Some lambda calculi with categorical sums and products *

Daniel J. Dougherty

Dept. of Mathematics,
Wesleyan University,
Middletown, CT, 06459 USA.
ddougherty@eagle.wesleyan.edu

Abstract. We consider the simply typed λ-calculus with primitive recursion operators and types corresponding to categorical products and coproducts.. The standard equations corresponding to extensionality and to surjectivity of pairing and its dual are oriented as expansion rules. Strong normalization and ground (base-type) confluence is proved for the full calculus; full confluence is proved for the calculus omitting the rule for strong sums. In the latter case, fixed-point constructors may be added while retaining confluence.

1 Introduction

The systems investigated here are simply typed λ-caluli whose types include pairs, unit, sums, an empty type, and a type of natural numbers supporting constructions by primitive recursion. In the core system the types behave as categorical product and coproducts, so the subject at hand is equivalently ([LS86]) the equational theory of the free bicartesian closed category (generated by objects for the base types) with weak natural numbers object. Such λ-calculi play a role in modeling several aspects of programming languages, and we are further led to investigate the consequences of adding fixed-point operators to the systems.

An important aspect of the present treatment is that we orient certain of the standard axioms (for example, (η)) as expansion rules. The resulting calculus enjoys certain pleasant properties lacking in the traditional reduction systems.

We prove strong normalization for the core calculus; it is then easy to conclude that ground (i.e., closed, base-type) terms reduce to numerals, and from that to derive ground confluence. The core system fails to be fully confluent. When the equation characterizing sums as categorical coproducts is dropped, together with the equation for the empty type, confluence is recovered. Furthermore, when fixed-point operators are added to this system, it remains confluent. As a consequence we conclude that the equational theory involving the fixed-points is a conservative extension of the theory with primitive recursion only. It is well-known that adding fixed-point operators to the theory of coproducts is

* from the 1993 Conference on Rewriting Techniques and Applications

inconsistent [Law69]; we think the analysis here sheds some additional light on that result (see the remarks at the end of the paper).

Normalization implies, as usual, that closed normal form terms are numerals, abstractions, pairs, etc. (depending on their type). In retrospect, this makes the use of expansions seem even more natural — closed terms will reduce to expanded terms in any event, so the calculus does it explicitly.

The ground confluence result for the theory with categorical coproducts implies that the reduction system can serve as an evaluator for a programming language (with primitive recursion only) whose observational equivalences are true in the theory. The existence of (weak) sums in ordinary programming languages provides a mechanism for building data supporting computation by cases; the use of true coproducts allows the programmer to *reason about the code* by cases.

The confluence-with-fixed-points result serves as a contrast to some work of J. W. Klop. Klop has shown [Klo80] that adding surjective pairing to the *untyped* λ-calculus — with the uniqueness axiom oriented in the traditional way — spoils confluence (although unique normal forms are retained [KV89]). The terms other than the fixed-point combinator used in Klop's argument can be simply-typed, so that the argument shows that adding fixed points to the typed system with products results in a non-confluent calculus when the pairing axiom is oriented as a contraction.

The λ-calculus with pair types was shown to be strongly normalizing (SN) by deVrier in 1982 [deV87] adapting Tait's method [Tai67]. The presence of a unit type (terminal object) spoils confluence for the traditional reduction. Say that a system has *products* if the types include both pairs and a terminal object. Poigné and Voss [PV87] explored a rich calculus including products, and gave proofs of termination and confluence, but the proofs contained errors. Curien and Di Cosmo [CD91] showed confluence and termination for a second-order calculus with products.

Meanwhile Mints [Min80] considered the use of expansion rules for products and function types, and gave proofs of weak normalization and confluence for the system. Jay [Jay91], motivated by category theory, showed strong normalization and confluence for a calculus with products and a natural numbers object (supporting iteration), assuming that all types are inhabited. Independently Cubric [Cub92] repaired some errors in Mint's proof and gave applications in category theory. Akama [Aka93] has recently shown strong normalization and confluence for the general expansion lambda calculus with products.

Let us say that a calculus has *sums* if there are types behaving as weak sums (in the category-theoretic sense), but not necessarily an empty type (initial object). Gandy [Gan80] proved termination for a typed lambda calculus with sums. The present author [Dou90] and independently Okada and Scott [OS91] showed strong normalization for a calculus with products, coproducts and primitive recursion (with contracting reductions). Most recently, and independently of the present work, Di Cosmo and Kesner [DK93b] have investigated the calculus with products, sums, and full recursion (using fixed-point operators), orienting the re-

ductions as expansions, and proved strong normalization and confluence. It is remarkable that so much (independent) recent work has investigated expansions; note that the only difference between the subject of present paper and that of Di Cosmo and Kesner is our use of true coproducts. The techniques are quite different, however. We recommend [DK93a] for a very careful analysis of some of the subtleties involved in these systems, with all details provided.

In the Girard/Reynolds polymorphic lambda calculus standard data types such as pairs, sums, and lists are definable implicitly, and of course that calculus is known to be terminating. But the "uniqueness" equations which characterize pairs and sums as categorical are *not* theorems of that calculus, so those results will not apply here.

We will assume familiarity with standard notation and results in the λ-calculus [Bar84] and rewriting [DJ91].

2 Preliminaries

Definition 2.1 Fix a set of *base types*, including at least the types $\mathbf{1}$, $\mathbf{0}$, and N. The set of *types* is the closure of the set of base types under the constructions $(A \times B)$, $(A + B)$, and $(A \to B)$.

For each type T, fix an infinite set of explicitly typed *variables* and an arbitrary set of *free constants*.

The set Λ of *terms* is the closure of the variables and constants under the following constructions (write $t : T$ to assert that t is a term of type T):

I. Introduction terms

$$\lambda x.b : A \to B \text{ when } x : a \text{ and } b : B$$
$$[f_1 \;\; f_2] : (A_1 + A_2) \to B \text{ when } f_1 : A_1 \to B \text{ and } f_2 : A_2 \to B$$
$$\Box : \mathbf{0} \to A$$
$$\langle a_1, \; a_2 \rangle : A_1 \times A_2 \text{ when } a : A_1 \text{ and } b : A_2$$
$$* : \mathbf{1}$$
$$\sigma_i d : A_1 + A_2 \text{ when } d : A_i, \quad i \in \{1, 2\}$$
$$0 : N$$
$$\mathsf{succ}\; n : N \text{ when } n : N$$

II. Elimination terms

$$fa : B \text{ when } f : A \to B \text{ and } a : A$$
$$\pi_i p : A_i \text{ when } p : A_1 \times A_2, \quad i \in \{1, 2\}$$
$$\mathsf{rec}\; fan : A \text{ when } f : N \to A \to A,\ a : A, \text{ and } n : N$$

The rec constructor builds primitive recursive functions.

We have abused notation here by not decorating the constructors with type information. For example, the type of $\sigma_i d$ cannot be inferred from the type of d with our notation, but this should never cause confusion.

Introduction terms (so designated because they correspond to logical introduction rules under the Curry-Howard isomorphism) will be referred to as *I-terms* . A *numeral* is either 0 or $(\mathsf{succ}\; n)$ where n is a numeral. The *size* of a type or term is the number of operations used in constructing it. The substitution of term a for variable x in b is denoted $b[x := a]$.

In the concrete syntax parentheses will suppressed whenever possible (under the usual conventions that the function-space constructor associates right and term application associates left), terms will be considered identical if they differ only by renaming of bound variables, and type information will be omitted if it can be easily inferred.

When h is a function with domain $A_1 + A_2$ we will often have occasion to consider its respective "summands" with domains A_i. We intend to expand abstractions $h : (A_1 + A_2) \to B$ to an explicit sum of their summands. To avoid creating loops, we define the following notions.

Let $h : (A_1 + A_2) \to B$. For $i \in \{1, 2\}$ we define $h \cdot \sigma_i : A_i \to B$ by:

1. if $h \equiv [h_1 \;\; h_2]$ then $h \cdot \sigma_i \equiv h_i$,
2. if $h \equiv \lambda x.b$ then $h \cdot \sigma_i \equiv \lambda x_i.b[x := (\sigma_i x_i)]$,
3. else $h \cdot \sigma_i \equiv \lambda x_i.h(\sigma_i x_i)]$.

For function f and argument a we define $f\,!\,a$ by:

1. If $f \equiv \lambda x.b$ then $f\,!\,a \equiv b[x := a]$,
2. otherwise $f\,!\,a \equiv (fa)$.

Definition 2.2 The relation E is generated by the substitution instances of the following axioms and rules of inference. An equation is, by definition, given by a pair of terms of the same type.

I. Computational axioms:

$$
\begin{array}{ll}
(\beta) & (\lambda x.b)a = b[x := a] \\
(\sigma) & [f_1 \;\; f_2]\,(\sigma_i a) = f_i\,!\,a, \quad i \in \{1, 2\} \\
(\pi) & \pi_i\,\langle t_1,\; t_2\rangle = t_i, \quad i \in \{1, 2\} \\
(\mathsf{rec}\,) & \mathsf{rec}\, fa0 = a, \\
 & \mathsf{rec}\, fa(\mathsf{succ}\; s) = fs(\mathsf{rec}\, fas)
\end{array}
$$

II. Uniqueness axioms:

$$
\begin{array}{ll}
(\eta) & f = \lambda x.fx \;\; (x \text{ not free in } f) \\
(+!) & h = [h \cdot \sigma_1 \;\; h \cdot \sigma_2] \\
(\times!) & p = \langle pr_1 p,\; pr_2 p\rangle \\
(\mathbf{1}!) & u = * \\
(\mathbf{0}!) & f = \Box
\end{array}
$$

The equations provable in this theory are precisely those obtained by omitting the first three uniqueness axioms and adding the following rules of inference to the primary axioms:

- From $(fx) = (gx)$ infer $f = g$, when x is not free in f or h.
- From $g(\sigma_1 x_1) = h(\sigma_1 x_1)$ and $g(\sigma_2 x_2) = h(\sigma_2 x_2)$ infer $g = h$.
- From $\pi_1 p = \pi_1 q$ and $\pi_2 p = \pi_2 q$ infer $p = q$.

The rule for the sum ensures that the meaning of a term g of type $(A_1 + A_2) \to B$ is determined by its action on terms from A and from B. This is the sense in which the sum type is "categorical"; a similar remark applies to for product type.

Definition 2.3 The reduction relation R^∞ is obtained by orienting each equation in Definition 2.2 from left to right.

Write $a \Longrightarrow b$ to indicate that a reduces to b under R^∞.

Certainly R^∞ generates E, in the sense that E is the least equivalence relation containing R^∞. But R^∞ is clearly not strongly normalizing. For example, the "expanding" reductions corresponding to the uniqueness axioms can be applied indefinitely, the the terms fa and $(\lambda x.fx)a$ reduce to each other, and (η) and $(+!)$ can alternate to produce an infinite reduction sequence. The reduction we will be most interested in is obtained by imposing the obvious (local) restrictions to prohibit these chains and loops:

Definition 2.4 The reduction relation R is obtained by orienting each equation in Definition 2.2 from left to right and adding the following constraints:

in (η): f is neither of the form $\lambda y.b$ nor of the form $[f_1 \;\; f_2]$ and
the reduced occurrence of f is not applied to an argument,
in $(+!)$: h is an abstraction,
in $(\times!)$: p is not of the form $\langle p_1, \; p_2 \rangle$ and
the reduced occurrence of p is not the subject of a projection,
in $(\mathbf{1}!)$: t is not $*$,
in $(\mathbf{0}!)$: f is not $\Box$.

Write $a \longrightarrow b$ to indicate that a reduces to b under R. We abuse notation by using the same notation to refer to an equation and to the R-rule which it induces.

Proposition 2.5 *The relation* R *generates* E.

Proof. It suffices to show that if $a \Longrightarrow b$ while $a \longrightarrow b$ fails then $a \twoheadleftarrow\!\!\!\!\!\!\rightarrow b$, by induction on the size of a.

If the reduction is one of $(\times!)$, $(1!)$, $(0!)$, or an (η) whose redex is not a $[f_1 \;\; f_2]$-term, the argument is easy. Suppose $a \equiv C[\,[f_1 \;\; f_2]\,] \Longrightarrow C[\lambda x.\,[f_1 \;\; f_2]\,x] \equiv b$. Then b reduces to $C[\,[\lambda x_1.{f_1}^! x_1 \;\; \lambda x_2.{f_2}^! x_2]\,]$, and it is easy to check that each $\lambda x_i.{f_i}^! x_i \twoheadleftarrow\!\!\!\!\!\!\rightarrow f_i$, possibly using the induction hypothesis at term f_i.

Finally suppose that b is obtained by (η)-expanding a non-abstraction h. If h is an explicit sum of functions, $a \equiv b$, so consider $a \equiv C[h]$ reducing to $b \equiv C[\,[\lambda x_1.h(\sigma_1 x_1) \;\; \lambda x_2.h(\sigma_2 x_2)]\,]$. Then $C[\lambda x.hx] \twoheadleftarrow\!\!\!\!\!\!\rightarrow a$ by induction the hypothesis, and $C[\lambda x.hx]$ reduces to b directly. $\Box$

The restrictions on the expansion rules have the unfortunate consequence that the reduction is not closed under substitution. We will have to be careful about this in certain places below.

Lawvere [Law69] has pointed that one can do propositional logic in a bicartesian closed category and so cannot postulate fixed points for all maps. It will be useful to sketch the argument here. Fix any type A; the type $B = A + A$ will play the role of a boolean type, in which elements of the form $\sigma_1 u$ and $\sigma_2 v$ play the roles of "true" and "false" values, respectively. For any T define

$$\mathsf{not} : B \to B \equiv [\lambda x_2.(\sigma_2 x_2) \;\; \lambda x_1.(\sigma_2 x_1)]$$

and

$$\mathsf{if} : B \to T \to T \to T \equiv \lambda z x y.\, [\mathsf{K}x \;\; \mathsf{K}y]\, z,$$

where K abbreviates $\lambda u v.u$. Then

$$\mathsf{not}\, \sigma_1 u \longrightarrow \sigma_2 u, \;\; \mathsf{not}\, \sigma_2 u \longrightarrow \sigma_1 u, \;\; \mathsf{if}\;\; (\sigma_1 u) x y \longrightarrow\!\!\!\!\rightarrow x, \text{ and } \mathsf{if}\;\; (\sigma_2 v) x y \longrightarrow\!\!\!\!\rightarrow y.$$

A weak form of equality-testing is provided by the term

$$\mathsf{tst} : B \to B \to T \to T \to T \equiv \lambda u v x y.\mathsf{if}\, u (\mathsf{if}\, v x y)(\mathsf{if}\, v y x).$$

Then

$$\mathsf{tst}\, (\sigma_i a)(\sigma_i b) x y \longrightarrow\!\!\!\!\rightarrow x \text{ and } \mathsf{tst}\, (\sigma_i a)(\sigma_j b) x y \longrightarrow\!\!\!\!\rightarrow y \text{ when } i \neq j.$$

None of the above constructions relied on the uniqueness equations for the sum. But in the presence of +! the following holds of the function tst :

$$\mathsf{tst}\; z z x y = x \text{ and } \mathsf{tst}\; z (\mathsf{not}\; z) x y = y.$$

To prove this observe that $\lambda z.\mathsf{tst}\; z z x y$ and $\mathsf{K}x$ have a common reduct, as do $\lambda z.\mathsf{tst}\; z (\mathsf{not}\; z) x y$ and $\mathsf{K}y$.

If we now postulate the existence of fixed-point operators, so that every term has a fixed point, let w be a fixed point of not . Then for any x and y,

$$\mathsf{tst}\; w w x y = x = \mathsf{tst}\; w (\mathsf{not}\; w) x y = y$$

and the theory is inconsistent.

The technique developed in section 4 will shed some additional light on this situation.

3 Strong Normalization

In this section we show that every sequence of R-reductions terminates.

An individual term t will be said to be strongly normalizing (in any of the reduction systems we consider) if every sequence of reductions out of t terminates. If a term is not strongly normalizing, we will say it is *infinite*.

Some preliminary observations will be helpful. It is not obvious even that simple variables are SN, since non-base-type variables can undergo expansions. On the other hand, if t is an I-term other than an abstraction whose immediate subterms are SN, then t is SN — it is straightforward to prove this using the fact that root expansions never apply to such terms. The situation with λ-abstractions is more delicate, since they can undergo (+!)-reductions.

To handle abstractions we require the following notion.

Definition 3.1 Given a type T, the *pseudo-variables* $PV(T)$ of T are the variables of type T, together with, in case $T \equiv T_1 + T_2$, the set $\{\sigma_i p_i \mid p_i \in PV(T_i),\ i \in \{1,2\}\}$.

Lemma 3.2 *Let U be a type such that $PV(U) \subseteq SN$. Then $\lambda x.b : U \to V$ is SN provided $\{b[x := p] \mid p \in PV(U)\} \subseteq SN$.*

Proof. Easy (use induction on U). □

Definition 3.3 The set of *computable* terms of type T is defined by induction on T. The term $t : T$ is computable if t is strongly normalizing, and if furthermore

1. if $t \longrightarrow\!\!\!\!\!\rightarrow \lambda x.b$ then for all computable a, $b[x := a]$ is computable,
2. if $t \longrightarrow\!\!\!\!\!\rightarrow [f_1\ \ f_2]$ then each f_i is computable,
3. if $t \longrightarrow\!\!\!\!\!\rightarrow \langle t_1,\ t_2\rangle$ then t_1 and t_2 are each computable, and
4. if $t \longrightarrow\!\!\!\!\!\rightarrow \sigma_i a$ then A is computable.

Let $\mathcal{C}^T$ denote the computable terms of type T, and set $\mathcal{C}$ to be $\bigcup\{\mathcal{C}^T \mid T \text{ a type.}\}$

Tait [Tai67] originated the strategy of using an inductively defined predicate such as computability to prove termination in the λ-calculus. Prawitz [Pra70] pointed out the possibility of basing a notion of computability, there termed *validity*, based on I-terms rather than on E-terms (as Tait's method is), having observed that the latter approach breaks down when sum types are involved. He proves termination of a certain calculus by a method based on I-terms but slightly different from the definition above. The reductions in that calculus include "commuting reductions" for sum terms, inspired directly by proof theory rather than computation, and do not include reductions corresponding to "uniqueness" equations.

We will use the following observations about computability below. A term of base type is in $\mathcal{C}$ precisely when it is SN. A non-abstraction I-term is in $\mathcal{C}$ if all of its immediate subterms are in $\mathcal{C}$ — this submits to same sort of argument as the corresponding remark about SN. The set $\mathcal{C}$ is closed under reduction, so (since every computable term is SN) we will be justified in proofs by induction over maximal-reduction-length for terms in $\mathcal{C}$.

When t is SN, define $\sharp t$ to be the maximum length of a reduction path out of t.

Notation 3.4 Let us call the rules arising from the uniqueness axioms *expansion* rules. (Note that an "expansion" is still a reduction, as a rule in the calculus). A *root* reduction of a is a reduction whose redex is a itself; a *proper* reduction is one which is not a root expansion.

The following lemma establishes the key facts we need for the termination proof. The importance of the second assertion in the lemma was highlighted by Girard in his proof of termination for the λ-calculus with polymorphic types. The crucial point here is the restriction to proper reductions.

Lemma 3.5 *For each T:*

1. *$\lambda x.b : T$ is computable if for all computable a, the term $b[x := a]$ is computable,*
2. *if $t : T$ is not an I-term and if each proper one-step reduct of t is computable, then t is computable,*
3. *$PV(T) \subseteq \mathcal{C}$,*
4. *$\pi_i p : T$ is computable if p is computable, and*
5. *if $f : T$ is computable then for all computable a, (fa) is computable.*

Proof.

The proof is by induction on T, and it importanmt that we prove the clauses in the order stated.

Clause 1. Write $T \equiv U \to V$, so that $x : U$ and $b : V$. First note that $\lambda x.b$ is SN by Lemma 3.2, using (3) at type U.

Next, suppose $U \equiv U_1 + U_2$ and $\lambda x.b \longrightarrow\!\!\!\!\!\rightarrow [\lambda x_1.b_1 \;\; \lambda x_2.b_2]$; we want to show that each $\lambda x_i.b_i$ is computable at type $U_i \to V$. Without loss of generality the reduction must have begun with a $(+!)$-reduction, and so, since $\mathcal{C}$ is closed under reduction it suffices to consider the case where each b_i is $b[x := \sigma_i x_i]$. By induction, it suffices to show that for each $d \in \mathcal{C}^{U_i}$, $b_i[x_i := d]$ is computable; but this is simply $b[x := \sigma_i d]$, computable by assumption on b.

Finally, we wish to show that if $\lambda x.b \longrightarrow\!\!\!\!\!\rightarrow \lambda x.c$ then for any computable a, $c[x := a]$ is computable.

Let us introduce the notation $\mathcal{C}_x^V$ for the set of terms t such that $t[x := a] \in \mathcal{C}^V$ whenever $a \in \mathcal{C}^U$ (remember that $x : U$).

Now suppose we can show that for $a \in \mathcal{C}^U$, $(\lambda x.b)a \in \mathcal{C}V$. Then we can simply observe that $(\lambda x.b)a \longrightarrow\!\!\!\!\!\rightarrow (\lambda x.c)a \longrightarrow c[x := a]$. We concentrate, then, on showing $(\lambda x.b)a \in \mathcal{C}V$.

We may use (2) at type V, and be content with showing that all proper reducts are computable. Furthermore b and a are SN, and so we may argue by induction over $\sharp b + \sharp a$, *provided* we know that $\mathcal{C}_x^V$ is closed under reduction. The reducts are of the form $(\lambda x.b')a$, $(\lambda x.b)a'$, or $b[x := a]$; the first two are computable by induction hypothesis, the latter by assumption on b. But it is not clear that $\mathcal{C}_x^V$ is in fact closed under reduction. The remainder of the argument is devoted to verifying this.

Consider a reduction $b \equiv C[u] \longrightarrow C[u'] \equiv c$ in which u is the redex. To argue that $c \in \mathcal{C}_x^V$, choose a computable term a to replace x in c; we want $c[x := a] \in \mathcal{C}^V$. Now $b[x := a] \in \mathcal{C}^V$, and if $b[x := a] \longrightarrow c[x := a]$ we can simply use the fact that $\mathcal{C}$ is closed under reduction. Of course, $b[x := a] \longrightarrow c[x := a]$ will not hold if one of the expansion-restrictions is violated. So we restrict attention to this situation.

Now (since $b \equiv C[u] \longrightarrow C[u'] \equiv c$) the way in which $b[x := a] \longrightarrow c[x := a]$ can fail is for a restriction as to the form of the redex to be violated . Indeed, it must be the case that u is in fact the variable x, the original $b \longrightarrow c$ was an expansion of x, and the term a was either $*$, $\Box$, $\langle a_1,\ a_2 \rangle$, $\lambda y.d$, or $[f_1 \;\; f_2]$, according to the expansion rule.

We claim that the expansions a' of a are themselves computable and that $a' \longrightarrow\!\!\!\!\!\rightarrow a$. If so, then $b[x := a']$ is computable by assumption on b, and $c[x := a]$ is either $b[x := a']$ or (in case there are occurrences of x other than as the redex in b) obtained by reduction from $b[x := a']$. Then we are done.

Actually, in case $a \equiv *$, $a \equiv \Box$, or $a \equiv [f_1\ f_2]$ we have $a \equiv a'$ and there is nothing to prove.

Taking the other cases in turn (note that a has type U and so we may invoke the current Lemma there): if $a \equiv \langle a_1,\ a_2\rangle$ the corresponding expansion is $\langle \pi_1 \langle a_1,\ a_2\rangle,\ \pi_2 \langle a_1,\ a_2\rangle \rangle$. But this is computable by part (4) and the remarks following Definition 3.3. If $a \equiv \lambda y.d$ the corresponding expansion is $\lambda x.(\lambda y.d)x$, and this is computable by (1) and then (5). The fact that $a' \longrightarrow\!\!\!\!\!\rightarrow a$ is easily verified.

Clause 2. Since t is neutral, it is clear that if *every* one-step reduction of t were computable, then t would be computable. So it suffices here to show that each 1-step root expansion of t is computable. When T is either a base type or a sum, there is then nothing to prove.

When $T \equiv T_1 \times T_2$ we consider the term $\langle \pi_1 t,\ \pi_2 t\rangle$. It suffices to show that each $\pi_i t$ is computable. By the induction hypothesis it suffices to show that an arbitrary proper reduct of $\pi_i t$ is computable. Since t is not an I-term such a reduction is of the form $\pi_i t'$, where $t \longrightarrow t'$ via a proper reduction by the context restriction on expansions. Thus t' is computable by hypothesis on t. Part (4) at type T_i then tells us $\pi_i t'$ is computable, as desired.

When $T \equiv U \to V$ we consider $\lambda x.tx$. By (1) it suffices to see that for $a \in \mathcal{C}^U$, $ta \in \mathcal{C}^V$. By (2) at type V we may examine the proper reducts and induct over $\sharp a$. But since t is neutral the argument is easy.

Clause 3. When T is a base type, use (2) together with the fact that there are no improper reductions out of a base-type variable. Otherwise use the fact that the pseudo-variables are generated by the σ_i contexts, which preserve computability.

Clause 4. Induct on $\sharp p$. By (2) it suffices to show that each proper reduct of $\pi_i p$ is in $\mathcal{C}$. There are two forms such a reduction can take. One is a proper reduction $p \longrightarrow p'$ inside of p, which yields $\pi_i p'$, in $\mathcal{C}$ by the induction hypothesis. The other is a root reduction, in case $p \equiv \langle p_1,\ p_2\rangle$, yielding p_i, which is in $\mathcal{C}$ since p was.

Clause 5. Induct on $\sharp f + \sharp a$. Invoking (2), we examine the proper reducts of fa; non-root reductions submit to the induction hypothesis. There are two possible root reductions, in the cases $f \equiv (\lambda x.b)$ and $f \equiv [f_1\ f_2]$. In each case the result is computable by hypothesis on f.

□

Corollary 3.6 *If f, a, and n are each computable then* $\mathsf{rec}\, fan$ *is computable.*

Proof. It suffices to show that all proper reducts are computable; we show this by induction on $\sharp f + \sharp a + \sharp n$, with a secondary induction on the length of the normal form of n. The only interesting reduction is $\mathsf{rec}\, fa(\mathsf{succ}\ n) \longrightarrow fn(\mathsf{rec}\, fan)$. The

last argument is computable since the normal form of n is smaller than the normal form of $\mathsf{succ}\, n$, now use (5) of the previous Lemma. □

Theorem 3.7 R *is SN.*

Proof. It suffices to show that each term is computable. Using the standard trick, define $\mathcal{C}^{Vars}$ to be the set of all substitutions θ such that for every x, $\theta x \in \mathcal{C}$; and let $\mathcal{C}^*$ be the set of terms t such that for all $\theta \in \mathcal{C}^{Vars}$, $\theta t \in \mathcal{C}$. We show that all terms t are in $\mathcal{C}^*$ by induction on t. This suffices since variables are computable and thus the identity substitution is in $\mathcal{C}^{Vars}$.

Choose $\theta \in \mathcal{C}^{Vars}$. When t is a variable $\theta t \in \mathcal{C}$ by definition of $\mathcal{C}^*$; when t is a constant $\theta t \equiv t$, and so is in $\mathcal{C}$ by Lemma 3.5 (2).

When $t \equiv \lambda x.b$ then (since we may assume that x is not in the domain of θ) $\theta t \equiv \lambda x.\theta b$, and by Lemma 3.5 it suffices to show that for any $a \in \mathcal{C}$, $b[x := a] \in \mathcal{C}$. But $b[x := a]$ is $\theta' b$ where $\theta' \equiv \theta \cup \{x \mapsto a\}$, and this is a substitution in $\mathcal{C}^{Vars}$. So by the induction hypothesis for b, $b[x := a] \in \mathcal{C}$ as desired.

In every other case, the substitution θ filters down to the immediate subterms of t and the result can be seen to be computable by invoking the induction hypothesis and applying Lemma 3.5 or the remarks following Definition 3.3. □

Corollary 3.8 *When there are no constants other than 0, every closed term reduces to an I-term.*

Proof. By induction on terms; by strong normalization it suffices to show that no closed non-I term is irreducible. The argument is straightforward. □

4 Confluence

Proposition 4.1 *When there are no constants other than 0,* R *is confluent on closed terms of base type.*

Proof. It suffices to show that closed base-type normal forms are equal only if they are identical. But the terms in question are the numerals, and the full set-theoretic type hierarchy generated by the standard natural numbers provides a model for the equality theory, so distinct numerals can never be proved equal. □

Remark 4.2 R is not confluent.

Proof. Let $x : A + A$ and observe that

$$x = [\lambda x_1.(\sigma_1 x_1)\ \ \lambda x_2.(\sigma_2 x_2)]\, x$$

is provable in E, but each term is irreducible (if A is not a sum).

Abstracting over x gives a closed example. A similar example shows that the presence of rule (0!) also blocks confluence. □

We are led to consider the theory of sums, obtained by deleting equations (+!) and (0!). The natural step in constructing a corresponding reduction system is to use R without (+!) and (0!) rules. We can further and relax the restriction on (η) by allowing the redex to be be an explict sum of functions.

In fact, rule (+!) was used in demonstrating that

$$[f_1 \ \ f_2] = \lambda x.\, [f_1 \ \ f_2]\, x$$

was admissible in the theory generated by R, so we *must* relax the restriction on (η).

Definition 4.3 The equational theory E^- is generated by the axioms of Definition 2.2 with (+!) and (0!) omitted.

The reduction relation $\mathsf{R}^{\bowtie}$ is obtained by orienting each equation in E^- from left to right. Write $a \stackrel{-}{\Longrightarrow} b$ for this relation.

The reduction relation R^- is generated by the rules of Definition 2.4 with (+!) and (0!) omitted, and with the first restriction on rule (η) revised to read simply "f is not of the form $\lambda y.b$". Write $a \stackrel{-}{\longrightarrow} b$ for this relation.

It is not hard to see that R^- generates E^-. The main results of this section are that R^- is confluent and furthermore remains confluent when fixed-point operators are added.

The technique makes use of the strong normalization of R^-, but this does not follow from the results of the previous section since R^- is not a sub-system of R. The proof is a slight modification of the arguments of the last section, so we will just give an outline:

Theorem 4.4 R^- *is* SN.

Proof (outline). The analogue of Lemma 3.2 is the easy observation that $\lambda x.b$ is SN if b is — abstractions undergo no expansions in the present system. The definition of *computable* terms is precisely the same as in Definition 3.3, and most of the remarks about the computability of I-terms still hold, with the exception that it is no longer clear that $[f_1 \ \ f_2]$ is computable when the f_i are (such terms may now enjoy root (η) expansions).

So we add this as a final assertion in the main Lemma. Clearly $[f_1 \ \ f_2]$ is SN. The argument will reduce to showing (since $[f_1 \ \ f_2] \stackrel{-}{\longrightarrow} \lambda x.\, [f_1 \ \ f_2]\, x$) that for computable a, $[f_1 \ \ f_2]\, a$ is computable. A routine application of (2) suffices. □

The system R^- has better substitutivity properties than R.

Proposition 4.5 *1. If* $a \stackrel{-}{\Longrightarrow} b$ *then either* $a \stackrel{-}{\longrightarrow} b$ *or* $b \stackrel{-}{\longrightarrow} a$.

2. If $a \stackrel{-}{\longrightarrow} a'$ *then for all* t $t[x := a]$ *and* $t[x := a']$ *have a common* $\stackrel{-}{\longrightarrow}$*-reduct.*

3. If $t \stackrel{-}{\longrightarrow} t'$ *then for all* a, *either*

(a) $t[x := a] \stackrel{-}{\longrightarrow} t'[x := a]$ *or*

(b) $t'[x := a] \stackrel{-}{\longrightarrow} t[x := a]$.

Proof. Part (1) is an easy examination of cases. For (2) first note that if $a \xrightarrow{-} a'$ then $a \xRightarrow{-} a'$. Since there are no restrictions on $\mathsf{R}^{\bowtie}$ it is easy to see that $t[x := a] \xRightarrow{-} t[x := a']$, by rewriting the various occurrences of a to a'. Now use (1), but note that the direction of the $\xrightarrow{-}$-reductions may be different for different occurrences of x. Part (3) is similar, and easier.

□

Theorem 4.6 R^{-} *is confluent.*

Proof. It suffices to establish local confluence. Since R^{-} must obey context constraints we cannot, *a priori*, restrict attention to finding common reducts for the critical pairs. We proceed by induction on terms t to show that

If $t \xrightarrow{-} u$ and $t \xrightarrow{-} v$ then there exists w such that $u \xrightarrow{-} w$ and $v \xrightarrow{-} w$.

First consider the case in which $t \xrightarrow{-} u$ is a root expansion. If $t \xrightarrow{-} u$ is $t \xrightarrow{-} \lambda x.tx$ then, writing t' for v, note that $t' \xrightarrow{-} \lambda x.t'x$ and that $\lambda x.tx \xRightarrow{-} \lambda x.t'x$. Proposition 4.5 yields the desired w. The case of a $(\times!)$-expansion is similar.

We may assume then that neither $t \xrightarrow{-} u$ nor $t \xrightarrow{-} v$ is a root expansion. If t is an I-term it is easy to construct a common reduct of u and v. The other cases are straightforward unless $t \equiv fa$, u is $f'a$ and v is fa'. Here consider the term $f'a'$. We can be sure that $f'a \xrightarrow{-} f'a'$ since there are no context restriction on rewriting an argument; the fact that $fa' \xRightarrow{-} f'a'$ and an application of Proposition 4.5 yield our w.

We are left with the case in which u is obtained by a root non-expansion from fa. We discuss only the case $fa \equiv (\lambda x.b)a \xrightarrow{-} b[x := a] \equiv u$. The two possibilities for v are $(\lambda x.b')a$ and $(\lambda x.b)a'$; these reduce, respectively, to $b'[x := a]$ and $b[x := a']$. A final application of Proposition 4.5 completes the proof. □

We conclude that the unrestricted system is confluent as well:

Corollary 4.7 $\mathsf{R}^{\bowtie}$ *is confluent.*

Proof. $\mathsf{R}^{\bowtie}$ and R^{-} generate the same equational theory, and R^{-} is a subsytem of $\mathsf{R}^{\bowtie}$. □

Motivated by the interpretation of λ-calculi as theories of programming languages, we next consider the consequences of adding fixed-point operators to Λ. As described in section 2, adding fixed-point operators to E yields an inconsistent theory, so we certainly cannot expect confluence there. We again turn to E^{-}.

It is easy to see that adding fixed-point operators to E^{-} yields a consistent theory — indeed the familiar category of complete partial orders (with either separated or coalesced sums) is cartesian closed (but not bicartesian closed) and so provides models. But a confluence result gives more information, of course. As described in the introduction, the traditional system for products and fixed point fails to be confluent. We show now in contrast that R^{-} with fixed points added remains confluent.

Definition 4.8 Assume that for each type T other than $\mathbf{0}$ there is a distinguished constant $\varphi_T : (T \to T) \to T$. The reduction system R^{φ} is obtained from R^{-} by adding the following rules:

$$(\varphi) \quad \varphi_T f \xrightarrow{\varphi} f(\varphi_T f).$$

For emphasis, when fixed-point constants have been distiguished we designate the set of terms by Λ^{φ}. We will henceforth supress the type subscript on the φ_T.

We will analyze the behavior of this new calculus by simulating the fixed-point operators by operators providing *bounded* recursion. This technique apparently originates with Lévy, and was used by Poigné and Voss [PV87] (I am indebted to Roberto Di Cosmo and Delia Kesner for this reference).

Definition 4.9 Assume that for each type T other than $\mathbf{0}$ and each natural number n there is a distinguished constant $\varphi^n_T : (T \to T) \to T$. The reduction system $\mathsf{R}^{\#}$ is obtained from R^{-} by adding the following rules (omitting, as usual, type tags):

$$(\varphi^n) \quad \varphi^{n+1} f \xrightarrow{\#} f(\varphi^n f).$$

In this setting we will designate the set of terms by $\Lambda^{\#}$.

Lemma 4.10 *$\mathsf{R}^{\#}$ is SN and confluent.*

Proof. We can code the φ^n as terms $\overline{\varphi^n}$ in the ordinary R^{-} system by letting $\overline{\varphi^0}$ be an arbitrary free constant and $\overline{\varphi^{n+1}}$ be $\lambda f.f(\overline{\varphi^n} f)$. Then a reduction in $\mathsf{R}^{\#}$ induces a corresponding reduction in R^{-}, hence is finite. Note that there were no restrictions on the set of free constants in the SN proof.

For confluence, it suffices to check local confluence (observe that a naive argument based on the coding above will not suffice). The argument is just as for the proof of Theorem 4.6; the additional cases due to the φ^n cause no difficulty. □

We now show how to lift the previous result to obtain CR for the pure R^{φ} system.

Definition 4.11 Given t from $\Lambda^{\#}$, the *erasure* $|t|$ of t is obtained by replacing each occurrence of a φ^n by φ.

It is easy to prove an "erasing lemma" showing that if $t \xrightarrow{\#}\!\!\!\!\!\rightarrow s$ then $|t| \xrightarrow{\varphi}\!\!\!\!\!\rightarrow |s|$.

The following "lifting" lemma shows that any computation can be simulated by the bounded recursors. Say that the *index* of a term in $\Lambda^{\#}$ is the least n such that some φ^n occurs in the term (or 0, if no φ^n appear).

Lemma 4.12 *Let $t \in \Lambda^{\#}$ have index k and suppose that*

$$|t| \xrightarrow{\varphi}\!\!\!\!\!\rightarrow u \text{ in no more than } k \text{ steps.}$$

Then there exists $s \in \Lambda^{\varphi}$ such that $t \xrightarrow{\#}\!\!\!\!\!\rightarrow s$ and $|s| \equiv u$.

Proof. It suffices to consider a 1-step reduction under the hypothesis that k is at least 1, provided we show that in this case the index of the resulting s is at least $k-1$.

But s can be determined by applying the same reduction (in the obvious sense) out of t as the given reduction out of $|t|$. Specifically, (and assuming for the sake of ease of notation that the given reduction is at the root of t) the relevant fact is that if $|t|$ is of the form θl where $l \xrightarrow{\varphi} r$ is one of the rules of R^{φ}, then t will be of the form $\theta' l'$ where the erasures of l' and θ' (pointwise) are l and θ. The term s will then be $\theta' r$. It is clear that the index is either unchanged or diminished by one.

It is just here that we use the fact that all of our reduction rules are left-linear. For example, if the rule for surjective pairing were oriented as $\langle \pi_1 p,\ \pi_2 p\rangle \longrightarrow p$ then the left-hand side could arise as the erasure of a term t without t being a redex. □

Theorem 4.13 R^{φ} *is confluent.*

Proof. Suppose $a \xrightarrow{\varphi}\!\!\!\!\rightarrow b_1$ and $a \xrightarrow{\varphi}\!\!\!\!\rightarrow b_2$; we seek c such that $b_1 \xrightarrow{\varphi}\!\!\!\!\rightarrow c$ and $b_2 \xrightarrow{\varphi}\!\!\!\!\rightarrow c$. Let n be the maximum of the lengths of the given reductions, and construct $a^{\#} \in \Lambda^{\#}$ from a by replacing each occurrence of φ by φ^n.

By the lifting lemma there are $b_1^{\#}$ and $b_2^{\#}$ such that $a^{\#} \xrightarrow{\#}\!\!\!\!\rightarrow b_1^{\#}$ and $a^{\#} \xrightarrow{\#}\!\!\!\!\rightarrow b_2^{\#}$ and $|b_i^{\#}| \equiv b_i$.

By the confluence of $\mathsf{R}^{\#}$ there is a $c^{\#}$ such that for each i, $b_i^{\#} \xrightarrow{\#}\!\!\!\!\rightarrow c^{\#}$. Apply the erasing lemma to produce $|c^{\#}|$ as the desired c. □

Corollary 4.14 *Adding fixed-point operators yields a conservative extension of* E^{-}.

□

As a final note we reconsider the original theory E. The construction above strengthens Remark 4.2 by showing that no *ground confluent* left-linear reduction system can be constructed for a functionally complete theory of coproducts, without some restriction on the free constants allowed. If such a system existed, we could build in the bounded recursors, show preservation of confluence in the presence of true fixed-point operators, and derive the above Corollary for the full theory. But the argument in section 2 shows that this is a contradiction.

Acknowledgements.

Roberto Di Cosmo and Delia Kesner pointed out an error in an early version of Proposition 4.5. I am grateful to Ramesh Subrahmanyam for several enlightening discussions.

References

[Aka93] Y. Akama. On Mint's Reduction for ccc-Calculus. *Proc. Typed Lambda Calculus and Applications* 1993.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Volume 103 of *Studies in Logic and the Foundations of Mathematics*, North-Holland, Amsterdam, 1981. Revised edition, 1984.

[CD91] P-L Curien, R. Di Cosmo. A confluent reduction system for the λ-calculus with surjective pairing and terminal object, in Leach et. al. (eds.), *Int'l. Conf. on Automata, Languages, and Programming*, LNCS 510, 291–302, Springer Verlag 1991.

[Cub92] D. Cubric. Embedding of a free cartesian closed category into the category of Sets. Manuscript, McGill University 1992.

[DK93a] R. Di Cosmo, D. Kesner. Simulating expansions without expansions. Tech Rep, INRIA 1993.

[DK93b] R. Di Cosmo, D. Kesner. *A confluent reduction system for the extensional λ-calculus with pairs, sums, recursion and terminal object.* Proc ICALP 1993.

[DJ91] N. Dershowitz, J.-P. Jounnaud. Term Rewriting Systems, in *Handbook of Theoretical Computer Science*, 243–320, North-Holland, Amsterdam, 1991.

[Dou90] D. J. Dougherty. Some reduction properties of a λ-calculus with categorical sums and products. Manuscript, Wesleyan University 1990.

[Gan80] R. O. Gandy. Proofs of strong normalization, in J. P. Seldin and J. R. Hindley (eds.) *To H. B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, New York, 1980.

[Jay91] C. Barry Jay. Long β-normal forms and confluence. Manuscript, University of Edinburgh, 1991.

[Klo80] J. W. Klop. *Combinatory Reduction Systems.* Mathematical Centre Tracts, v. 127. Centre for Mathematics and Computer Science, Amsterdam 1980.

[KV89] J. W. Klop, R.C. deVrijer. Unique normal forms for lambda calculus with surjective pairing. *Information and Computation*, v.80 no. 2, 97–113, 1989.

[LS86] J. Lambek, P. Scott. *Introduction to Higher-order Categorical Logic.* Cambridge Studies in Advanced Mathematics 7. Cambridge University Press 1986.

[Law69] F. W. Lawvere. Diagonal arguments and cartesian closed categories, in *Category Theory, Homology Theory, and Their Applications II*, LNM 92, Springer-Verlag, 1969.

[Min80] G. E. Mints. Category Theory and Proof Theory. In *Aktualnie Voprosi Logiki i Metodologiinauki* (Russian) 252–278. Kiev, 1980.

[OS91] M. Okada, P. Scott. Rewriting theory for uniqueness conditions: coproducts. Talk presented First Montreal Workshop on Programming Language Theory, April 1991.

[Pra70] D. Prawitz. Ideas and Results in Proof Theory, in J. E. Fenstad, ed., *Proceedings of the Second Scandinavian Logic Symposium.* North-Holland, Amsterdam, 1971.

[PV87] A. Poigné, J. Voss. On the implementation of Abstract Data Types by Programming Language Constructs. *Journal of Computer and System Science* **34** (1987), 340–376.

[Tai67] W. W. Tait. Intensional interpretation of functionals of finite type I, *J. Symbolic Logic* **32**, pp. 198-212, 1967.

[deV87] R. C. de Vrijer, Strong Normalization in $N - HA^{\omega}$. *Proc. Koninklijke Nederlandse Akademie van Wetenschappen* Series A, 90/4, pp. 473-478, 1987.