (\langle L^1(R^{c,d} d), ()\rangle\ [B])\ [C] \\
&\overset{\text{T.6}}{=} \langle \mathrm{L}^0(0\ 0), ()\rangle\ (\langle L^1(0), ()\rangle\ [B])\ [C]
\end{aligned}
$$

Categorical Multi-Combinator Rewriting Laws

The core of the Categorical Multi-Combinator machine is presented on page 71 of [Lin87b]. For a matter of convenience we will represent the multi-pair combinator, which forms evaluation environments as $(x_0, \ldots, x_n)$ and compositions, which represent closures, will be noted as $\langle a, b\rangle$. Using this notation the kernel of the Categorical Multi-Combinator rewriting laws is expressed as:

(M*.1) $\langle n, (x_m, \cdots, x_1, x_0)\rangle \Rightarrow x_n$

(M*.2) $\langle x_0 x_1 x_2 \ldots x_n, y\rangle \Rightarrow \langle x_0, y\rangle \ldots \langle x_n, y\rangle$

(M*.3) $\langle L^n(y), (w_0, \ldots)\rangle x_0 x_1 \cdots x_n x_{n+1} \cdots x_z \Rightarrow \langle y, (x_0, \cdots, x_n)\rangle\ x_{n+1} \cdots x_z$

(M*.4) $\langle k, (x_m, \cdots, x_1, x_0)\rangle \Rightarrow k,$ where k is a constant

The state of computation of a Categorical Multi-Combinator expression is represented by the expression itself. Rule (M*.1) performs environment look-up, this is the mechanism by which a variable fetches its value in the corresponding environment. (M*.2) is responsible for environment distribution. Rule (M*.3) performs environment formation. It is called multi-β reduction, because it is equivalent to performing several β-reductions in the λ-calculus. Rule (M*.4) discards the environment associated with a constant.

Example of Execution

The λ-expression

$$(\lambda a.a\ a)((\lambda c\lambda d.d)B)\ C$$

leftmost-outermost reduces to

$$\begin{aligned}
&\stackrel{\beta}{\Rightarrow}\ ((\lambda c\lambda d.d)B)\ ((\lambda c\lambda d.d)B)\ C\\
&\stackrel{\beta}{\Rightarrow}\ (\lambda d.d)\ ((\lambda c\lambda d.d)B)\ C\\
&\stackrel{\beta}{\Rightarrow}\ (\lambda c\lambda d.d)B\ C\\
&\stackrel{\beta}{\Rightarrow}\ (\lambda d.d)\ C\\
&\stackrel{\beta}{\Rightarrow}\ C
\end{aligned}$$

Now let us analyse how the translation of the top expression above rewrites by using the laws fo Categorical Multi-Combinators, where we assume that $[B] = B'$ and $[C] = C'$.

$$\begin{aligned}
\langle \mathrm{L}^0(0\ 0), ()\rangle\ (\langle L^1(0), ()\rangle\ B')\ C' &\stackrel{\mathrm{M*.3}}{\Rightarrow} \langle (0\ 0), (\langle L^1(0), ()\rangle\ B')\rangle\ C'\\
&\stackrel{\mathrm{M*.2}}{\Rightarrow} (\langle 0, (\langle L^1(0), ()\rangle\ B')\rangle\langle 0, (\langle L^1(0), ()\rangle\ B')\rangle)\ C'\\
&\stackrel{\mathrm{M*.1}}{\Rightarrow} (\langle L^1(0), ()\rangle\ B'\langle 0, (\langle L^1(0), ()\rangle\ B')\rangle)\ C'\\
&\stackrel{\mathrm{M*.3}}{\Rightarrow} \langle 0, (B', \langle 0, (\langle L^1(0), ()\rangle\ B')\rangle)\rangle\ C'\\
&\stackrel{\mathrm{M*.1}}{\Rightarrow} \langle 0, (\langle L^1(0), ()\rangle\ B')\rangle\ C'\\
&\stackrel{\mathrm{M*.1}}{\Rightarrow} \langle L^1(0), ()\rangle\ B'\ C'\\
&\stackrel{\mathrm{M*.3}}{\Rightarrow} \langle 0, (B', C')\rangle\\
&\stackrel{\mathrm{M*.1}}{\Rightarrow} C'
\end{aligned}$$

Reduction Order

It is a well known fact that leftmost-outermost reduction of λ-expressions is a safe but not optimal reduction strategy. In the λ-expression

$$(\lambda a.a\ a)((\lambda c\lambda d.d)B)\ C$$

if we reduce the rightmost redex first we would have

$$\begin{aligned}
&\stackrel{\beta}{\Rightarrow} \quad (\lambda a.a\ a)\ (\lambda d.d)\ C \\
&\stackrel{\beta}{\Rightarrow} \quad (\lambda d.d)\ (\lambda d.d)\ C \\
&\stackrel{\beta}{\Rightarrow} \quad (\lambda d.d)\ C \\
&\stackrel{\beta}{\Rightarrow} \quad C
\end{aligned}$$

The sequence of reductions above is shorter than the leftmost-outermost one, because the partial application, which forms the rightmost redex in the expression above, was reduced before being copied. This mechanism is desirable whenever we face similar situation to the one in the example shown.

If we analyse the sequence of reductions for Categorical Multi-Combinators we can see that the Categorical Multi-Combinator sub-expression equivalent to the rightmost redex in the λ-expression is not reducible by using any of the rewriting laws above. Categorical Multi-Combinators will make copies of the partial application and "waits" until all arguments are present to perform multi-β reduction. As functional languages only print expressions of ground type we know that the extra arguments needed will be in place whenever the partial application becomes the leftmost-outermost redex, thus making multi-β reduction possible. However, not being able to reduce partial applications has performance implications in the case they are shared.

Partial Categorical Multi-Combinator

In this section we introduce Partial Categorical Multi-combinators, a rewriting system which allows to reduce partially applied functions.

Compilation Algorithm

The compilation algorithm for translating λ-expressions into Partial Categorical Multi-Combinators is different from the presented above for Categorical Multi-Combinators. Now instead of working with the deBruijn representation for variables we work with the co-deBruijn number, as we want variables to which arguments are passed first to be represented by smaller numbers than the ones which correspond to arguments passed later. We are also explicit about parenthesisation of expressions. Thus the compilation algorithm for Partial Categorical Multi-Combinators from fully parenthesised λ-lifted expressions in the λ-Calculus is:

(T'.1) $[\underbrace{\lambda x_i \ldots \lambda x_j}_{n} .a] = \langle L^{n-1}(R^{x_j \ldots x_i} a), ()\rangle$

(T'.2) $[(\ldots(a\ b)\ldots c)] = (\ldots([a]\ [b])\ldots[c])$

(T'.3) $[c] = c$, where c is a constant

(T'.4) $R^{x_j \ldots x_i} \underbrace{\lambda x_k \ldots \lambda x_l}_{m} .a = L^{m-1}(R^{x_l \ldots x_k x_j \ldots x_i} a)$

(T'.5) $R^{x_j \ldots x_i}(\ldots(a\ b)\ldots c) = (\ldots(R^{x_j \ldots x_i} a\ R^{x_j \ldots x_i} b)\ldots R^{x_j \ldots x_i} c)$

(T'.6) $R^{x_j \ldots x_i} b = \begin{cases} b\ , \textit{if } b \textit{ is a constant} \\ n_k\ \ , \textit{if } b = x_k \end{cases}$

Again, if whenever applying rule T'.6 above a variable b can be associated with more than one x_k then one must choose the maximum correspondent n_k.

Example of Compilation

There follows an example of the translation of a λ-expression into Partial Categorical Multi-Combinators using the algorithm above:

$$
\begin{aligned}
[(((\lambda a.a\ a)\ ((\lambda c\lambda d.d)B))\ C)] &\overset{\mathrm{T}'.2}{=} (([(\lambda a.a\ a)]\ [((\lambda c\lambda d.d)B)])\ [C])\\
&\overset{\mathrm{T}'.1}{=} ((\langle \mathrm{L}^0(R^a(a\ a)),()\rangle\ [((\lambda c\lambda d.d)B)])\ [C])\\
&\overset{\mathrm{T}'.5}{=} ((\langle \mathrm{L}^0((R^a a\ R^a a)),()\rangle\ [((\lambda c\lambda d.d)B)])\ [C])\\
&\overset{\mathrm{T}'.6}{=} ((\langle \mathrm{L}^0((0\ R^a a)),()\rangle\ [((\lambda c\lambda d.d)B)])\ [C])\\
&\overset{\mathrm{T}'.6}{=} ((\langle \mathrm{L}^0((0\ 0)),()\rangle\ [((\lambda c\lambda d.d)B)])\ [C])\\
&\overset{\mathrm{T}'.2}{=} ((\langle \mathrm{L}^0((0\ 0)),()\rangle\ ([\lambda c\lambda d.d]\ [B]))\ [C])\\
&\overset{\mathrm{T}'.1}{=} ((\langle \mathrm{L}^0((0\ 0)),()\rangle\ (\langle L^1(R^{d,c}d),()\rangle\ [B]))\ [C])\\
&\overset{\mathrm{T}'.6}{=} ((\langle \mathrm{L}^0((0\ 0)),()\rangle\ (\langle L^1(1),()\rangle\ [B]))\ [C])
\end{aligned}
$$

Partial Categorical Multi-Combinator Rewriting Laws

In this section we generalise multi-β reduction to allow a function to fetch the arguments passed to it if these are fewer than its arity. Thus we have,

$$(\ldots(\langle L^n(y),(w_1,\ldots)\rangle x_0)x_1)\cdots)x_m) \Rightarrow \langle L^{n-m-1}(\langle y,(x_0,\cdots,x_m)\rangle),()\rangle \qquad \textit{if } m < n$$

Now we need to adjust the argument fetching mechanism such that variables can cope with partial multi-β reduction.

$$\langle n,(x_0,x_1,\cdots,x_m)\rangle \Rightarrow \begin{cases} x_n, & \text{if } n \le m \\ n-m-1, & \text{otherwise} \end{cases}$$

The complete set of rewriting laws for Partial Categorical Multi-Combinators is:

(P.1) $\langle n,(x_0,x_1,\cdots,x_m)\rangle \Rightarrow \begin{cases} x_n, & \text{if } n \le m \\ n-m-1, & \text{otherwise} \end{cases}$

(P.2) $\langle (x_0 x_1 x_2 \ldots x_n),y\rangle \Rightarrow (\langle x_0,y\rangle \ldots \langle x_n,y\rangle)$

(P.3) $(\ldots(\langle L^n(y),(\cdots,w_l)\rangle x_0)x_1)\cdots)x_n)x_{n+1})\cdots)x_z) \Rightarrow \langle(\ldots(y,(x_0,\cdots,x_n)\rangle\ x_{n+1})\cdots)x_z)$

(P.4) $(\ldots(\langle L^n(y),(\cdots,w_l)\rangle x_0)x_1)\cdots)x_m) \Rightarrow \langle L^{n-m-1}(\langle y,(x_0,\cdots,x_m)\rangle),()\rangle$ if $m < n$

(P.5) $\langle k,(x_m,\cdots,x_1,x_0)\rangle \Rightarrow k$ where k is a constant

Example of Evaluation

Let us analyse the Partial Categorical Multi-Combinator expression we had in the example above under a reduction strategy similar to the one we adopted for the reduction of the equivalent λ-expression in the last section, i.e. reducing the rightmost redex first.

$$
\begin{aligned}
((\langle \mathrm{L}^0((0\ 0)),()\rangle\ (\langle L^1(1),()\rangle\ B'))\ C') &\overset{\mathrm{P}.4}{\Rightarrow} ((\langle \mathrm{L}^0((0\ 0)),()\rangle\ \langle L^0(\langle 1,(B')\rangle),()\rangle)\ C')\\
&\overset{\mathrm{P}.1}{\Rightarrow} ((\langle \mathrm{L}^0((0\ 0)),()\rangle\ \langle L^0(0),()\rangle)\ C')
\end{aligned}
$$

at this point the partial parameterisation of the function on the right hand side of the expression above was fully reduced giving rise to a new function. Evaluation proceeds as follows:

$$
\begin{array}{ll}
\stackrel{\text{P.4}}{\Rightarrow} & (\langle(0\ 0), (\langle L^0(0), ()\rangle)\rangle\ C') \\
\stackrel{\text{P.2}}{\Rightarrow} & ((\langle 0, (\langle L^0(0), ()\rangle)\rangle\ \langle 0, (\langle L^0(0), ()\rangle)\rangle)\ C') \\
\stackrel{\text{P.1}}{\Rightarrow} & ((\langle L^0(0), ()\rangle\ \langle 0, (\langle L^0(0), ()\rangle)\rangle)\ C') \\
\stackrel{\text{P.3}}{\Rightarrow} & (\langle 0, (\langle 0, (\langle L^0(0), ()\rangle)\rangle)\rangle\ C') \\
\stackrel{\text{P.1}}{\Rightarrow} & (\langle 0, (\langle L^0(0), ()\rangle)\rangle\ C') \\
\stackrel{\text{P.1}}{\Rightarrow} & (\langle L^0(0), ()\rangle\ C') \\
\stackrel{\text{P.3}}{\Rightarrow} & \langle 0, (C')\rangle \\
\stackrel{\text{P.1}}{\Rightarrow} & C'
\end{array}
$$

Church-Rosser Theorems

In this section we analyse some of the formal aspects of Partial Categorical Multi-Combinators as a rewriting system.

The first Church-Rosser theorem for the λ-Calculus proves the uniqueness of normal forms of λ-expressions, if it exists. This means that whatever terminating sequence of reductions of a λ-expression will lead to the same result. A rewriting system to which the Church-Rosser property is valid is called *confluent* or Church-Rosser.

The second Church-Rosser theorem for the λ-Calculus shows that the reduction of the leftmost-outermost redex at each point of the reduction sequence of an expression leads to normal form, if it exists.

In this section we show that Partial Categorical Multi-Combinators have the properties stated by the two Church-Rosser theorems.

Normal Forms

Here we prove that Partial Categorical Multi-Combinators have the property that if we start from a Partial Categorical Multi-Combinator expression any terminating sequence of reductions lead to the same expression.

Our strategy for proving this property is based on Huet's version of the Knuth-Bendix algorithm [Hue80]. Huet proves that if a rewriting system is *left-linear* and has no *critical pairs* it is normalising. A rewriting system is said to be left-linear if no variable appear more than once in the left-hand side of any of its rewriting rules. Critical pairs are computed by a superposition algorithm, where one attempts to match in a most general way the left-hand side of some rewriting rule with a nonvariable subterm of all rewriting rules in the system, including itself. Critical pairs show the possibility of reduction sequences diverging.

If we analyse the set of rewriting laws for Partial Categorical Multi-Combinators we can see that there is no repeated variable in the left-hand side of any of the rewriting rules and that there is no possible overlapping of patterns in the left hand side of any of the rewriting laws. Any rewritable pattern matches trivially with a variable of any of the rewriting laws in the system, therefore there are no critical pairs. We have proved that Partial Categorical Multi-Combinators form a confluent rewriting system, thus have the Church-Rosser property of uniqueness of normal forms.

Normalisation Property

In this section we prove that the reduction of the leftmost-outermost redex in each point of the reduction sequence leads to normal form, if it exists. A direct proof of this theorem is not simple. Our strategy is to produce a proof in three steps. First, we present the λ-Calculus with lazy substitutions [Lin86], a rewriting system which performs β-reductions, but variable substitution is performed explicitly, if and only if needed. The second step is to introduce the λ-Calculus with Multiple Substitutions, a rewriting system in which each rewriting step is equivalent to several β-reductions. Variable substitution is also performed explicitly and on demand. We then show that leftmost-outermost rewritings of Partial Categorical Multi-Combinators are equivalent to leftmost-outermost rewritings in the λ-Calculus with Multiple Substitutions, therefore equivalent in each reduction step to a sequence of leftmost-outermost β-reductions in the λ-Calculus.

The λ-Calculus with Lazy Substitutions

The rewriting system called the λ-Calculus with lazy substitutions was introduced in [Lin86] as a way to prove that the leftmost-outermost rewriting of Simplified Categorical Combinators [Lin87a] was equivalent to perform leftmost-outermost β-reduction in the λ-Calculus.

The rewriting laws in the λ-Calculus with lazy substitutions are:

l.1 $(\lambda x.a)A \Rightarrow [A/x]a$

l.2 $[A/x]\ \lambda z.a \Rightarrow \begin{cases} \lambda z.a, & \text{if } x \neq z \\ \lambda z.[A/x]a, & \text{if } x \neq z \text{ and } z \text{ not free in } A \\ \lambda z.[A/x][w/z]a, & \text{where } w \text{ is a new variable} \end{cases}$

l.3 $[A/x]\ (a\ b) \Rightarrow ([A/x]a\ [A/x]b)$

l.4 $[A/x]\ x \Rightarrow A$

l.5 $[A/x]\ z \Rightarrow z$

Rule l.1 above is β-reduction with an explicit variable substitution operator $[A/x]$. Rules l.2 and l.3 shifts the substitution operator into the body of an abstraction and distributes it through an application, respectively. Rules l.4 and l.5 perform actual substitution of formal parameters for real parameters.

It is obvious that the leftmost-outermost reduction in the λ-Calculus with lazy substitutions is equivalent to leftmost-outermost β-reduction in the λ-Calculus.

The λ-Calculus with Multiple Substitutions

Assuming we have the following λ-expression,

$$(\lambda x.\lambda y.\lambda z.a)\ T\ U\ V \ldots$$

applying the rewriting rules of the λ-Calculus with lazy substitutions it leftmost-outermost reduces to:

$$\begin{aligned}
&\overset{l.1}{\Rightarrow} && ([T/x]\ \lambda y.\lambda z.a)\ U\ V \ldots \\
&\overset{l.2}{\Rightarrow} && (\lambda y.[T/x]\lambda z.a)\ U\ V \ldots \\
&\overset{l.1}{\Rightarrow} && ([U/y][T/x]\lambda z.a)\ V \ldots \\
&\overset{l.2}{\Rightarrow} && ([U/y]\lambda z.[T/x]a)\ V \ldots \\
&\overset{l.2}{\Rightarrow} && (\lambda z.[U/y][T/x]a)\ V \ldots \\
&\overset{l.1}{\Rightarrow} && ([V/z][U/y][T/x]a) \ldots
\end{aligned}$$

As we can observe in the sequence of reductions above no other rewriting takes place until all the substitution operators appear. There is no reason why not to rewrite the top expression directly into the bottom one, as this will always be the leftmost-outermost rewriting path. Thus having,

$$(\lambda x.\lambda y.\lambda z.a)\ T\ U\ V \ldots \Rightarrow ([V/z][U/y][T/x]a) \ldots$$

Making this a new rewriting law and adopting a more convenient notation for the substitution operator we have the following rewriting system, called the λ-Calculus with Multiple Substitutions:

m.1 $(\lambda x_1.\lambda x_2.\dots\lambda x_n.a)A_1\ A_2\dots A_m \Rightarrow \lambda x_{m+1}\dots\lambda x_n.[A_1/x_1, A_2/x_2, \dots, A_m/x_m]a$, if $m < n$.

m.2 $(\lambda x_1.\lambda x_2.\dots\lambda x_n.a)A_1\ A_2\dots A_n A_{n+1}\dots \Rightarrow [A_1/x_1, A_2/x_2, \dots, A_n/x_n]a\ A_{n+1}\dots$, otherwise.

m.3 $[A_1/x_1, \dots, A_n/x_n]\ \lambda z.a \Rightarrow \lambda z.[A_1/x_1, \dots, A_n/x_n]a$

m.4 $[A_1/x_1, \dots, A_n/x_n]\ (a\dots b) \Rightarrow ([A_1/x_1, \dots, A_n/x_n]a\dots[A_1/x_1, \dots, A_n/x_n]b)$

m.5 $[A_1/x_1, \dots, A_n/x_n]\ x_i \Rightarrow A_i$

m.6 $[A_1/x_1, \dots, A_n/x_n]\ z \Rightarrow z$

In rule m.3 we assume that for all i we have $x_i \neq z$ and that z does not appear free in any expression A_i, α-conversion may be needed to guarantee this condition.

By construction we can see that the leftmost-outermost reduction in the λ-Calculus with Multiple Substitutions is equivalent to a sequence of leftmost-outermost reductions in the λ-Calculus with Lazy Substitutions, therefore a non-terminating sequence in the former gives rise to one in the latter; thus this reduction strategy is normalising.

Final Step

Now we prove that the rewriting of the leftmost-outermost pattern of Partial Categorical Multi-Combinators corresponds to rewriting the leftmost-outermost redex in the λ-Calculus with Multiple Substitutions. We first introduce a translation function $\mathcal{T}$ which translates Partial Categorical Multi-Combinator expressions into expressions of the λ-Calculus with Multiple Substitutions. The translation function $\mathcal{T}$ is defined as:

(t.1) $\mathcal{T}^{w_m\dots w_1}(\dots(a\ b)\dots)\ l) = \mathcal{T}^{w_m\dots w_1}a\ \mathcal{T}^{w_m\dots w_1}b\dots\mathcal{T}^{w_m\dots w_1}l$

(t.2) $\mathcal{T}^{w_m\dots w_1}L^n(\langle y, (x_1, \dots, x_m)\rangle) = (\lambda w_{m+1}\dots\lambda w_{n-m+1}.\mathcal{T}^{w_{m+n+1}\dots w_1}y)$

(t.3) $\mathcal{T}^{w_m\dots w_1}\langle L^n(y), (x_m, \cdots, x_1)\rangle = (\lambda w_1\dots\lambda w_{n+1}.\mathcal{T}^{w_{m+n}\dots w_1}y)$

(t.4) $\mathcal{T}^{w_m\dots w_1}\langle y, (x_n, \cdots, x_1)\rangle = [\mathcal{T}^{[\,]}x_1/w_1, \dots, \mathcal{T}^{[\,]}x_n/w_n]\mathcal{T}^{w_{m+n}\dots w_1}y$

(t.5) $\mathcal{T}^{w_m\dots w_1}n = w_{n+1}$, if n is a variable.

(t.6) $\mathcal{T}^{w_m\dots w_1}k = k$, if k is a constant.

As we can observe the translation function $\mathcal{T}$ keeps syntactic ordering, i.e. there is no transposition of redexes, therefore the leftmost-outermost Partial Categorical Multi-Combinator redex after translation is the leftmost-outermost redex in the λ-Calculus with Multiple Substitutions.

Now we need to prove that the translation function $\mathcal{T}$ is a correct mapping between the two rewriting systems by analysing the behaviour of original and translated expressions. We show that if a Partial Categorical Multi-Combinator expression A leftmost-outermost rewrites in one step to expression A' then the translation of A into the λ-Calculus with Multiple Substitutions, $\mathcal{T}A$, leftmost-outermost rewrites to $\mathcal{T}A'$, in one step. So we have commutativity of the following diagram,

$$\begin{array}{ccc} A & \xrightarrow{\mathcal{T}} & \mathcal{T}A \\ \Downarrow & & \Downarrow \\ A' & \xrightarrow{\mathcal{T}} & \mathcal{T}A' \end{array}$$

We will analyse each of the rewritable patterns for Partial Categorical Multi-Combinators.

P.1

$$
\begin{array}{ccl}
\mathcal{T}^{[]}\langle n,(x_0,\ldots,x_m)\rangle & \stackrel{t.4}{=} & [\mathcal{T}^{[\,]}x_0/w_1,\ldots,\mathcal{T}^{[\,]}x_m/w_{m+1}]\mathcal{T}^{w_{m+1}\ldots w_1}n \\
\Downarrow P.1 & & \parallel t.5 \\
\mathcal{T}^{[\,]}x_n & & [\mathcal{T}^{[\,]}x_0/w_1,\ldots,\mathcal{T}^{[\,]}x_m/w_{m+1}]w_{n+1} \\
 & & \Downarrow m.5 \\
 & & \mathcal{T}^{[\,]}x_n
\end{array}
$$

The second clause in rule P.1 is never the leftmost-outermost redex in a Partial Multi-Combinator expression. If it were we would have a situation equivalent to having a free variable in the code.

P.2

$$
\begin{array}{rcl}
\mathcal{T}^{[\,]}(\langle(x_0\ldots x_n),(v_1,\ldots,v_m)\rangle) & \stackrel{t.4}{=} & [\mathcal{T}^{[\,]}v_1/w_1,\ldots,\mathcal{T}^{[\,]}v_m/w_m]\mathcal{T}^{w_m\cdots}(x_0,\ldots,x_n) \\
\Downarrow P.2 & & \parallel t.1 \\
\mathcal{T}^{[\,]}(\langle x_0,(v_1,\ldots,v_m)\rangle\ldots\langle x_n,(v_1,\ldots,v_m)\rangle) & & [\mathcal{T}^{[\,]}v_1/w_1,\ldots]\mathcal{T}^{w_m\cdots}x_0\ldots\mathcal{T}^{w_m\cdots}x_n \\
\parallel t.1 & & \Downarrow m.4 \\
\mathcal{T}^{[\,]}\langle x_0,(v_1,\ldots,v_m)\rangle\ldots\mathcal{T}^{[\,]}\langle x_n,(v_1,\ldots,v_m)\rangle & & [\mathcal{T}^{[\,]}v_1/w_1,\ldots]\mathcal{T}^{w_m\cdots}x_0\ \ldots[\mathcal{T}^{[\,]}v_1/w_1,\ldots]\mathcal{T}^{w_m\cdots}x_n \\
\parallel t.4 & & \\
[\mathcal{T}^{[\,]}v_1/w_1,\ldots]\mathcal{T}^{w_m\cdots}x_0\ldots[\mathcal{T}^{[\,]}v_1/w_1,\ldots]\mathcal{T}^{w_m\cdots}x_n & &
\end{array}
$$

P.3

$$
\begin{array}{rl}
\mathcal{T}^{[\,]}(\ldots(\langle L^n(y),(v_l,\ldots,v_1)\rangle x_0)\cdots)x_n)\cdots)x_z) & \stackrel{t.1}{=} \\
\Downarrow P.3 & \\
\mathcal{T}^{[\,]}(\ldots(\langle y,(x_0,\cdots,x_n)\rangle\ x_{n+1})\cdots)x_z) & \\
\parallel t.1 & \\
\mathcal{T}^{[\,]}\langle y,(x_0,\cdots,x_n)\rangle\ \mathcal{T}^{[\,]}x_{n+1}\cdots\mathcal{T}^{[\,]}x_z & \\
\parallel t.4 & \\
[\mathcal{T}^{[\,]}x_0/w_1,\ldots,\mathcal{T}^{[\,]}x_n/w_{n+1}]\mathcal{T}^{w_{n+1}\ldots w_1}y\ \mathcal{T}^{[\,]}x_{n+1}\cdots\mathcal{T}^{[\,]}x_z &
\end{array}
$$

$$
\begin{array}{cl}
\stackrel{t.1}{=} & \mathcal{T}^{[\,]}\langle L^n(y),(v_l,\ldots,v_1)\rangle\mathcal{T}^{[\,]}x_0\cdots\mathcal{T}^{[\,]}x_n\cdots\mathcal{T}^{[\,]}x_z \\
 & \parallel t.3 \\
 & (\lambda w_1\ldots\lambda w_{n+1}.\mathcal{T}^{w_{n+1}\ldots w_1}y)\ \mathcal{T}^{[\,]}x_0\cdots\mathcal{T}^{[\,]}x_n\cdots\mathcal{T}^{[\,]}x_z \\
 & \Downarrow m.2 \\
 & [\mathcal{T}^{[\,]}x_0/w_1\ldots\mathcal{T}^{[\,]}x_n/w_{n+1}]T^{w_{n+1}\ldots w_1}y\mathcal{T}^{[\,]}x_{n+1}\cdots\mathcal{T}^{[\,]}x_z
\end{array}
$$

P.4

$$
\begin{array}{r}
\mathcal{T}^{[\,]}(\ldots(\langle L^n(y),(v_l,\ldots,v_1)\rangle x_0)\cdots)x_m) \quad \stackrel{t.1}{=} \\
\Downarrow P.4 \qquad \\
\mathcal{T}^{[\,]}(L^{n-m-1}(\langle y,(x_0,\ldots,x_m)\rangle)) \qquad \\
\| \; t.2 \qquad \\
(\lambda w_{m+1}\ldots\lambda w_{n-m-1}.\mathcal{T}^{w_{n-m}\ldots w_1}\langle y,(x_0,\ldots,x_m)\rangle) \qquad \\
\| \; t.4 \qquad \\
(\lambda w_1\ldots\lambda w_{n-m}.[\mathcal{T}^{[\,]}x_0/w_1,\ldots,\mathcal{T}^{[\,]}x_m/w_{m+1}]\ \mathcal{T}^{w_{n+1}\ldots w_1}y) \qquad
\end{array}
$$

$$
\begin{array}{ll}
\stackrel{t.1}{=} & \mathcal{T}^{[\,]}\langle L^n(y),(v_l,\ldots,v_1)\rangle\mathcal{T}^{[\,]}x_0\cdots\mathcal{T}^{[\,]}x_m \\
& \qquad \| \; t.3 \\
& \lambda w_1\ldots\lambda w_{n+1}.\mathcal{T}^{w_{n+1}\ldots w_1}y\ \mathcal{T}^{[\,]}x_0\cdots\mathcal{T}^{[\,]}x_m \\
& \qquad \Downarrow m.1 \\
& (\lambda w_{m+1}\ldots\lambda w_{n-m-1}.[\mathcal{T}^{[\,]}x_0/w_1\ldots\mathcal{T}^{[\,]}x_m/w_{m+1}]\mathcal{T}^{w_{n+1}\ldots w_1}y)
\end{array}
$$

P.5

$$
\begin{array}{ccc}
\mathcal{T}^{[\,]}\langle k,(x_1,\cdots,x_m)\rangle & \stackrel{t.4}{=} & [\mathcal{T}^{[\,]}x_1/w_1,\cdots,\mathcal{T}^{[\,]}x_m/w_m]\ \mathcal{T}^{w_m\ldots w_1}k \\
\Downarrow P.5 & & \| \; t.6 \\
\mathcal{T}^{[\,]}k & & [\mathcal{T}^{[\,]}x_1/w_1,\cdots,\mathcal{T}^{[\,]}x_m/w_m]\ k \\
\| \; t.6 & & \Downarrow m.6 \\
k & & k
\end{array}
$$

Because $\mathcal{T}$ makes no redex transposition during translation the leftmost-outermost Partial Categorical Multi-Combinator redex corresponds to the leftmost-outermost redex in the λ-Calculus with Multiple Substitutions. We proved that $\mathcal{T}$ is a correct mapping between Partial Categorical Multi-Combinators and the λ-Calculus with Multiple Substitutions, in the sense that the rewriting of the leftmost-outermost redexes in both system are equal modulo translation. We can conclude that leftmost-outermost rewritings of Partial Categorical Multi-Combinators is a normalising strategy.

Conclusions

Partial Categorical Multi-Combinators form a rewriting system, which performs the equivalent to a sequence of β-reductions in one rewriting step, but also allow for the reduction of partially parameterised functions. We showed in this paper that Partial Categorical Multi-Combinators have the Church-Rosser properties of uniqueness of normal forms and that the rewriting of the leftmost-outermost pattern leads to normal form, if it exists. Although being extremely simple Partial Categorical Multi-Combinators are slightly more complex than Categorical Multi-Combinators. Whenever performance is an important issue, such as in the implementation of functional languages, both systems can work together. Each function in the script would be compiled into both systems. If we need to reduce a partially applied function, which is not the leftmost-outermost redex, we enter the Partial Categorical Multi-Combinator code for it, allowing reduction to take place.

Acknowledgements

I express gratitude to Simon Thompson for his comments on a previous version of this paper.
Research reported herein has been sponsored jointly by the British Council, C.N.Pq. (Brazil) grants

No 40.9110/88.4. and 46.0782/89.4, and CAPES (Brazil) grant 2487/91-08.

References

[Cur86] P-L.Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Research Notes in Theoretical Computer Science. Pitman Publishing Ltd., 1986.

[Fai87] J.Fairbairn and S.Wray. TIM: A simple, lazy abstract machine to execute supercombinators. In *Proceedings of Third International Conference on Functional Programming and Computer Architecture*, pages 34–45. LNCS 274, Springer Verlag, 1987.

[Hue80] G.Huet. Confluent Reductions: Abstract Properties and Applications to Term-Rewriting Systems. *Journal of the ACM*, 27(4):797-821, October 1980.

[Joh87] T.Johnsson. *Compiling Lazy Functional Languages.* PhD thesis, Chalmers Tekniska Högskola, Göteborg, Sweden, January 1987.

[Lam80] J.Lambek. From lambda-calculus to cartesian closed categories. In J.P.Seldin and J.R.Hindley, editors, *in To H.B.Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism.* Academic Press, 1980.

[Lin86] R.D.Lins. A new formula for the execution of categorical combinators. In *Proceedings of 8th. International Conference on Automated Deduction*, pages 89–98. Springer Verlag, July 1986. LNCS 230.

[Lin87a] R. D. Lins. On the efficiency of categorial combinators as a rewriting system. *Software — Practice and Experience*, 17(8):547–559, August 1987.

[Lin87b] R.D.Lins. Categorical multi-combinators. In Gilles Kahn, editor, *Functional Programming Languages and Computer Architecture*, pages 60–79. Springer-Verlag, September 1987. LNCS 274.

[Lin90] R.D.Lins & S.J.Thompson. CM-CM: A categorical multi-combinator machine. In *Proceedings of XVI LatinoAmerican Conference on Informatics*, Assuncion, Paraguay, September 1990.

[Lin92] R.D.Lins & B.O.Lira. ΓCMC: A Novel Way of Compiling Functional Languages. *in preparation.*

[Mus91] M.A.Musicante & R.D.Lins. GMC: A Graph Multi-Combinator Machine. *Microprocessing and Microprogramming*, 31:31–35, April 1991.

[Tho92] S.J.Thompson & R.D.Lins. The categorical multi-combinator machine: Cmcm. *The Computer Journal*, April 1992.

[Sco80] D.Scott. Relating theories of the lambda-calculus. In J.P.Seldin and J.R.Hindley, editors, *in To H.B.Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism.* Academic Press, 1980.

[Tur79] D.A. Turner. A new implementation technique for applicative languages. *Software — Practice and Experience*, 9, 1979.